Conceptos Básicos de Memoria Caché

Disco	Memoria Principal	Memoria Caché	Procesador
+ lenta	…	…	+ rápida
+ grande	…	…	+ pequeña
+ barata	…	…	+ cara

> Localidad espacial: si accedes a una posición de memoria, es muy probable que accedamos a posiciones cercanas (por ejemplo en un bucle).

Por lo tanto, si yo accedo a la posición `i` de memoria, me traigo a caché un rango de direcciones consecutivas `i - x` .. `i + x` (un bloque entero de datos).

> Localidad temporal: si accedes a una posición de memoria, es muy probable que vuelvas a acceder a la misma posición de memoria.

Algoritmos de emplazamiento

El algoritmo de emplazamiento determina, dado un bloque de la memoria principal, ¿a qué bloque de la memoria caché va a parar?

Correspondencia directa

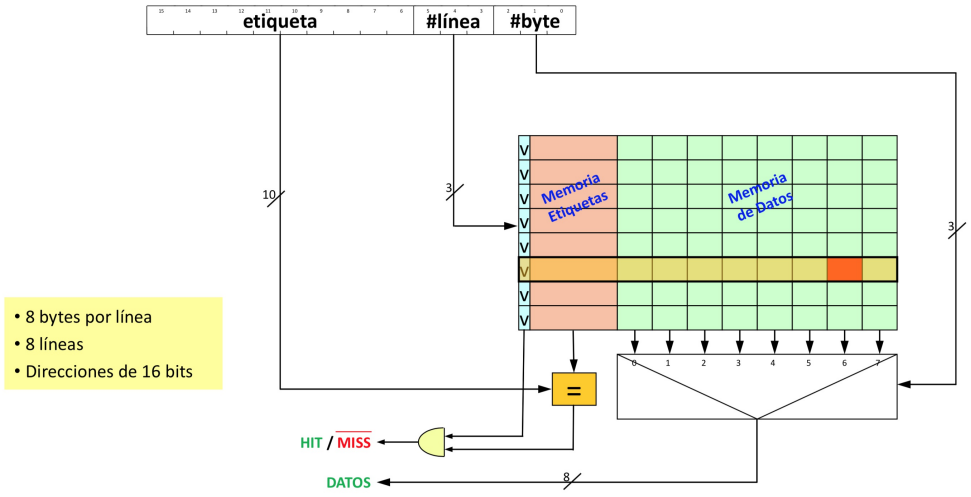
> Cada bloque va a una posición determinada.


```
MC = MP % sizeof(MC)    // más o menos
```

Esto lo determinan los bits de menor peso (los bits necesarios).

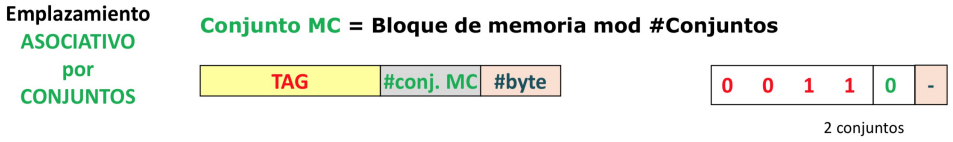
Para saber a qué bloque de memoria principal pertenece el bloque de la memoria caché se guardan los bits de mayor peso (que combinados con la posición del bloque en memoria

caché me da la dirección en memoria principal).

