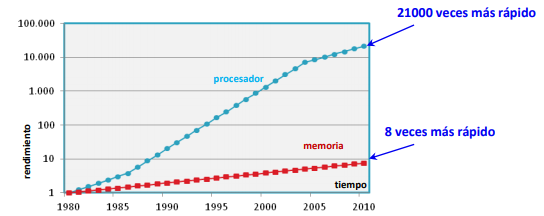
**TEMA 3: Conceptos Básicos Memoria Cache**

**Introducción**

Visión general de la jerarquía

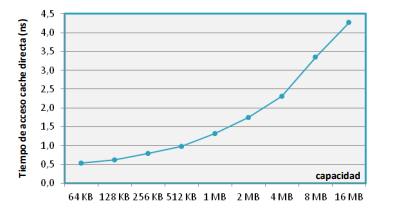
La jerarquía de memorias está justificada por las siguientes situaciones:

* Velocidad procesadores > Velocidad memorias



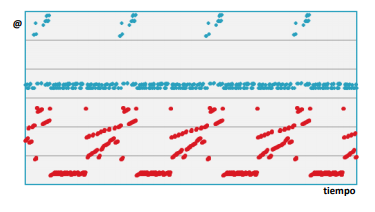
La velocidad de los procesadores ha evolucionado mucho durante el tiempo, hasta alcanzar hasta 21000 veces más rápido, en cambio, la velocidad de los accesos a memoria es notablemente más lentos y no han evolucionado tanto, “solo” son 8 veces más rápido.

* Propiedades de las memorias



Como más grande sea la capacidad de la memoria, más se tarda en acceder a ella.

* Propiedades de los programas



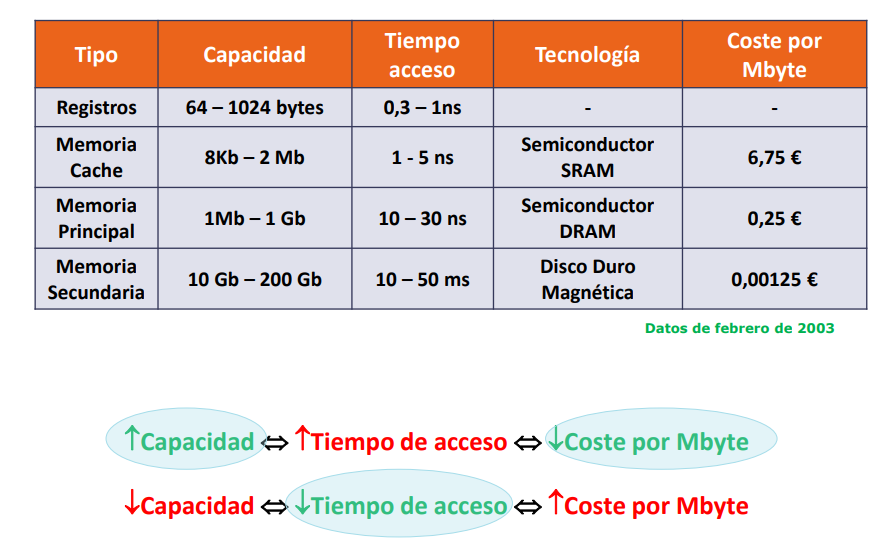
Los accesos a memoria son predecibles en cierto modo, por lo que acelera los accesos.

Tipos de memoria de semiconductores

Memoria estática: SRAM, Static RAM, cada celda de memoria equivale a 1 biestable (6-8 transistores). En comparación con las DRAM son más rápidas, tienen un alto consumo, son pequeñas (poca capacidad) y caras. => MEMORIA CACHE.

