



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament d'Arquitectura de Computadors

Estructura de Computadores

Tema 6: Memoria Cache

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya



Introducción

- ❑ Uno de los componentes que más condicionan el rendimiento de un computador es el acceso a memoria.
- ❑ ¿Cuántos accesos a memoria hay en este código?

```
initV:  
    li $t0, 0  
    li $t1, 100  
for: bge $t0, $t1, end  
      sll $t2, $t0, 2  
      addu $t2, $t2, $a0  
      sw $zero, $($t2) ←  
      addiu $t0, $t0, 1  
      b for  
end: jr $ra
```

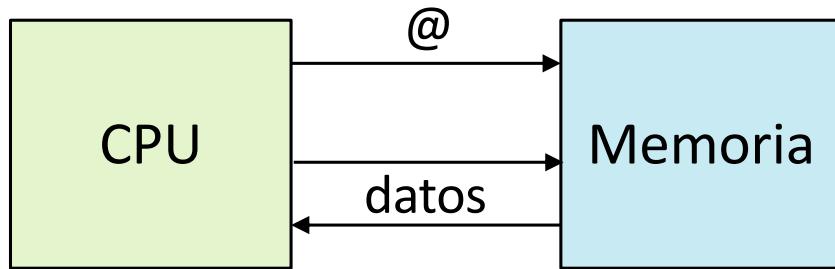
```
void initV(int V[100]) {  
    for (int i=0; i<100; i++)  
        V[i] = 0;  
}
```

100 escrituras en memoria

Las instrucciones también se leen de memoria.
En este código se leen $4 + 6 \cdot 100 = 604$ instrucciones.

Introducción

- ❑ El rendimiento de un computador estará limitado por la velocidad a la cual sea capaz de leer instrucciones.
- ❑ Además un porcentaje muy elevado de las instrucciones son accesos a memoria para leer o escribir datos ($\approx 1/3$).
- ❑ Modelo Simplificado de Computador



Tipos de acceso:

- Leer datos de Memoria (**load**)
 - $lw \$t0, 40(\$sp)$
 $\$t0 \leftarrow M[\$sp+40]$
- Escribir datos en Memoria (**store**)
 - $sw \$t1, 12(\$a0)$
 $M[\$a0+12] \leftarrow \$t1$
- Leer Instrucciones de Memoria (**fetch**)

Introducción

- ❑ **Objetivo: queremos una memoria muy grande y muy rápida.**
- ❑ ¿Cómo medimos la velocidad de la memoria?
- ❑ Dos figuras de mérito
 - **ANCHO de BANDA.** Cantidad de información que movemos por unidad de tiempo.
 - **TIEMPO de ACCESO.** Tiempo consumido desde que se pide un dato hasta que está disponible.
Muchas veces lo llamamos **LATENCIA de MEMORIA.**

Introducción

Modelo de Memoria Principal

- La Memoria Principal (MP) puede verse como:

byte M[MemSize];

- Operaciones básicas:

- Lectura: **dato = M[direccion];**
 - Escritura: **M[direccion] = dato;**

- ¡Atención! Desde el punto de vista del programador:

- La MP se direcciona a nivel de byte.
 - Los accesos a memoria pueden ser de múltiples tamaños: 1, 2, 4 u 8 bytes.
 - Si leemos 4 bytes en la dirección X, accedemos a las direcciones X, X+1, X+2 y X+3
 - Little endian vs big endian

Introducción

Tipos de Memoria

En función de la **perdurabilidad**:

- Volátil
- No Volátil

En función del **tipo de acceso**:

- Sólo lectura (ROM, Read Only Memory)
- Lectura / Escritura (RAM, Random Access Memory)

En función del **tipo de uso**:

- **Primaria (semiconductores)**
- Secundaria (dispositivos de almacenamiento E/S, magnéticos y ópticos).

En función de la **forma de acceso**:

- Memorias de Acceso Secuencial (cinta VHS)
- Memorias de Acceso Directo (BluRay)

Memorias de Semiconductores

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 **rápidas**, tienen un **alto consumo, pequeñas** (poca capacidad) y **caras**.