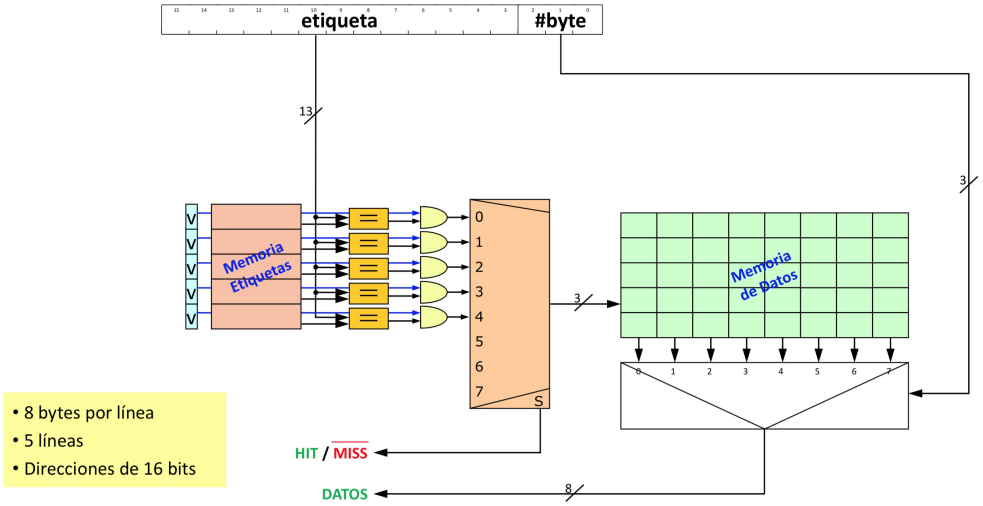


Asociativo por conjuntos

> Partimos la caché en conjuntos y cada línea de memoria principal va a un conjunto. Pero dentro de este conjunto puede ir a cualquier bloque de memoria caché.

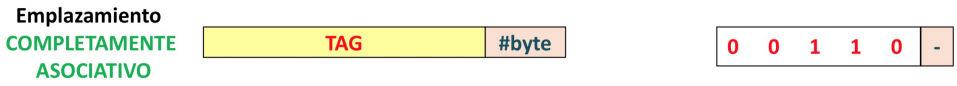


Cogemos los bits de menor peso (los bits necesarios) que nos dirán el número del conjunto. Pero también tengo que guardar la parte de mayor peso para saber a qué posición de memoria principal pertence.

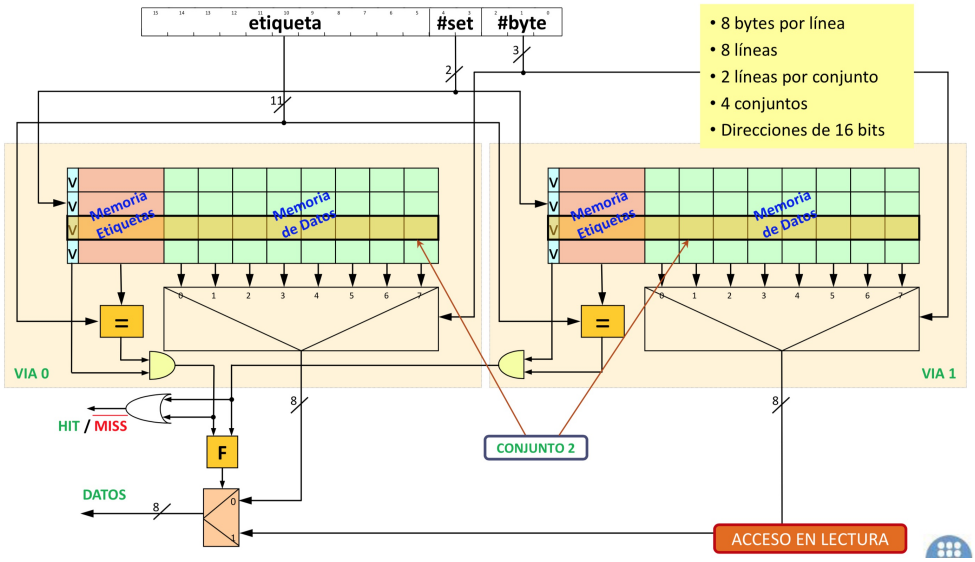


Completamente asociativo

> Cualquier bloque de memoria principal puede ir a cualquier bloque de la memoria caché



Necesito guardar toda la dirección de memoria principal en caché para poder saber a qué bloque pertenece.



