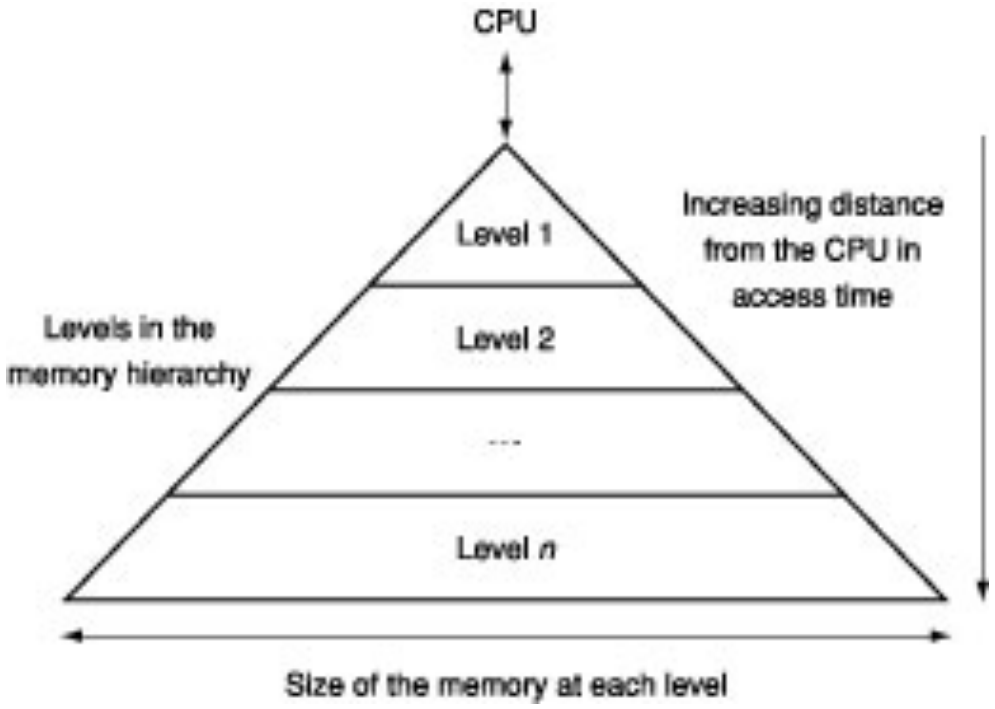
Jerarquía de Memoria

Rodrigo Baessa

Figuras: "Computer Organization and Design" 3 Ed., Patterson, D. y Henessy, J. Morgan Kaufmann 2005

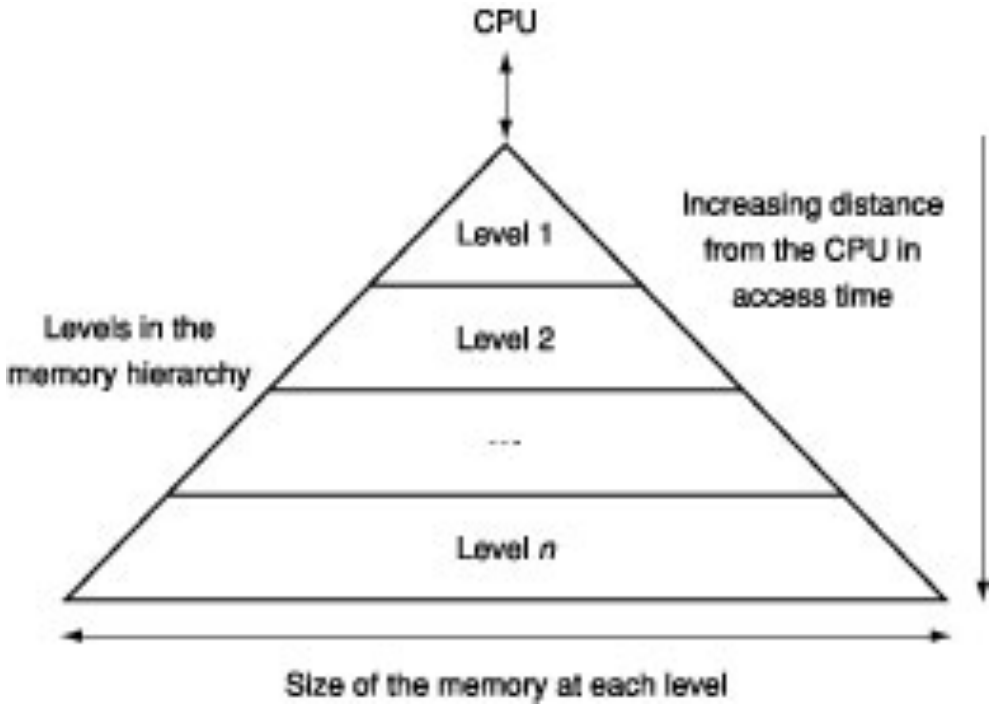
Jerarquía de Memoria

En una jerarquía de memorias, tenemos distintos tipos de memoria que conforman nuestro sub-sistema de memoria. Estas están organizadas en una forma piramidal.