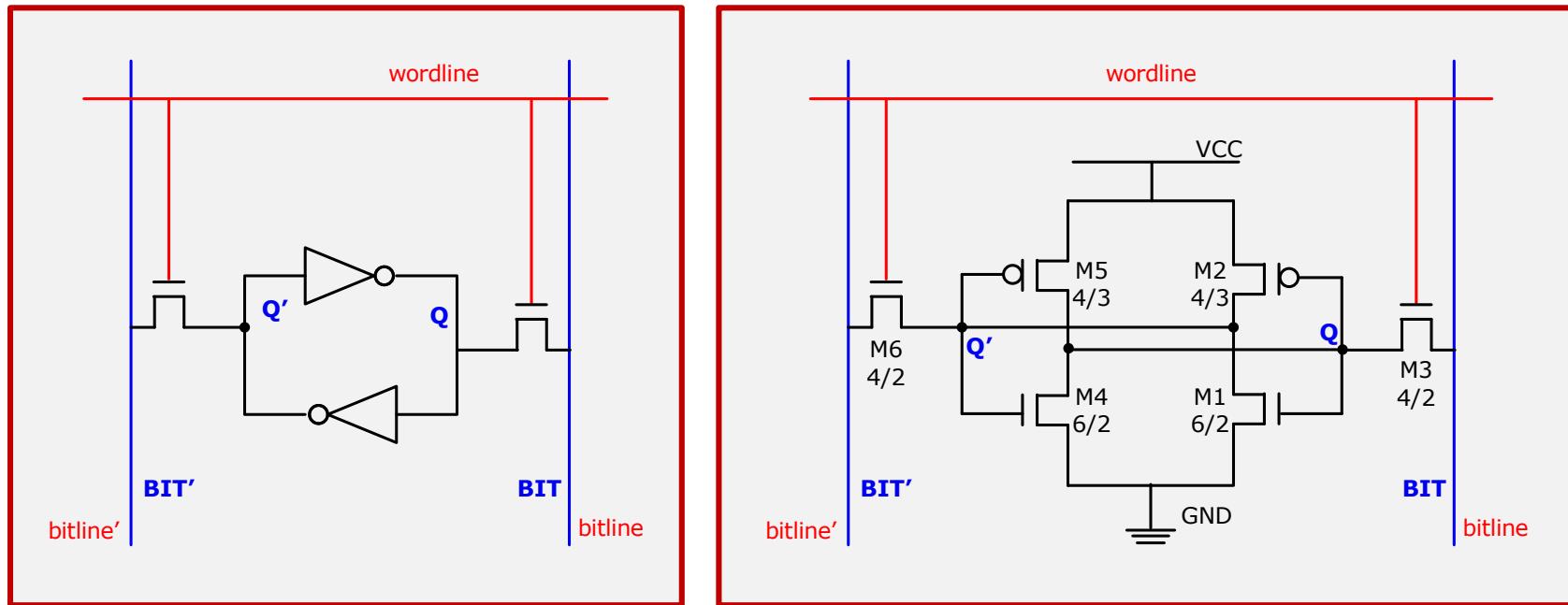
→ Memoria Cache

- **Memoria Dinámica** (DRAM, Dynamic RAM). Cada celda se comporta como un condensador (1-1.x transistores). En comparación con las SRAM son **lentas**, tienen un **bajo consumo, grandes** (mucha capacidad) y **baratas. Problema del refresco**.