Algoritmos de reemplazo

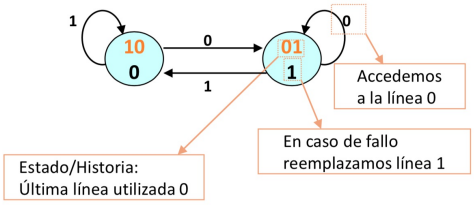
Qué bloque de memoria caché hay que quitar para poner el bloque de memoria principal en caché.

LRU

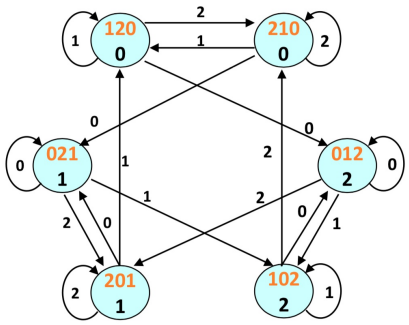
> Siempre se quita el último usado

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

2-asociativa



3-asociativa



Implementación imposible para un número elevado de líneas por conjunto (n).
Número de estados = n!

Pseudo LRU

Si tenemos n líneas por conjunto, sólo mantenemos información de las k últimas líneas utilizadas.

Estado original: 32
Accedemos en acierto a línea 1
Nuevo estado: 13
última línea accedida: 1
Penúltima línea accedida: 3

Una cache 5-asociativa necesitaría 120 estados para implementar el algoritmo LRU completo.

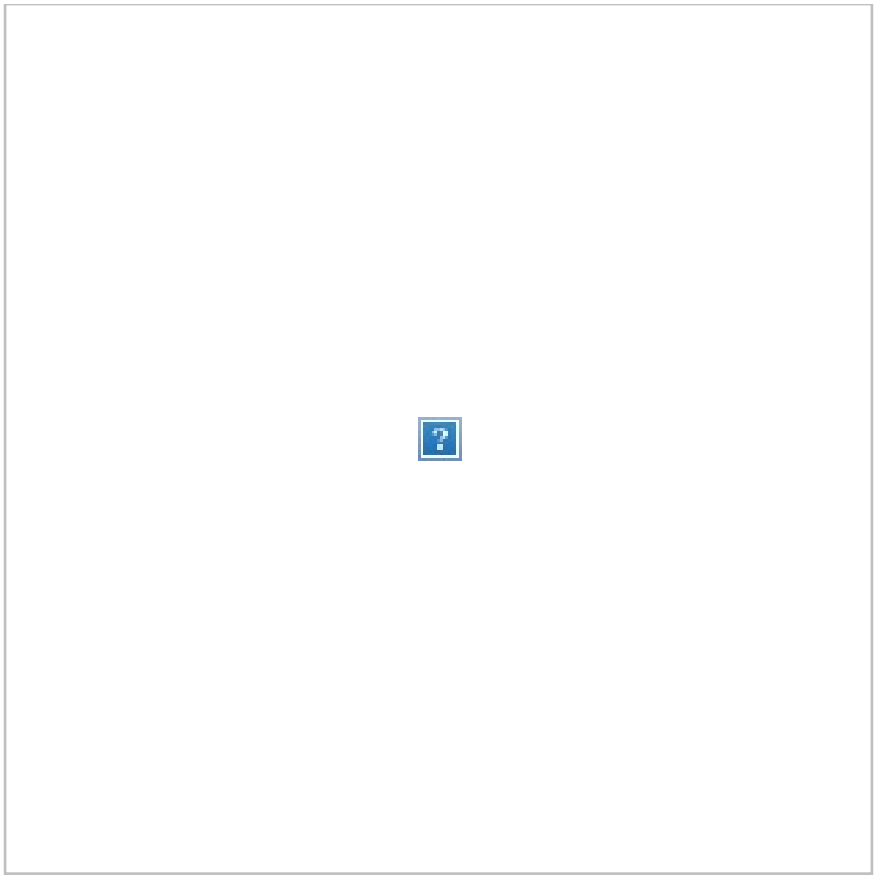
5-asoc	0	1	2	3	4
01 / 2	01	10	20	30	40
02 / 3	02	10	20	30	40
03 / 4	03	10	20	30	40
04 / 1	04	10	20	30	40
10 / 2	01	10	21	31	41
12 / 3	01	12	21	31	41
13 / 4	01	13	21	31	41
14 / 0	01	14	21	31	41
20 / 3	02	12	20	32	42
21 / 4	02	12	21	32	42
23 / 0	02	12	23	32	42
24 / 1	02	12	24	32	42
30 / 4	03	13	23	30	43
31 / 0	03	13	23	31	43
32 / 1	03	13	23	32	43
34 / 2	03	13	23	34	43
40 / 1	04	14	24	34	40
41 / 2	04	14	24	34	41
42 / 3	04	14	24	34	42
43 / 0	04	14	24	34	43