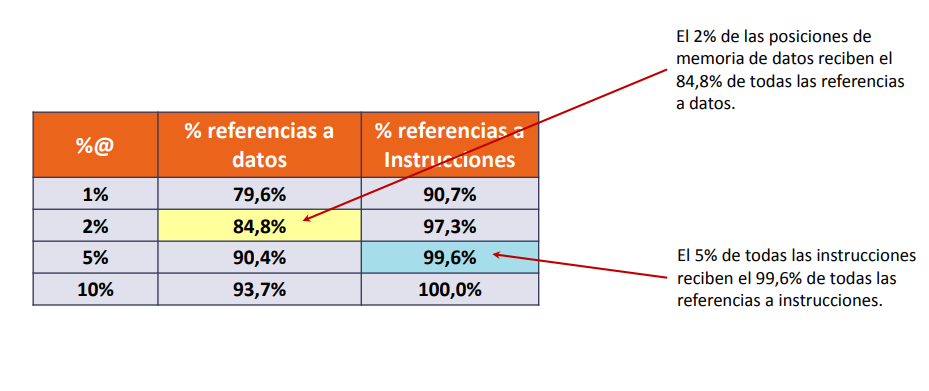
Memoria dinámica: DRAM, Dynamic RAM, cada celda se comporta como un condensador (1-1.x transistores). En comparación con las SRAM son más lentas, tienen un bajo consumo, son grandes (mucha capacidad) y baratas. Pero además tiene el problema del refresco. => MEMORIA PRINCIPAL.

Caracterización de las memorias



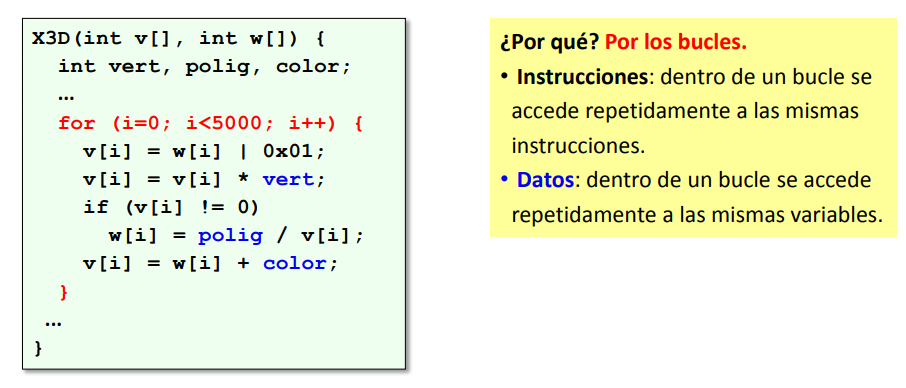
Propiedades de los programas: Regla del 90/10

El 90% de todas las referencias a memoria (datos e instrucciones) son realizadas por el 10% del código. Ejemplo: Benchmark Spec2000.



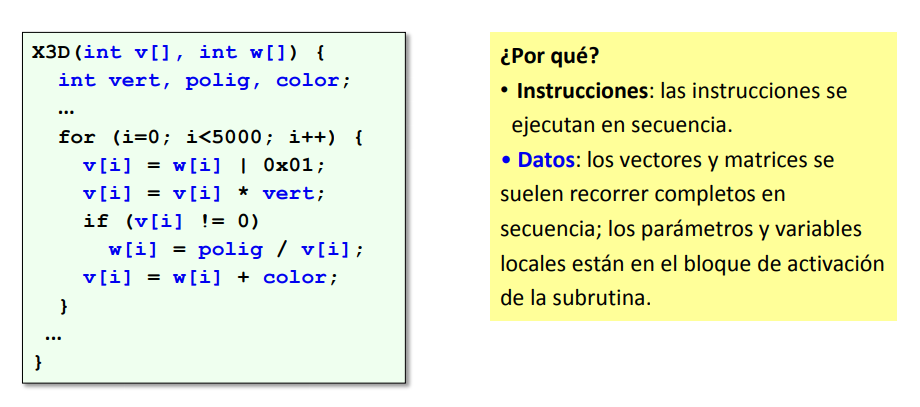
Esto es debido a 2 propiedades:

Localidad Temporal

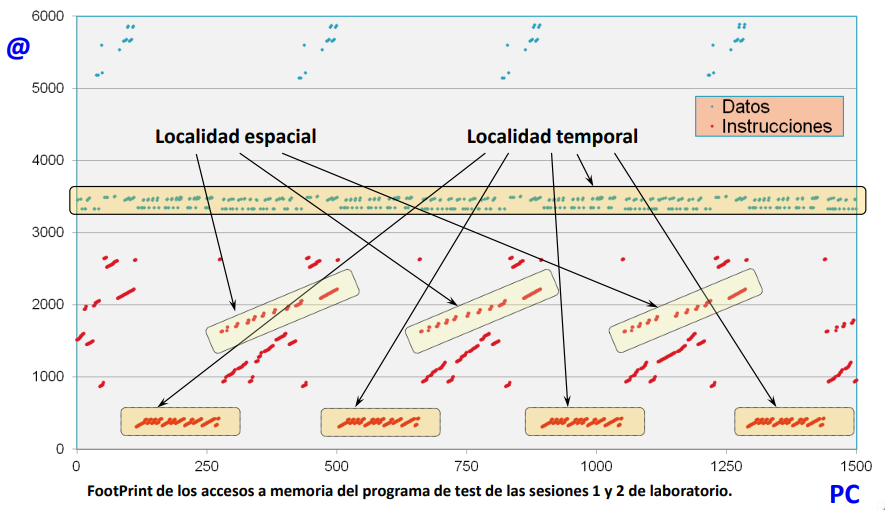
Si accedemos a una posición de memoria, es muy probable que se vuelva a acceder a la misma posición en un futuro cercano.