→ Memoria Principal

Memorias de Semiconductores

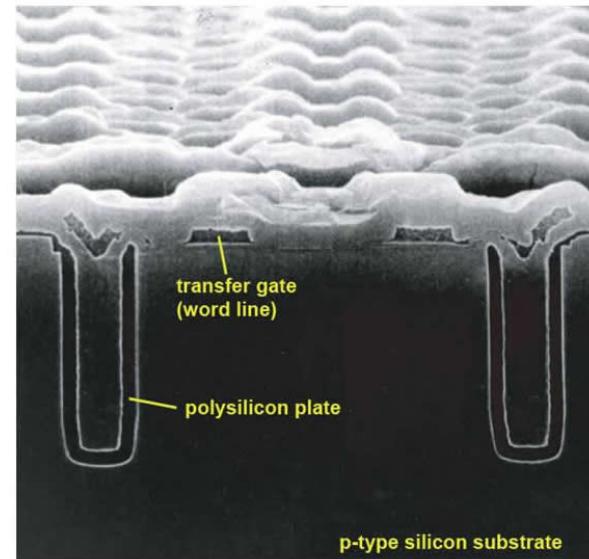
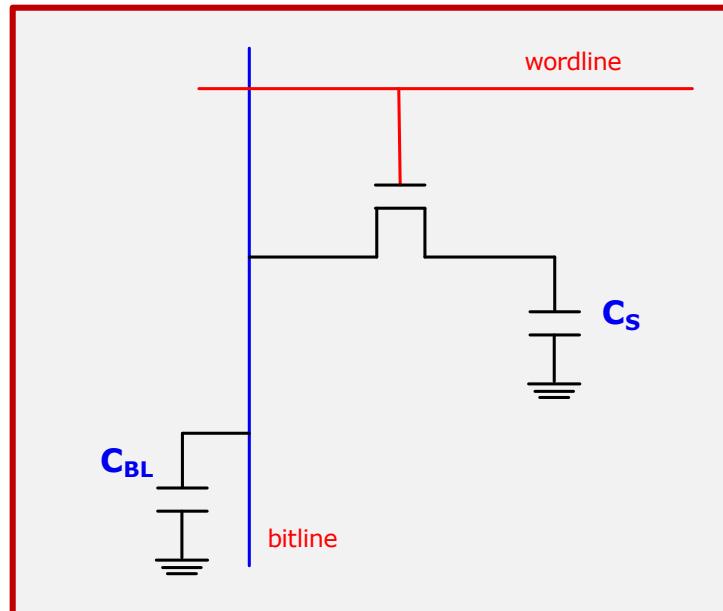
Celda SRAM de 6 Transistores



- ❑ La información se almacena en 2 inversores acoplados.
- ❑ Al activar la word line el dato almacenado se lee a través de las bit lines.
- ❑ Se obtiene el dato negado y sin negar.

Memorias de Semiconductores

Celda DRAM de 1 Transistor

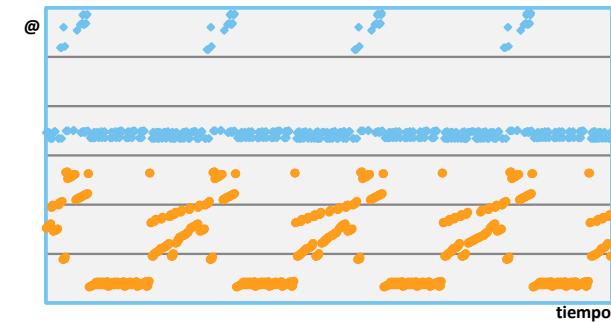
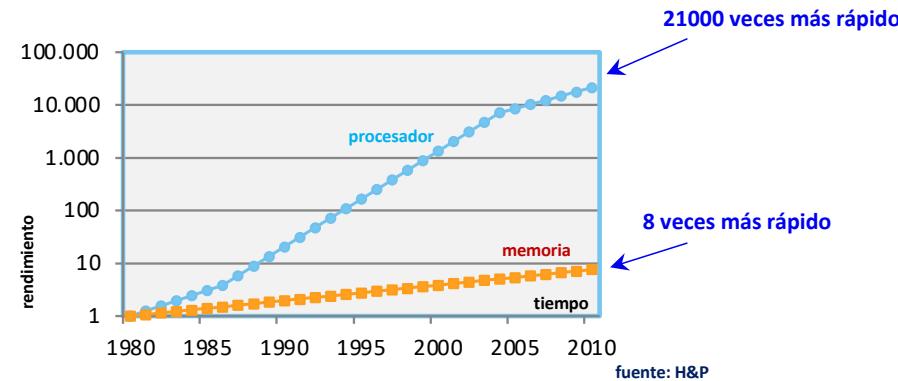