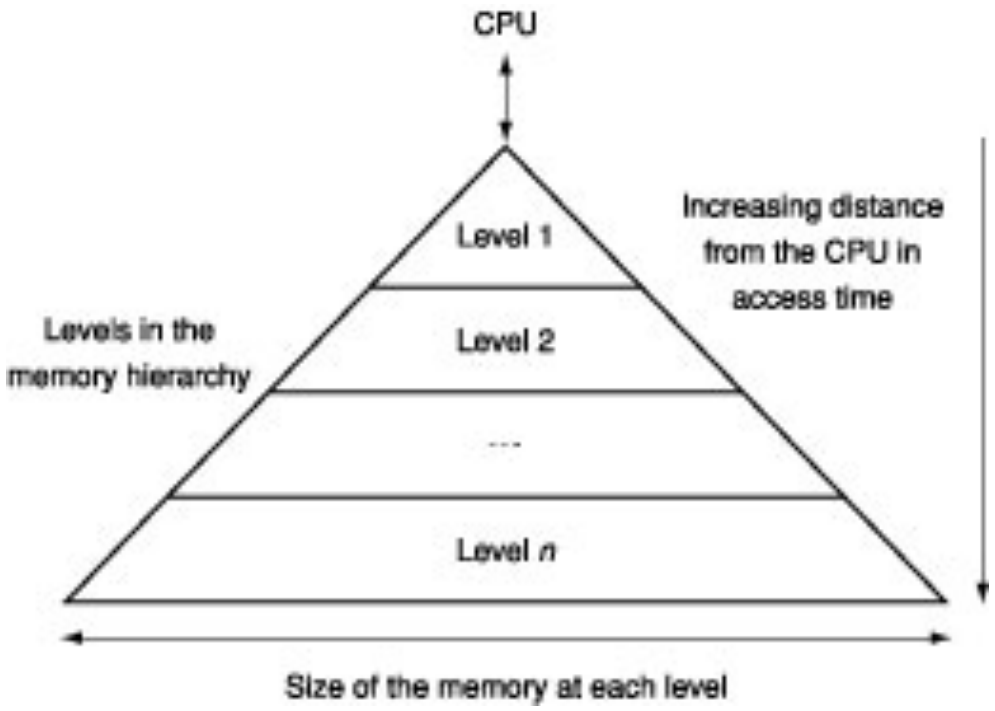
Jerarquía de Memoria

Tenemos memoria muy rápida (pero muy cara y por lo tanto pequeña) inmediatamente al lado del CPU (Level 1). El CPU intenta realizar todos sus accesos a esta memoria.



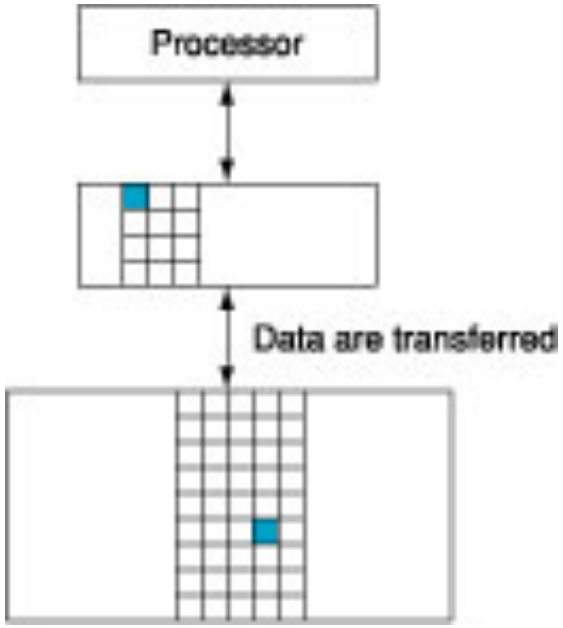
Jerarquía de Memoria

A medida que nos alejamos del CPU las memorias son más grandes y baratas, pero más lentas.



Jerarquía de Memoria

Si un dato o instrucción no se encuentra en un nivel i, se busca en uno mayor ($i+1$) hasta que se encuentre. Se copia un bloque de esta la memoria a la memoria de un nivel más bajo sucesivamente hasta llegar al nivel 1.



Jerarquía de Memoria

Al emplear este esquema, el CPU ve al sub-sistema de memoria como una memoria muy grande y rápida.

Sin embargo, para que una jerarquía de memoria funcione como deseamos se debe de cumplir el principio de ***localidad*** (temporal y espacial).