Localidad Espacial

Si accedemos a una posición de memoria, es muy probable que se acceda a posiciones próximas en un futuro cercano.



Visualización de la localidad espacial y temporal

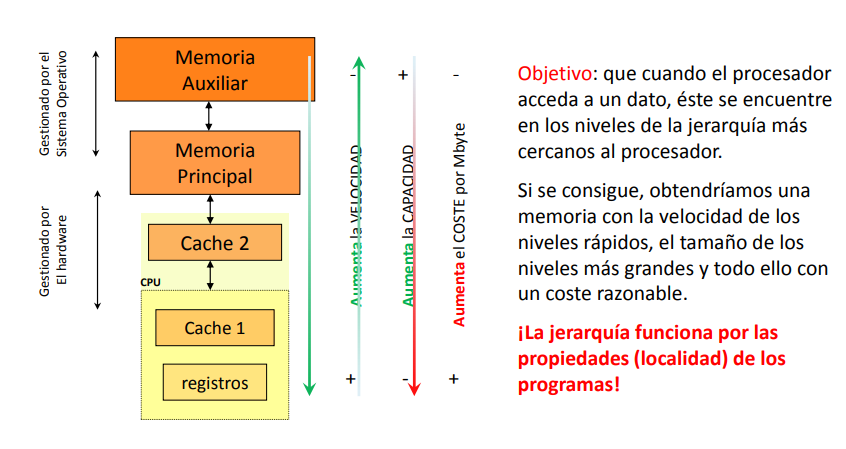


Podemos ver que los accesos no son totalmente aleatorios, sino que siguen un patrón.

¿Cómo podemos aprovechar la localidad?

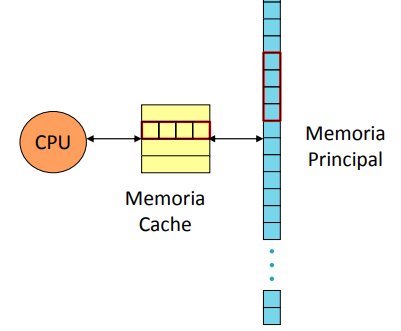
* Localidad temporal: si traemos un dato (o instrucción) de memoria, sería útil guardarlo “cerca” del procesador para que los futuros accesos sean más rápidos.
* Localidad espacial: si traemos un dato (o instrucción) de memoria, sería útil traer también los datos próximos y dejarlos “cerca” del procesador. Esto sólo tiene sentido si traer datos próximos sólo cueste un poco más que traer un solo dato.

Solución: Jerarquía de memorias



Principios de funcionamiento de las memorias cache

Memoria cache: memoria pequeña y rápida que almacena una parte del contenido de una memoria más grande y lenta. La memoria cache se encargará de que la información que se almacene sea útil. Esta memoria es transparente al programador.



Objetivos:

* Velocidad de la memoria cache.
* Capacidad de la memoria principal.
* Coste de la memoria principal más un porcentaje razonable.

Esta solución es posible debido a la localidad de los programas. La cache retiene información recientemente usada e información próxima a la recientemente usada.

Conceptos:

* Acierto / fallo.
* Línea de cache / bloque de memoria.
* Algoritmos de emplazamiento.
* Algoritmos de reemplazo.
* Políticas de escritura.
* Evaluación.