IEEE Solid-State Circuits Society

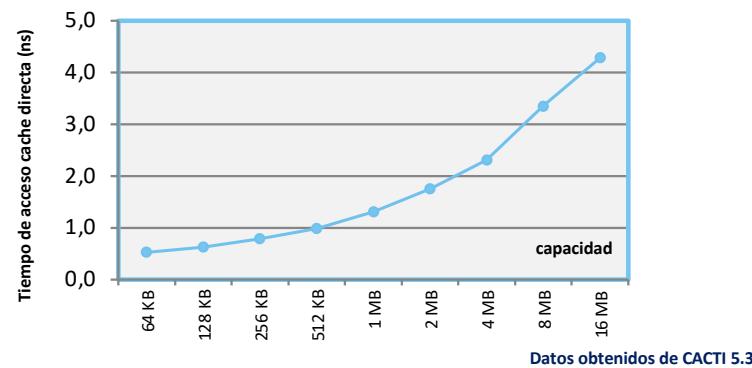
- ❑ La información se almacena en el condensador C_S .
- ❑ Al activar la wordline el dato almacenado en C_S se lee a través de la bitline.
- ❑ El condensador se va descargando poco a poco, es necesario recargarlo regularmente (refresco).

Visión General

- La Existencia de la Memoria Cache está justificada por las siguientes situaciones:
 - Velocidad Procesadores >> Velocidad Memorias.
 - Propiedades de los Programas.



- Propiedades de las memorias.



Velocidad Procesadores >> Velocidad Memorias

Año	Procesador	Freq.	Número máximo referencias/ciclo	Ancho Banda necesario
1991	DEC Alpha 21064	200 MHz	2.5	2 GB/s
2000	DEC Alpha 21264	600 MHz	5	12 GB/s
2003	Intel Pentium 4	3 GHz	3.75	45 GB/s
2010	Intel Xeon X5680 (6 cores)	3,33 GHz	22.5	300 GB/s

Año	Tipo DRAM (Capacidad)	Freq. Placa Base	Latencia 1r dato / resto	Ancho Banda 32 bytes	Ancho Banda de pico
2000	EDO DRAM (64 Mb)	66 MHz	50/20 ns	290.9 MB/s	400 MB/s
2002	SDRAM (128 Mb)	167 MHz	36/6 ns	592.6 MB/s	1,33 GB/s
2003	DDR SDRAM (256 Mb)	200 MHz	30/2.5 ns	853.3 MB/s	3,2 GB/s
2010	DDR3 SDRAM (2 Gb)	1,1 GHz	17/0.45 ns	1,74 GB/s	17,6 GB/s

Propiedades de las Memorias

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 **rápidas**, tienen un **alto consumo, 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 **lentas**, tienen un **bajo consumo, grandes** (mucha capacidad) y **baratas. Problema del refresco**.

→ Memoria Principal

Propiedades de las Memorias

Tipo	Capacidad	Tiempo acceso	Tecnología	Coste por GB
Registros	64 – 1024 B	0.3 – 1ns	-	(1)
Memoria Cache	32KB – 32 MB	1 - 5 ns	Semiconductor SRAM	8500 \$
Memoria Principal	1GB – 16 GB	10 – 30 ns	Semiconductor DRAM	4.5 €
Memoria Secundaria	500 GB – 10 TB	10 – 50 ms	Disco Duro Magnética	0.035 €

Datos de abril de 2022

(1) Integrado en la CPU, no se puede comprar por separado

↑Capacidad ↔ ↑Tiempo de acceso ↔ ↓Coste por Mbyte

↓Capacidad ↔ ↓Tiempo de acceso ↔ ↑Coste por Mbyte

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.
- ❑ Spec2000

%@	% referencias a datos	% referencias a Instrucciones
1%	79,6%	90,7%
2%	84,8%	97,3%
5%	90,4%	99,6%
10%	93,7%	100,0%

El 2% de las posiciones de memoria de datos reciben el 84,8% de todas las referencias a datos.