Localidad

Localidad Temporal: Si accesamos un determinado dato o instrucción es muy probable que este sea accesado otra vez en un tiempo corto.

Localidad Espacial: Si accesamos un determinado dato o instrucción es muy probable que el próximo que se accese esté en una dirección cercana.

Caching

Las memorias de los primeros niveles se conocen como *caches*. El manejo de estos es por medio de *hardware*. El CPU siempre (con algunas excepciones) tratará de buscar un dato o instrucción en el cache (nivel 1).

Caching

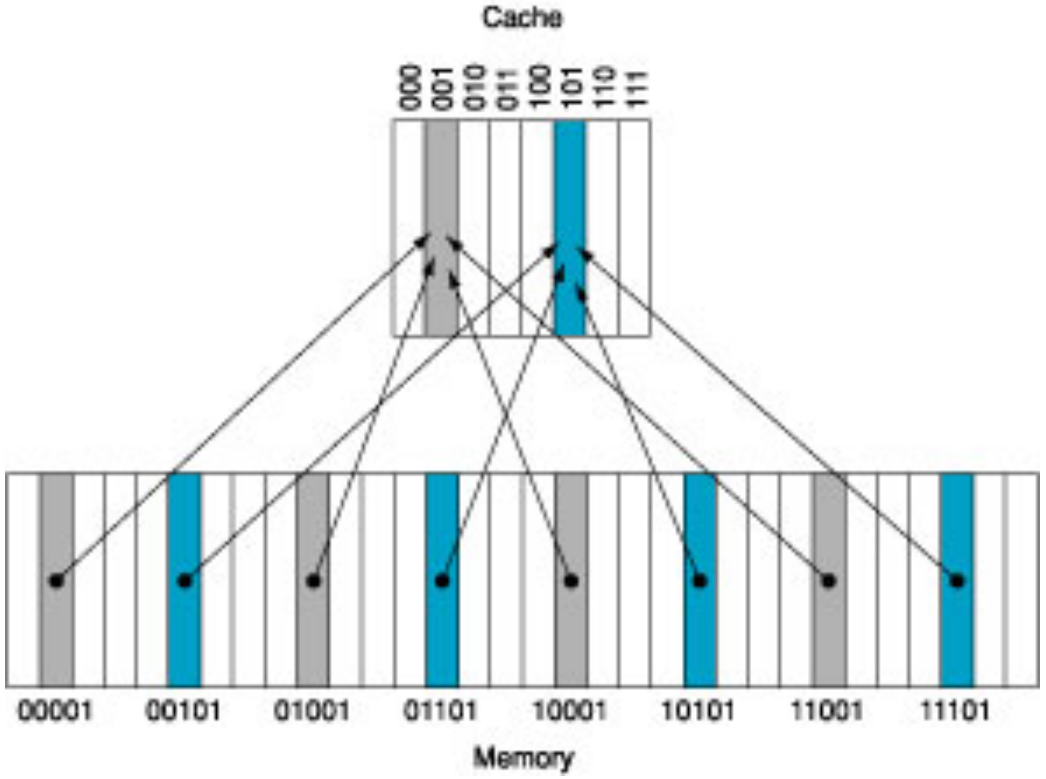
Si se encuentra dentro del cache el dato o instrucción que buscamos, ocurre un ***cache hit***.

Si no se encuentra se genera un ***cache miss***. En este caso, se buscará en el siguiente nivel de la jerarquía hasta encontrarlo. Al encontrarlo, se copiará un bloque (no solo el dato buscado) al cache. Este bloque se conoce como *line* o *block*.