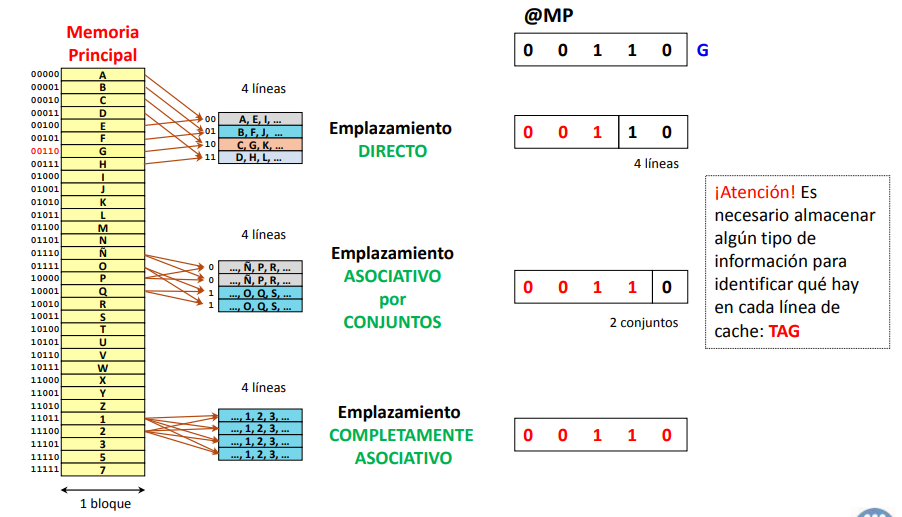
En cualquier memoria cache hay que definir:

* Algoritmo de emplazamiento: determina en qué líneas de MC puede colocarse un bloque. Determina, también, donde hay que buscar un dato.
* Algoritmo de reemplazo: determina en qué línea se ha de eliminar de la cache para dejar espacio a un nuevo bloque.
* Políticas de escritura: determina cómo se hacen las escrituras. En cualquier caso, al final siempre se ha de escribir en MP.

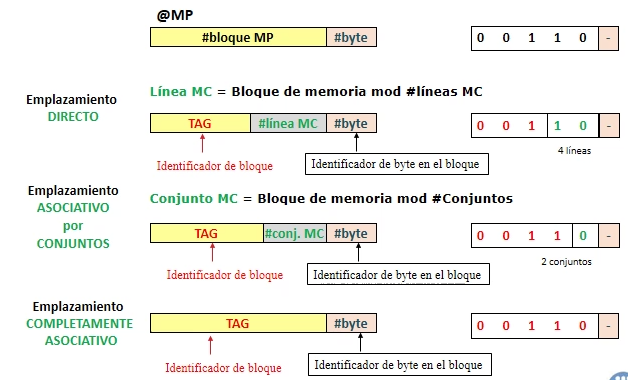
Han de ser algoritmos hardware:

* Algoritmos sencillos.
* Algoritmos rápidos.

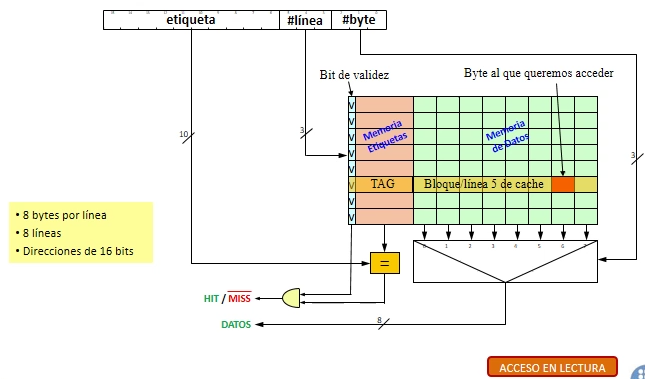
**Algoritmos de emplazamiento**



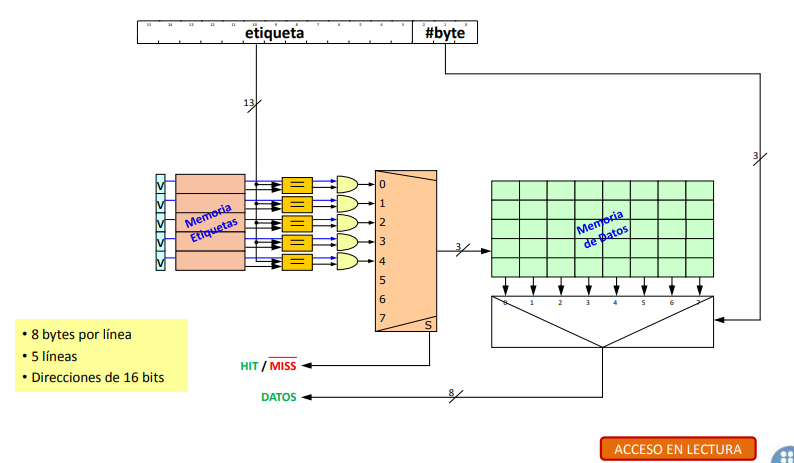
3 tipos de emplazamiento:



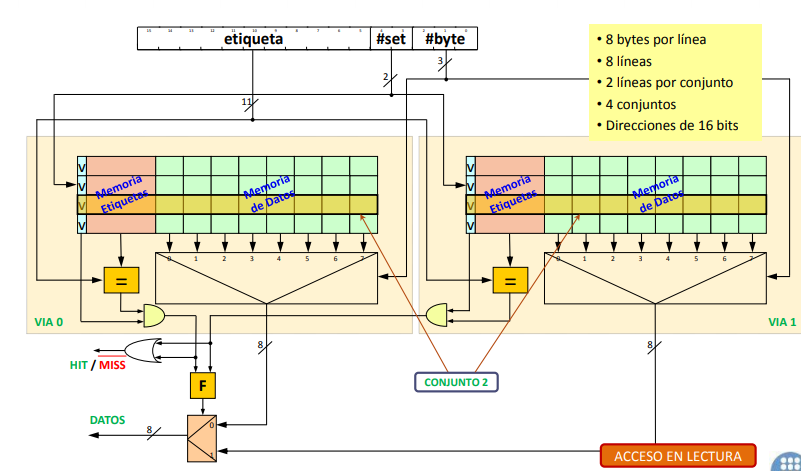
Cache directa:

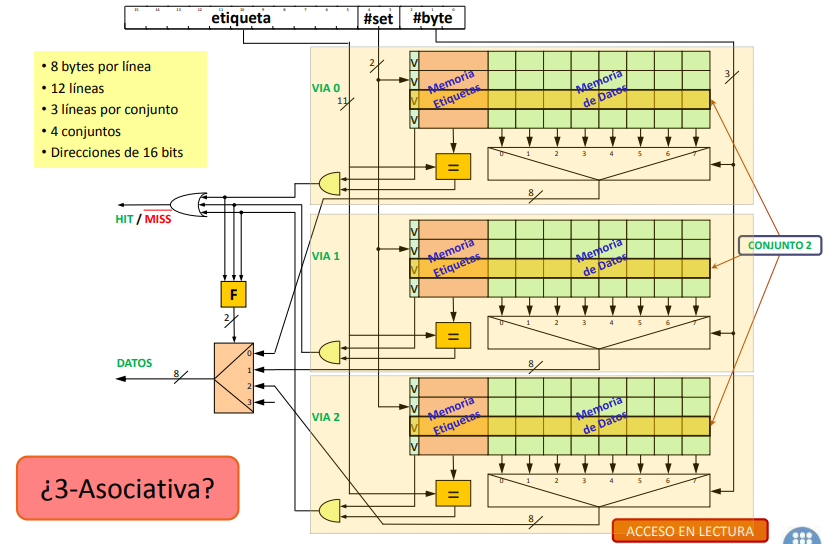


Cache completamente asociativa

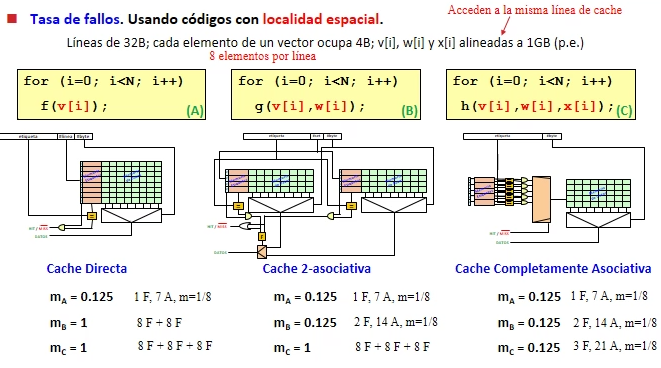


Cache asociativa por conjuntos





Evaluación algoritmos emplazamiento



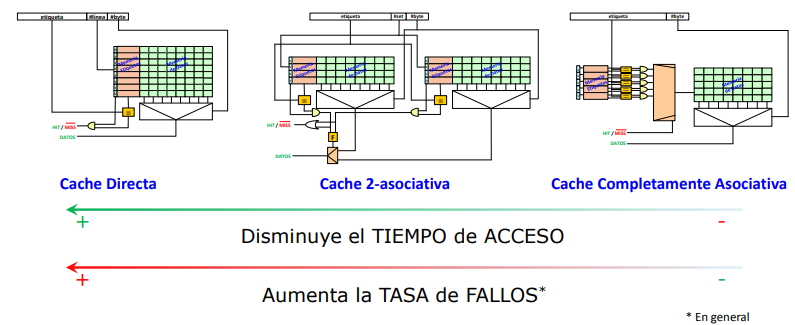
Tiempo de acceso: depende del camino crítico.

Cache directa: Acceso (Memoria datos y etiquetas), comparar etiqueta y validar línea.

Cache asociativa por conjuntos: Acceso (Memoria datos y etiquetas), comparar etiqueta, validar línea y seleccionar vía.

Cache completamente asociativa: Acceso (Memoria datos y etiqueta), comparar etiqueta, validar línea y Acceso (Memoria datos).

Comparación de los 3 algoritmos de emplazamiento



MC directa: MC asociativa por conjuntos con 1 línea por conjunto.

MC completamente asociativa: MC asociativa por conjuntos con 1 único conjunto.

Para poder comparar correctamente las diversas configuraciones de cache hay que utilizar otros parámetros, como por ejemplo, el tiempo medio de acceso (Tma).

**Algoritmos de reemplazo**

Si se produce un fallo y el conjunto donde debe ubicarse la nueva línea está lleno, ¿qué línea del conjunto hay que reemplazar?