El 5% de todas las instrucciones reciben el 99,6% de todas las referencias a instrucciones.

Propiedades de los Programas

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.

```
X3D(int v[], int w[]) {  
    int vert, polig, color;  
    ...  
    for (i=0; i<5000; i++) {  
        v[i] = w[i] | 0x01;  
        v[i] = v[i] * vert;  
        if (v[i] != 0)  
            w[i] = polig / v[i];  
        v[i] = w[i] + color;  
    }  
    ...  
}
```

¿Por qué? Por los bucles.

- **Instrucciones.** Dentro de un bucle se accede repetidamente a las mismas instrucciones.
- **Datos.** Dentro de un bucle se accede repetidamente a las mismas variables.

Propiedades de los Programas

Localidad Espacial

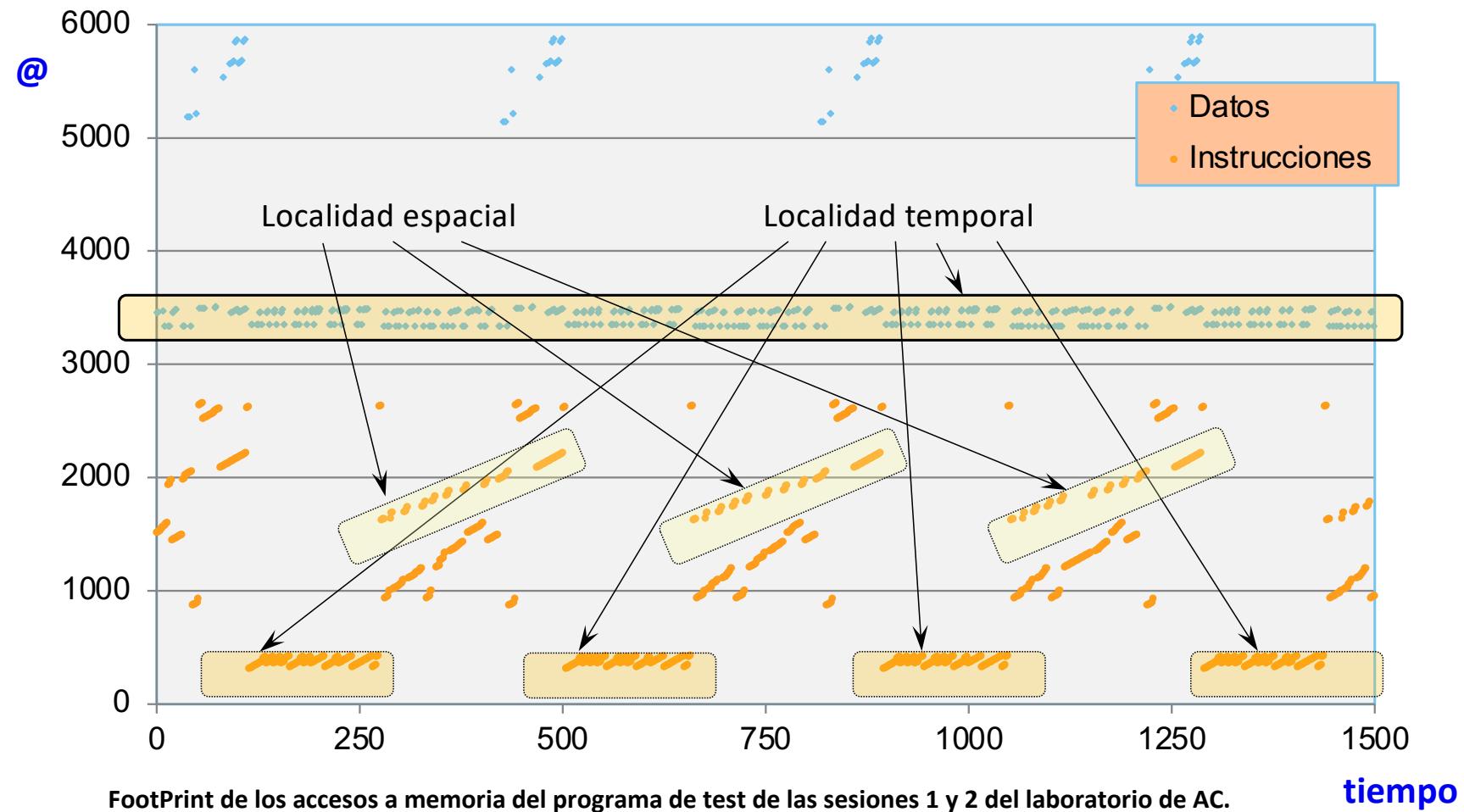
- Si accedemos a una posición de memoria, es muy probable que se acceda a posiciones próximas en un futuro cercano.

```
X3D(int v[], int w[]) {  
    int vert, polig, color;  
    ...  
    for (i=0; i<5000; i++) {  
        v[i] = w[i] | 0x01;  
        v[i] = v[i] * vert;  
        if (v[i] != 0)  
            w[i] = polig / v[i];  
        v[i] = w[i] + color;  
    }  
    ...  
}
```