Cache Mapping

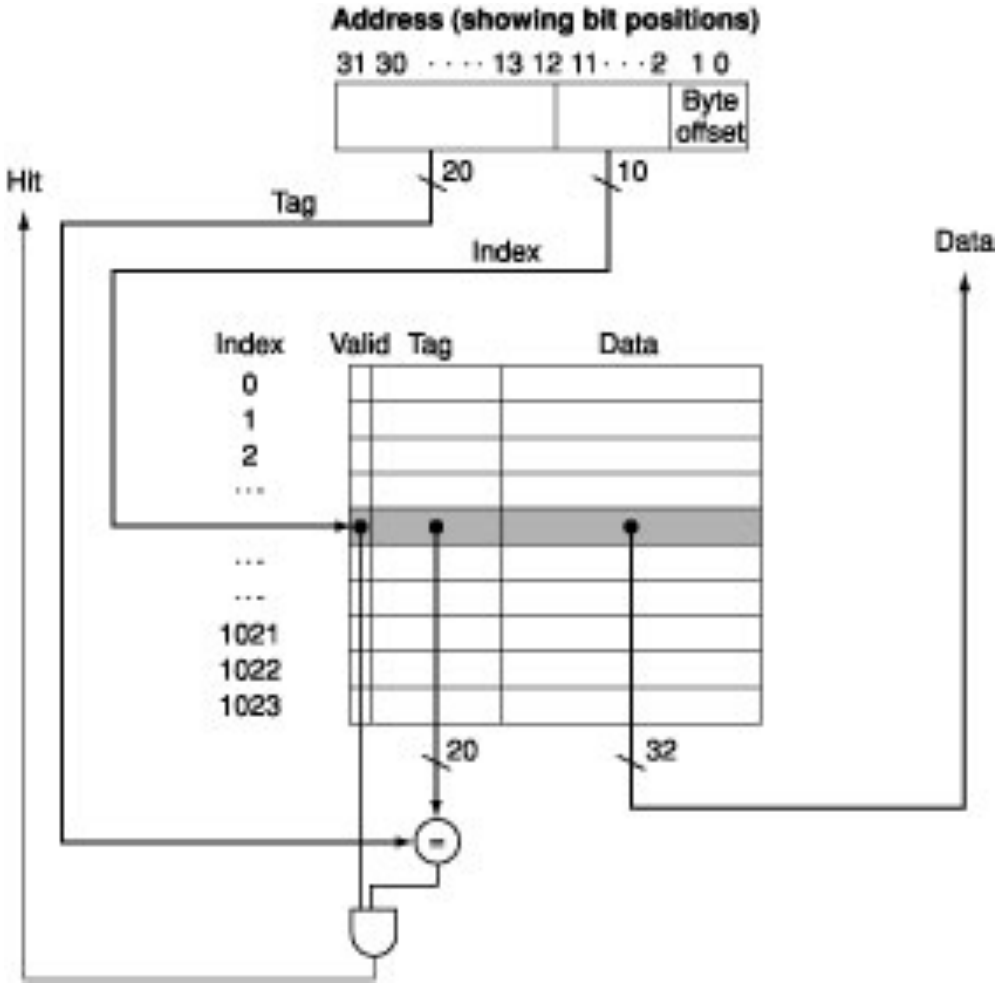
- ¿Cómo sé si esta el dato que busco en el cache?

En el esquema más simple, un dato solo puede escribirse en un determinado lugar del cache. Este esquema se conoce como *Direct Mapped*. Se utilizan los bits menos significativos de la dirección para determinar la línea del cache que le corresponde, este número se conoce como el *cache index*.