¿Algoritmo importante? Un comportamiento deficiente puede provocar que reemplacemos una línea que se va a utilizar en un acceso próximo.

Algoritmos hardware muy simples. Se han de ejecutar en un plazo de tiempo muy pequeño.

Los algoritmos de reemplazo se aplican en caso de fallo, pero algunos de ellos necesitan actualizar cierta información después de cada acierto.

En una MC directa no tiene sentido hablar de algoritmo de reemplazo.

Principales algoritmos de reemplazo

Reemplazo aleatorio:

* Se selecciona aleatoriamente una línea de entre todas las candidatas a ser reemplazadas.
* Es un algoritmo muy sencillo de implementar.
* A pesar de su aparente falta de sentido, funciona bastante bien.

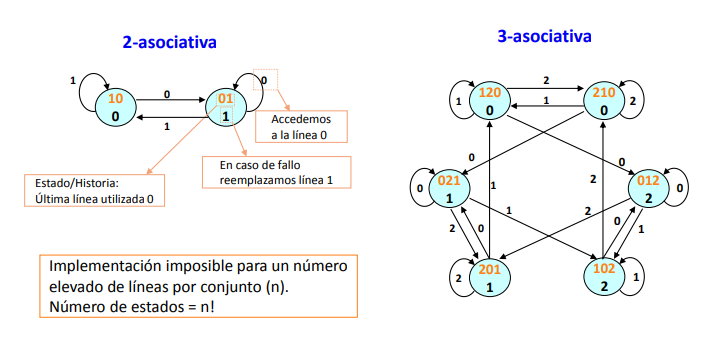
Reemplazo FIFO (First In First Out):

* De entre todas las líneas candidatas a ser reemplazadas, se selecciona la que lleva más tiempo en la cache.
* Es un algoritmo muy sencillo de implementar, solo es necesario utilizar un contador con módulo.
* Tiene un comportamiento patológico no deseable porque no tiene en cuenta la utilización.

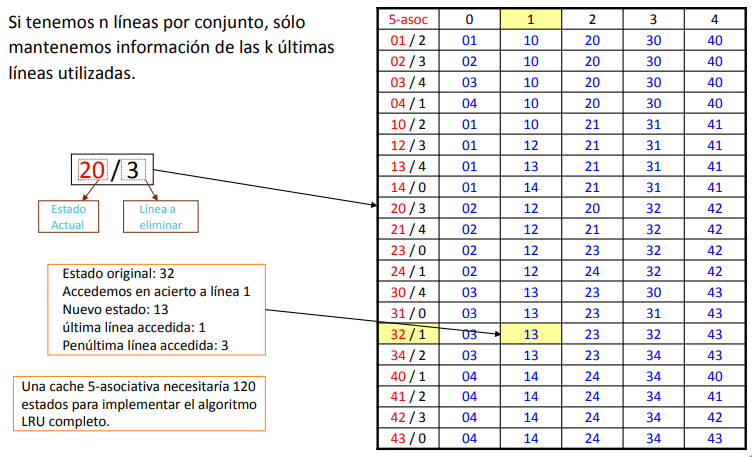
Reemplazo LRU (Least Recently Used):