¿Por qué?

- Instrucciones.** Las instrucciones se ejecutan en secuencia.
- Datos.** Los vectores y matrices se suelen recorrer completos en secuencia; los parámetros y variables locales suelen estar en el bloque de activación de la subrutina..

Propiedades de los programas



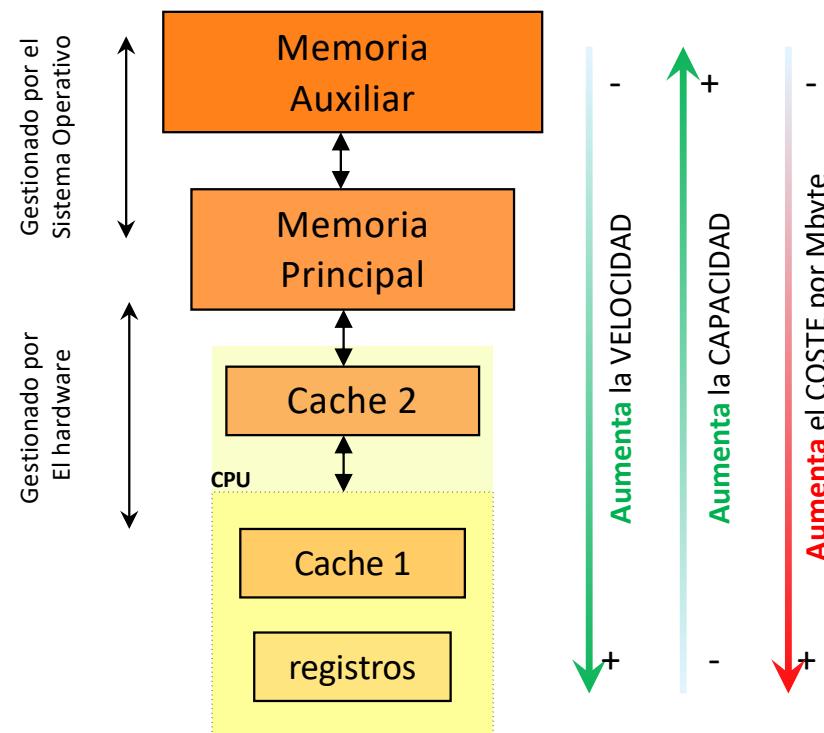
Propiedades de los Programas

¿Cómo podemos aprovechar la localidad?

- **Localidad Temporal.** Si traemos un dato (o instrucción) de memoria, sería útil guardarlos “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 cuesta un poco más que traer un solo dato.