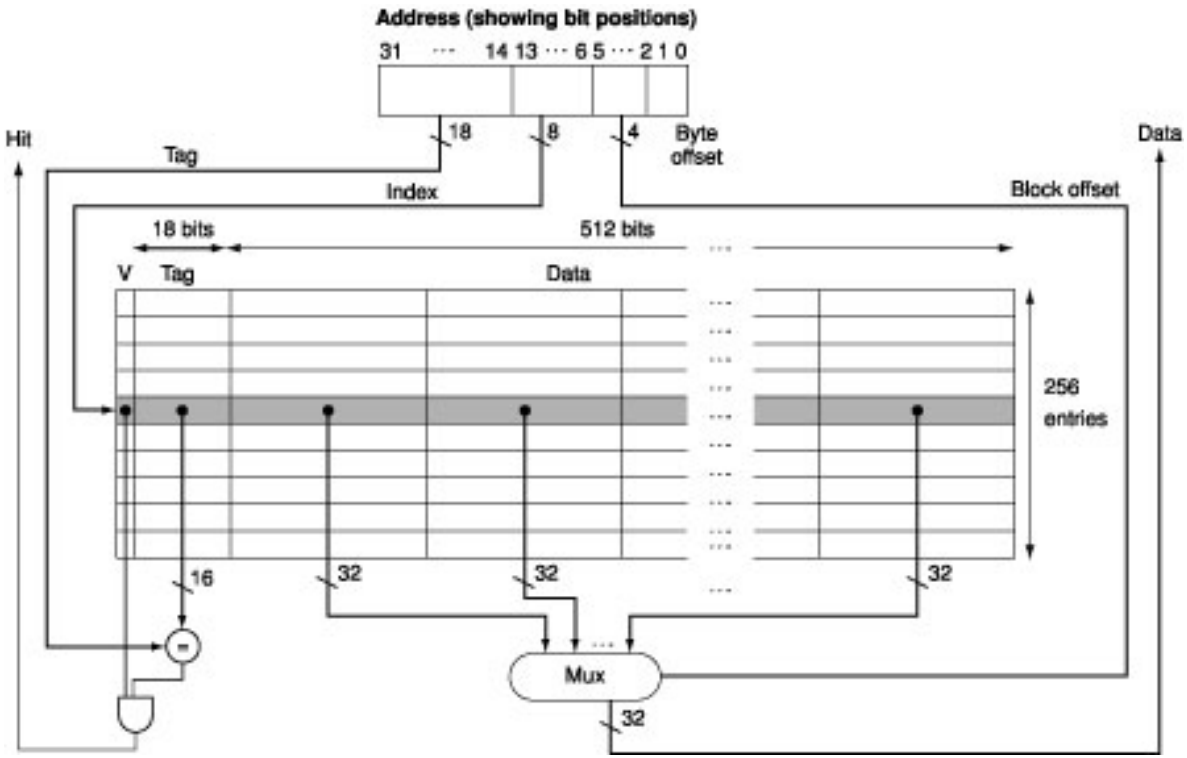
Cache Mapping

Cada línea tiene un *tag* que corresponde a los bits más significativos de la dirección. También se tiene un bit de validez que se marca cuando hay datos o instrucciones validos en el cache. Para comprobar si esta un elemento en el cache (generando un cache hit), se compara el contenido del tag con los bits superiores de la dirección que se busca y se comprueba que el bit de validez este en "1".



Cache Mapping

Si los bloques son de más de una palabra, entonces los bits menos significativos se utilizan para indicar la posición de la palabra dentro del bloque.

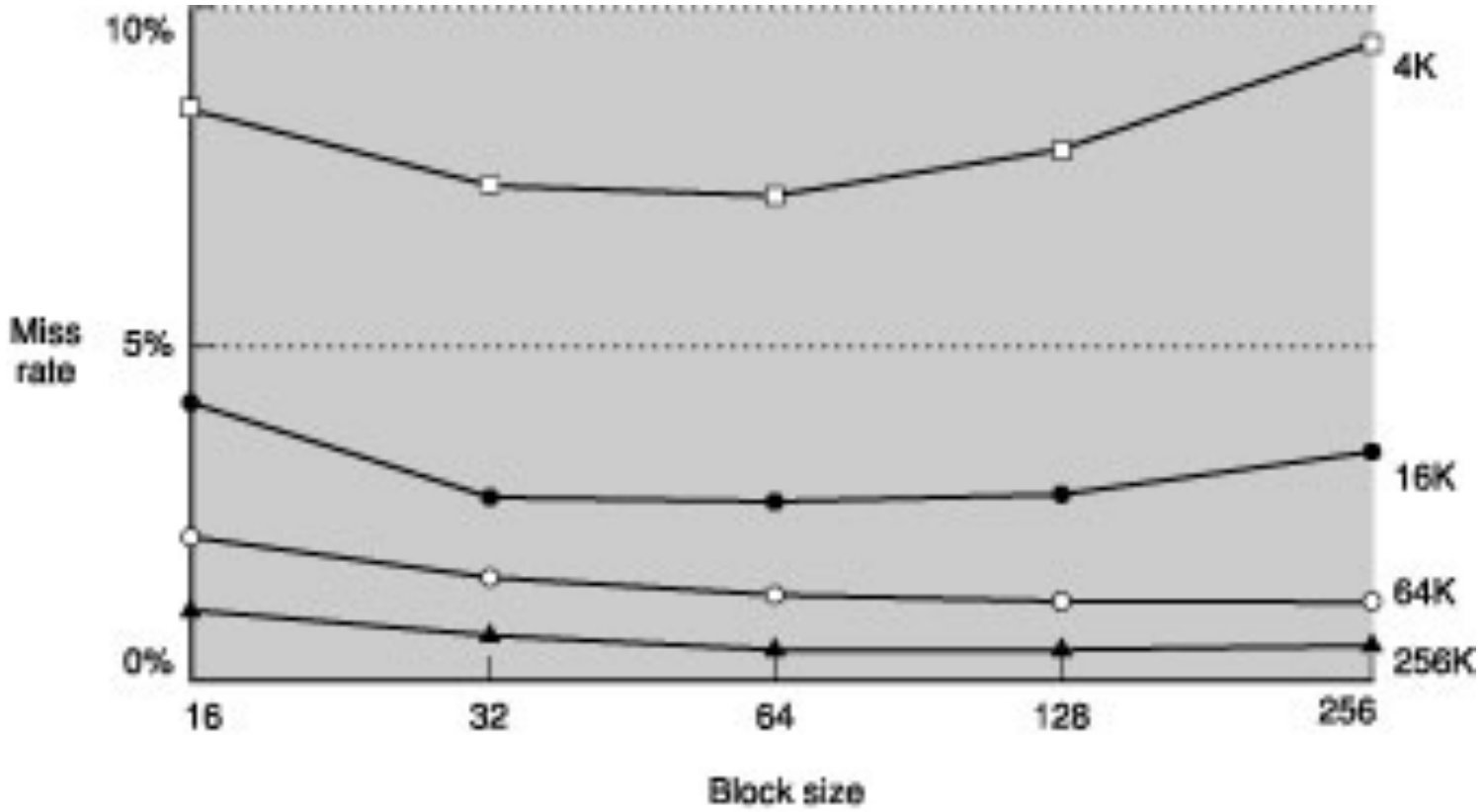


Cache Mapping

-Al aumentar el tamaño de bloque estamos explotando la localidad espacial.