* De entre todas las candidatas a ser reemplazadas, se selecciona la que lleva más tiempo en la cache sin ser utilizadas.
* Este algoritmo da buenos resultados. Teniendo en cuenta el comportamiento de los programas, parece la opción más lógica.
* Sin embargo, es muy costoso de implementar si el grado de asociatividad es alto. El coste de implementar este algoritmo es n! (siendo n el grado de asociatividad).
* Normalmente se implementa un algoritmo PseudoLRU. Un algoritmo LRU ha de mantener información de en qué orden se ha accedido a todas las líneas de un conjunto. En un algoritmo pseudoLRU solo se mantiene parte de esa información.

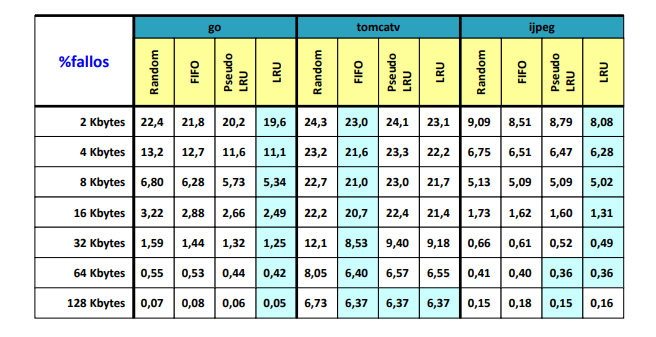
Posible implementación de un reemplazo LRU

Un registro de estado para cada conjunto. En función del estado, sabemos qué línea hay que reemplazar.

Posible implementación de un algoritmo pseudoLRU (5 vías)



Algoritmos de reemplazo



A la vista de estos resultados, se puede concluir que los algoritmos de reemplazo no influyen sustancialmente en el rendimiento de la MC.

**Políticas de escritura**

Premisa: las escrituras, finalmente, se han de hacer en memoria principal.

¿Cuándo se actualiza la memoria principal?

WRITE THROUGH (escritura a través o escritura inmediata):

* Se actualiza simultáneamente el dato en la MC y la MP.
* El tiempo de servicio es el tiempo de acceso a MP.
* La MP siempre está actualizada.
* Se puede reducir el tiempo de escritura utilizando buffers.

COPY BACK (escritura diferida):

* En una escritura sólo se actualiza el dato en MC.
* Para cada línea se añade un bit de control (dirty bit) que indica si la línea ha sido modificada o no.
* Se actualiza el bloque en la MP cuando la línea (si ha sido modificada) ha de ser reemplazada.
* Las escrituras son rápidas (velocidad de MC).
* El tiempo de penalización en caso de fallo aumenta.
* Durante un tiempo existe una inconsistencia entre MP y MC.

¿Qué hacer en caso de fallo en escritura?

WRITE ALLOCATE (con migración en caso de fallo):

* Se trae el bloque de MP a MC y después se realiza la escritura.

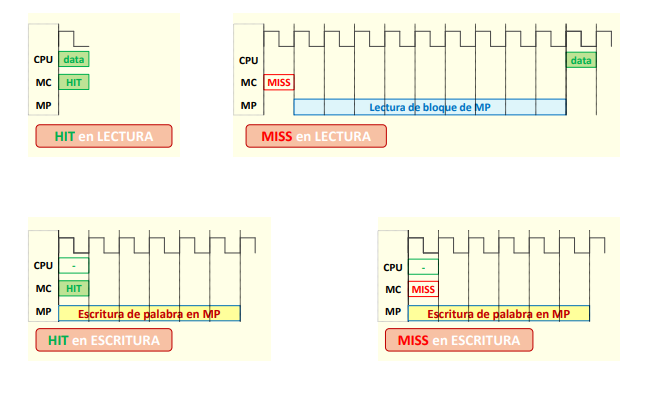
WRITE NO ALLOCATE (sin migración en caso de fallo):

* El bloque no se trae a MC. Esto obliga a realizar la escritura directamente en MP.