Solución: Jerarquía de Memorias



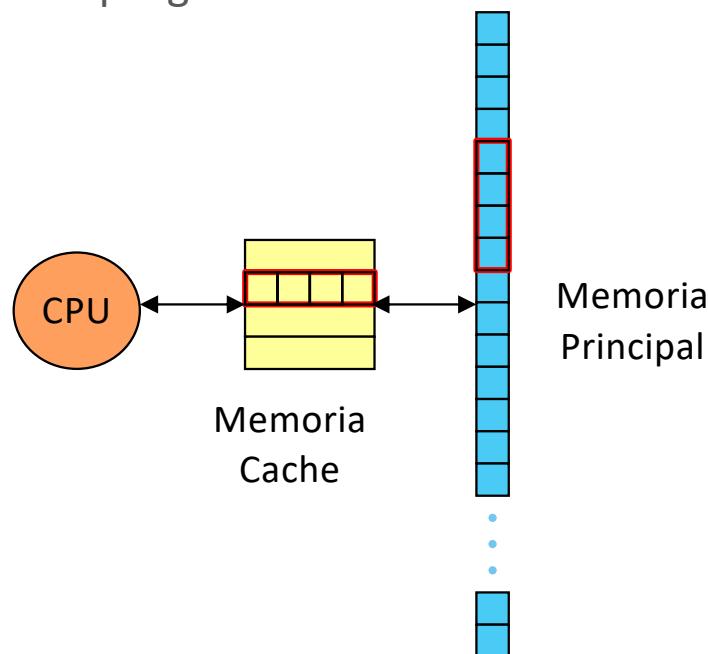
Objetivo: que cuando el procesador acceda a un dato, éste se encuentre en los niveles de la jerarquía más cercanos al procesador.

Si se consigue, obtendríamos una memoria con la velocidad de los niveles rápidos, el tamaño de los niveles más grandes y todo ello con un coste razonable.

¡La jerarquía funciona por las propiedades (localidad) de los programas!

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.



- **Objetivo:**
 - **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 cercana a la recientemente usada.
- **Conceptos:**
 - Acierto / Fallo
 - Línea / Bloque de memoria
 - Algoritmos emplazamiento
 - Algoritmos reemplazo
 - Políticas de Escritura
 - Evaluación

Terminología y Funcionamiento

```
for (i=0; i<200; i++)
    sum = sum + V[i];
```

```
la $t0,V
move $t1,$zero
move $t2,$zero
for: bge $t1,$t2,end
    lw $t3,0($t0)
    addu $t2,$t2,$t3
    addu $t0,$t0,4
    addu $t1,$t1,1
    b for
end:
```

- ❑ **Referencia a Memoria.** Acceso a Memoria
- ❑ **Acierto (HIT).** El dato accedido por la CPU **ESTÁ** en Memoria Cache.
- ❑ **Fallo (MISS).** El dato al que intenta acceder la CPU **NO ESTÁ** en Memoria Cache.
- ❑ ¿Qué hacemos en caso de FALLO?
 - Hay que traer el dato que provoca el fallo a Memoria Cache.
 - No sólo traemos ese dato, traemos un **bloque de datos** (p.e. 16 bytes).

Terminología y Funcionamiento

```
for (i=0; i<200; i++)  
    sum = sum + V[i];
```

```
la $t0,V  
move $t1,$zero  
move $t2,$zero  
for: bge $t1,$t2,end  
    lw $t3,0($t0)  
    addu $t2,$t2,$t3  
    addu $t0,$t0,4  
    addu $t1,$t1,1  
    b for  
end:
```



- Accedemos a V[0] ⇒ **FALLO**
 - Traemos a MC Bloque de datos: V[0] – V[3] (16 bytes)
- Accedemos a V[1] ⇒ **ACIERTO**
- Accedemos a V[2] ⇒ **ACIERTO**
- Accedemos a V[3] ⇒ **ACIERTO**
- Accedemos a V[4] ⇒ **FALLO**
 - Traemos a MC Bloque de datos: V[4] – V[7] (16 bytes)
- Accedemos a V[5] ⇒ **ACIERTO**
- ...