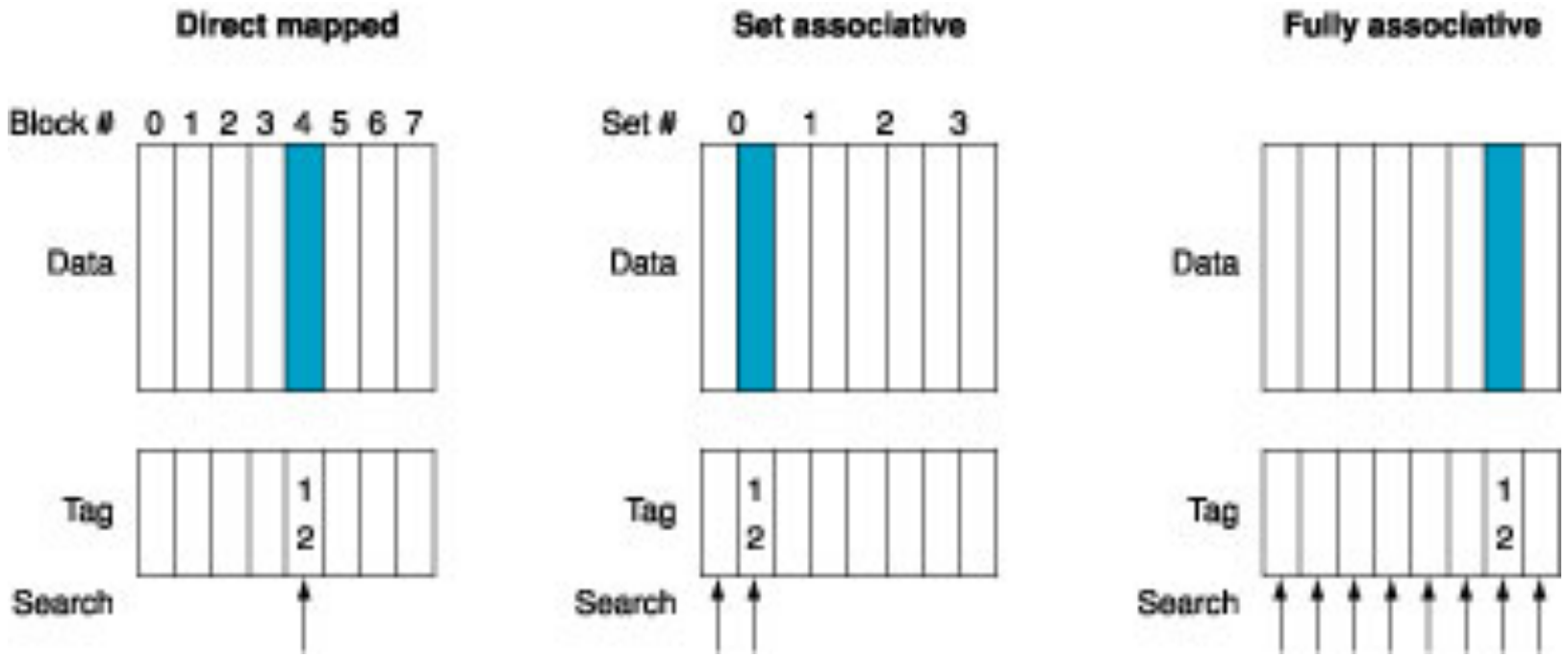
-Sin embargo, también aumentamos el número de reemplazos ya que el cache se llena rápido y terminamos no usando elementos del bloque.

-Adicionalmente, la penalización causada por un cache miss aumenta con el tamaño del bloque. Normalmente, existe un tamaño óptimo para el tamaño del cache y el tipo de carga de trabajo.