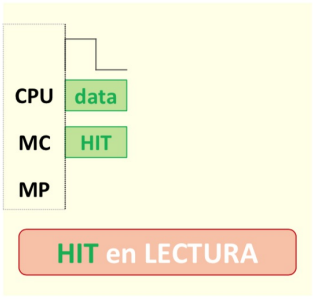
FIFO

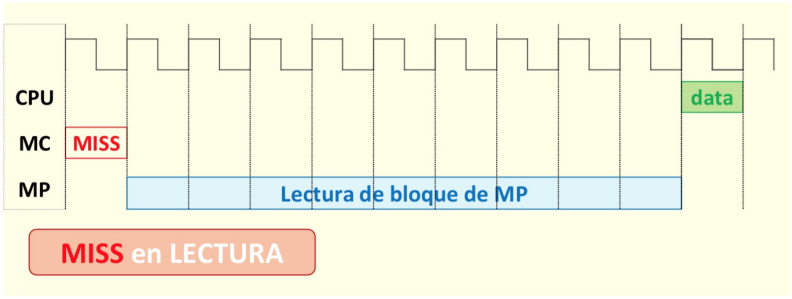
Random

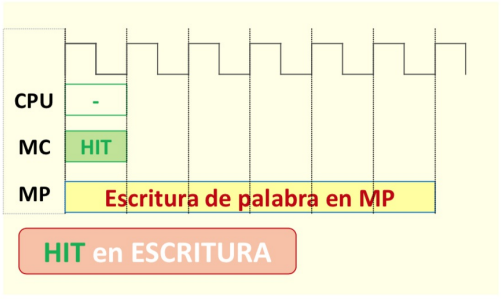
Políticas de escritura

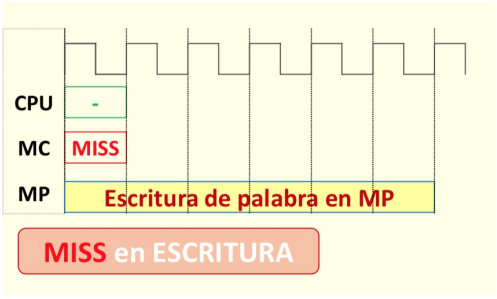
Write-through with no allocation



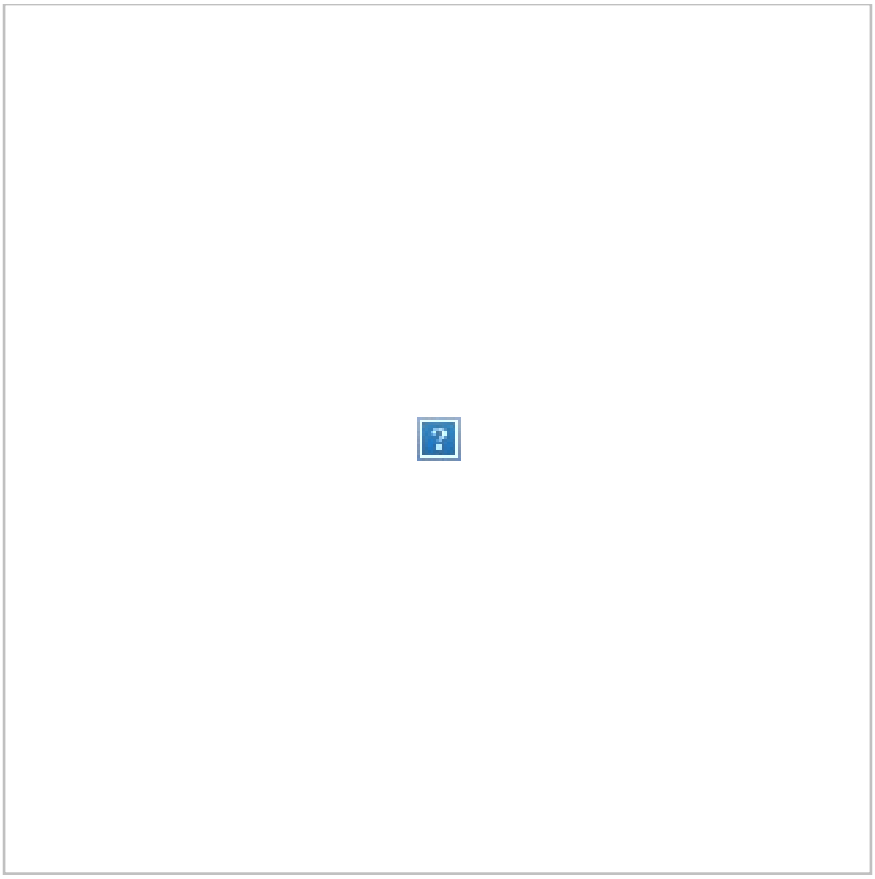


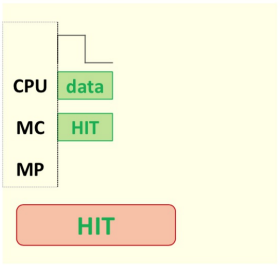


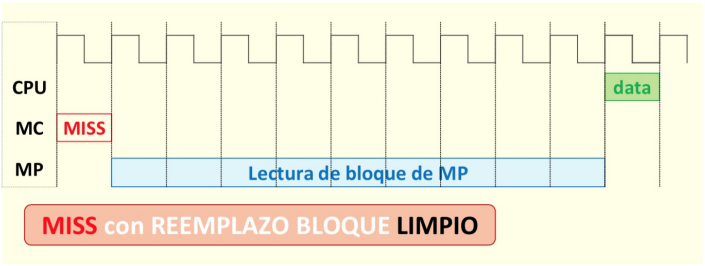


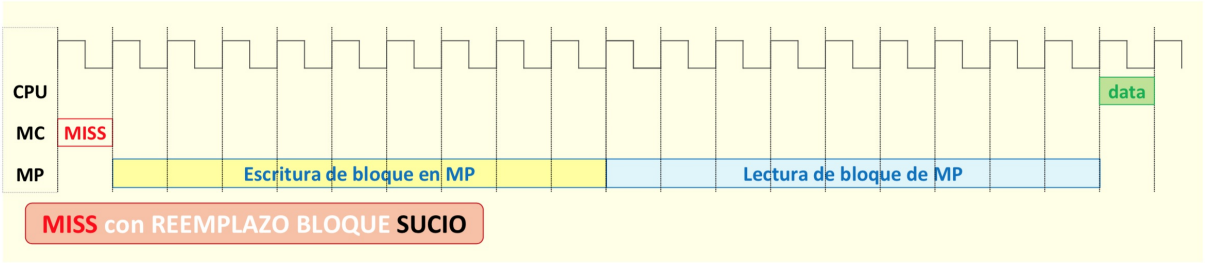


Write-back with allocation










```
movl A(%eax), %eax        # Si la caché es de 64B, muevo 64/4 = 16 long
movl %eax, A(%eax)        # Depende de la política de escritura
```

Cuando el elemento está en caché

Write-back

> Solamente se escribe en caché cuando el bloque abandona la caché.

Write-through

> Se escribe en memoria principal cada vez que se escribe en caché.

Cuando el elemento no está en caché

Write allocate

> Si me bajo la línea a caché.

Write no allocate

> Si no me bajo la línea a caché.