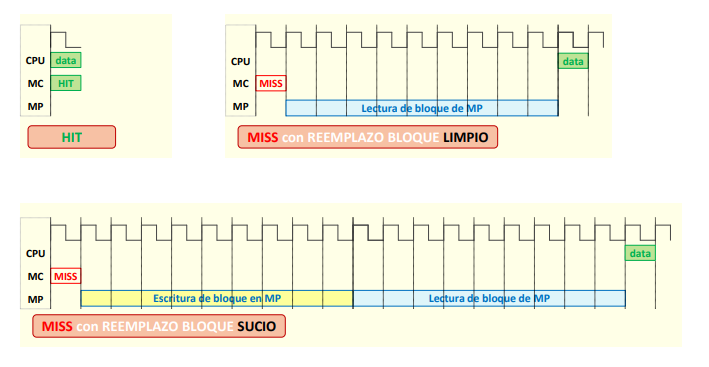
Operaciones a realizar en un acceso a cache

Modelo utilizado:

* Cache de datos.
* Políticas de escritura (2 casos):
  + WRITE THROUGH + WRITE NO ALLOCATE.
  + COPY BACK + WRITE ALLOCATE.
* Tamaño de línea: 32B.
* Tiempo de servicio en caso de acierto: 1B.
* Memoria principal:
  + Lectura línea: 9 ciclos.
  + Escritura línea: 9 ciclos.
  + Escritura palabra: 6 ciclos.

WRITE THROUGH + WRITE NO ALLOCATE

COPY BACK + WRITE ALLOCATE



**Influencia de los parámetros de la cache en el rendimiento**

Medida de rendimiento (tasa aciertos y tasa de fallos):

* Tasa de aciertos: h = (#aciertos / #referencia).
* Tasa de fallos: m = (#fallos / #referencia) = 1 – h.
* ¿De qué dependen?
  + Del tamaño de cache.
  + Del tamaño de bloque.
  + De los algoritmos de emplazamiento y reemplazo.
  + Del programa evaluado.

El programa influye de forma sustancial en la tasa de fallos (depende de unos programas u otros hay más fallos o no).

Aumentar el tamaño de cache provoca que la tasa de fallos disminuya (ya sea cache de datos o cache de instrucciones).

El tamaño de bloque influye, pero, en ambos sentidos. (pueden hacer aumentar o disminuir la tasa de fallos)

Aumentar la asociatividad implica disminuir la tasa de fallos.

Como más grandes y más mayor asociatividad sean las memorias caches, mejor ya que disminuye la tasa de fallos, el tamaño del bloque no influye.

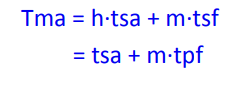
Medida de rendimiento (tiempo servicio en caso acierto):

* Tiempo de servicio en caso de acierto (coste de un acceso a cache en acierto).
* ¿De qué dependen?
  + Del tamaño de cache.
  + De los algoritmos de emplazamiento y reemplazo.
* ¿En qué influyen?
  + El acceso a cache está en el camino critico de un procesador.
  + Puede comprometer el tiempo de ciclo del procesador.

El tsa (tiempo servicio en caso de acierto) aumenta con la asociatividad y el tamaño de cache.

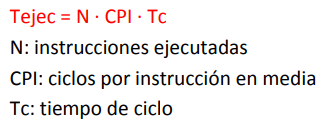
Medida de rendimiento (tiempo medio de acceso): (solo en MCs de solo lectura)

* Tiempo medio de acceso (coste de un acceso a memoria): Tma.
* Componentes:
  + Coste de un acceso en acierto: tsa.
  + Coste de un acceso en fallo: tsf: tsa + tpf.
  + Penalización de un fallo: tpf.
* Tiempo medio de acceso a memoria:



Medida de rendimiento (tiempo de ejecución):

* Tiempo de ejecución de un programa:



* CPI = CPIideal + CPImem.
* CPImem son los ciclos que perdemos por tener una cache imperfecta (m != 0).

CPImem = nr \* (Tma – tsa).

CPImem = nr \* m \* tpf (caso particular de una MC de solo lectura).

(nr = número medio de referencias por instrucción).

Estructura básica de un computador actual