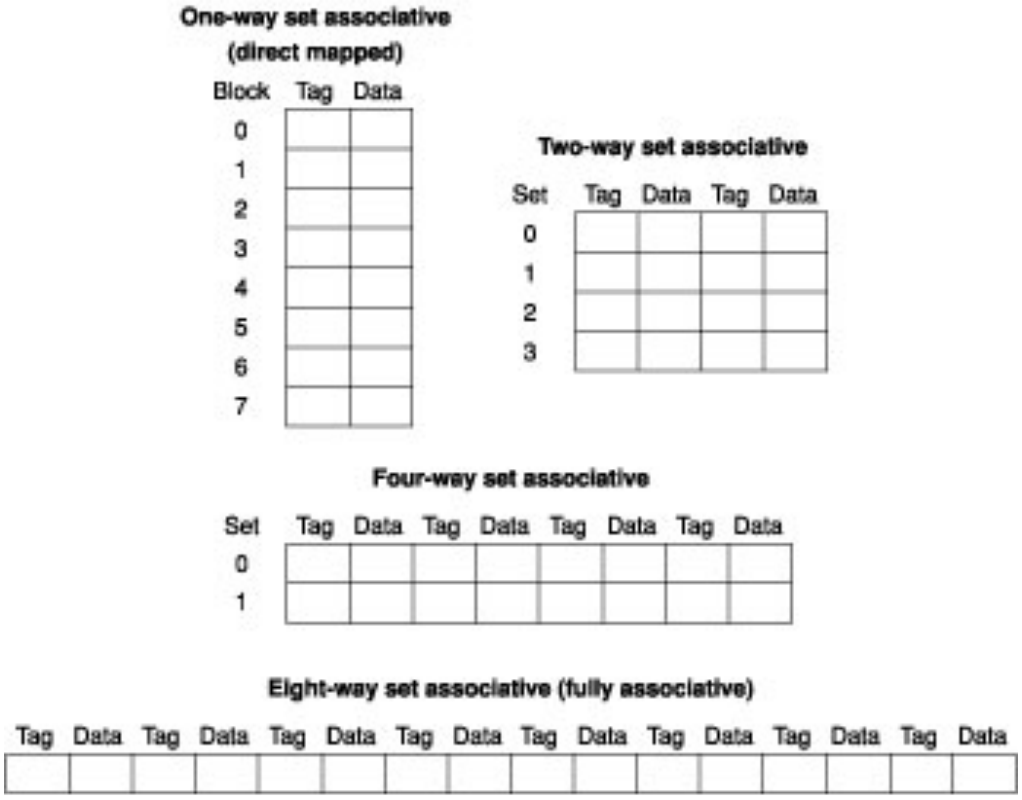
Cache Mapping

El problema con el esquema simple de Direct Mapped es que reemplazamos constantemente elementos que nos pueden servir. Una solución es utilizar esquemas de mapeo más flexibles. En estos esquemas, cada posición de memoria puede ocupar más de una posición de cache. Puede ocupar uno de un set de posiciones. Este esquema de mapeo se conoce como *Set Associative*.



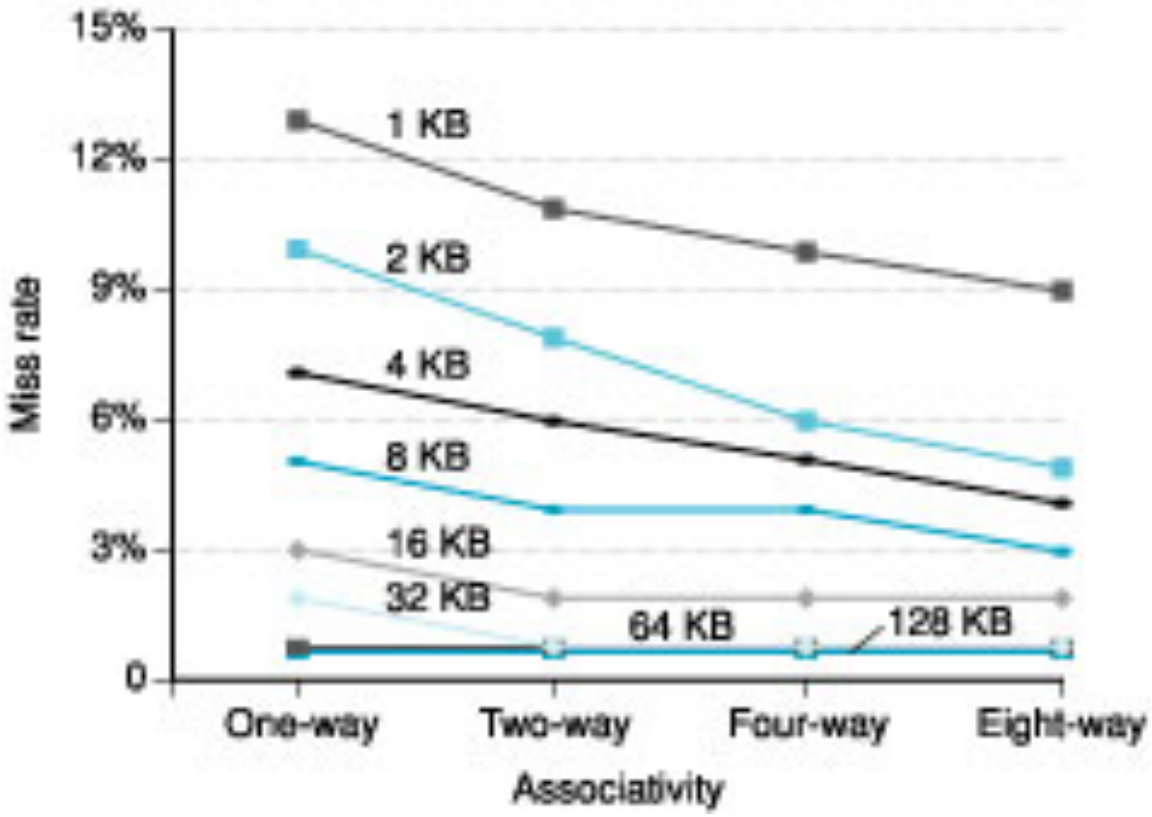
Cache Mapping

El set puede llegar a ser del tamaño de cache. En este caso extremo de flexibilidad, cualquier posición de memoria puede estar en cualquier línea del cache. Este esquema de mapeo se conoce como *Fully Associative*.



Cache Mapping

Al aumentar el tamaño del cache y la asociatividad del mapeo disminuye la cantidad de cache misses. Sin embargo la relación no es lineal y llega un momento donde ya no es considerable. Adicionalmente, al aumentar el grado de asociatividad, aumenta la complejidad de la determinación de si esta o no un elemento en el cache. También aumenta el tamaño del tag (y por lo tanto del cache), disminuyendo la eficiencia del almacenamiento.



Cache Mapping

Cuando se llena el cache en esquemas que utilizan asociatividad, ¿cómo escogemos que línea reemplazar?

Típicamente utilizamos el esquema de Least Recently Used (LRU). En otras palabras, reemplazamos la línea que fue utilizada hace más tiempo.

Hay otros esquemas pero típicamente el overhead es mayor que la mejora en desempeño.

Cache Writing

Recordemos que lo que esta en el cache es una copia de lo que se encuentra en memoria en niveles mayores.

Si escribimos un dato al cache debemos de modificar también eventualmente el contenido en estas memorias de mayor nivel.

Cache Writing – cont.

¿Por qué?

Para evitar que ambas posiciones tengan valores distintos (esto se conoce como que son inconsistentes).

Hay dos esquemas de escritura: *Write-Through* y *Write-Back*.

Cache Writing – cont.

Write-Through: Cada vez que se escribe se escribe también a memorias de niveles superiores. Para evitar que el procesador espere al escribir a memorias lentas, comúnmente se implementa un *write buffer*.

Cache Writing – cont.

Write-Back: Solo se escribe al cache. Solo cuando la línea a la que se escribió va a ser reemplazada se escribe el dato a la memoria de mayor nivel.

Paging

Paging es un mecanismo similar a caching solo que es entre las memorias de mayor nivel. Es un esquema manejado por software (Sistema Operativo).

Esta íntimamente ligado con el concepto de memoria virtual.

Paging – cont.

Al implementar memoria virtual, el sistema tiene disponible mucha más memoria para direccionar que la memoria principal real (memoria física).

Es necesario realizar una "traducción" de la dirección virtual para obtener la dirección física.

Paging – Page Fault

Cuando un elemento que se desea accesar no esta en la memoria principal, ocurre un *page fault* (similar al cache miss). El elemento es buscado en el almacenamiento masivo (ej. disco duro) que es el último nivel de la jerarquía y el bloque (*page*) que contiene al elemento es copiado a la memoria principal.

Paging – Page Table

Para disminuir el gran overhead causado por un page fault, el esquema de mapeo de las páginas que se emplea es típicamente completamente flexible, como el fully associative en un cache.

Para determinar si esta ó no un elemento, se emplea una tabla (*page table*) implementada en memoria principal que traduce la dirección virtual a una dirección física.

Paging - Reemplazo

Para el reemplazo de páginas en memoria, se emplean algoritmos similares a los caches asociativos como el Least Recently Used (LRU).

Paging - TLB

Dado que la tabla esta en memoria principal es tardado accesarla y, dado que en cada acceso a memoria sería necesario accesarla (dos accesos a memoria por cada lectura o escritura), es deseable encontrar una forma de evitar esto.

Paging - TLB

Algunos procesadores modernos implementan un buffer (como un cache) para las traducciones del page table. Este cache se conoce como *Translation Lookaside Buffer* (TLB). En él, se guarda la traducción de las páginas más utilizadas. Siempre se intenta realizar la traducción en el TLB antes que buscar en el page table.

Paging – TLB cont.

En el TLB, el *virtual page number* (parte alta de la dirección virtual) se utiliza como índice y/o tag, y el dato es el *physical page number* (parte alta de la dirección física). Típicamente, el mapeo de los TLB es asociativo (set o fully).