**LOCALIDAD
ESPACIAL**

Terminología y Funcionamiento

la... mo... mo... bg... lw... add... add... add... b...

```
la $t0,V  
move $t1,$zero  
move $t2,$zero  
for: bge $t1,$t2,end  
lw $t3,0($t0)  
addu $t2,$t2,$t3  
addu $t0,$t0,4  
addu $t1,$t1,1  
b for  
end:
```

- ❑ fetch “la...” ⇒ **FALLO**
 - Traemos a MC Bloque de inst: “la ... bge” (16 bytes)
- ❑ fetch “move ...” ⇒ **ACIERTO**
- ❑ fetch “move ...” ⇒ **ACIERTO**
- ❑ fetch “bge ...” ⇒ **ACIERTO**
- ❑ fetch “lw ...” ⇒ **FALLO**
 - Traemos a MC Bloque de inst: “lw ... add” (16 bytes)
- ❑ fetch “add ...” ⇒ **ACIERTO**
- ❑ fetch “add ...” ⇒ **ACIERTO**
- ❑ fetch “add ...” ⇒ **ACIERTO**
- ❑ fetch “b ...” ⇒ **FALLO**
 - Traemos a MC Bloque de inst: “b ...” (16 bytes)
- ❑ fetch “bge ...” ⇒ **ACIERTO**
- ❑ fetch “lw ...” ⇒ **ACIERTO**
- ❑ ...

LOCALIDAD
TEMPORAL

LOCALIDAD
ESPACIAL

Terminología y Funcionamiento

```
for (i=0; i<200; i++)
    sum = sum + V[i];
```

```
la $t0,V
move $t1,$zero
move $t2,$zero
for: bge $t1,$t2,end
    lw $t3,0($t0)
    addu $t2,$t2,$t3
    addu $t0,$t0,4
    addu $t1,$t1,1
    b for
end:
```

- ❑ **Referencia a Memoria.** Acceso a Memoria
- ❑ **Acierto (HIT).** El dato accedido por la CPU **ESTÁ** en Memoria Cache.
- ❑ **Fallo (MISS).** El dato al que intenta acceder la CPU **NO ESTÁ** en Memoria Cache.
- ❑ **Tasa de aciertos (hit ratio).** Porcentaje de aciertos.
 - $h = \# \text{aciertos} / \# \text{referencias}$
- ❑ **Tasa de fallos (miss ratio).** Porcentaje de fallos.
 - $m = \# \text{fallos} / \# \text{referencias}$
 - $m = 1-h$

Problema, calculad la tasa de fallos (m)

```
for (i=0; i<200; i++)
    sum = sum + V[i];
```

```
la $t0,V
move $t1,$zero
move $t2,$zero
for: bge $t1,$t2,end
    lw $t3,0($t0)
    addu $t2,$t2,$t3
    addu $t0,$t0,4
    addu $t1,$t1,1
    b for
end:
```

❑ Accesos a Memoria:

- A instrucciones: 1205
- A datos: 200

❑ Fallos de Cache (consideramos 16 bytes por línea)

- Instrucciones: 3 fallos
- Datos: 50 fallos (1 de cada 4)

❑ Tasa de fallos:

- $m_I = 3 / 1205 = 0,0025 = 0,25\%$
- $m_D = 50/200 = 0,25 = 25\%$
- $m = 53/1405 = 0,038 = 3,8\%$

Terminología y Funcionamiento

```
for (i=0; i<200; i++)
    sum = sum + V[i];
```

```
la $t0,V
move $t1,$zero
move $t2,$zero
for: bge $t1,$t2,end
    lw $t3,0($t0)
    addu $t2,$t2,$t3
    addu $t0,$t0,4
    addu $t1,$t1,1
    b for
end:
```

□ **Tasa de aciertos (hit ratio).** Porcentaje de aciertos.

$$\circ \quad h = \text{#aciertos} / \text{#referencias}$$

□ **Tasa de fallos (miss ratio).** Porcentaje de fallos.

$$\circ \quad m = \text{#fallos} / \text{#referencias}$$

$$\circ \quad m = 1-h$$

□ **¿De qué depende la tasa de fallos?**

○ Del tamaño de la cache

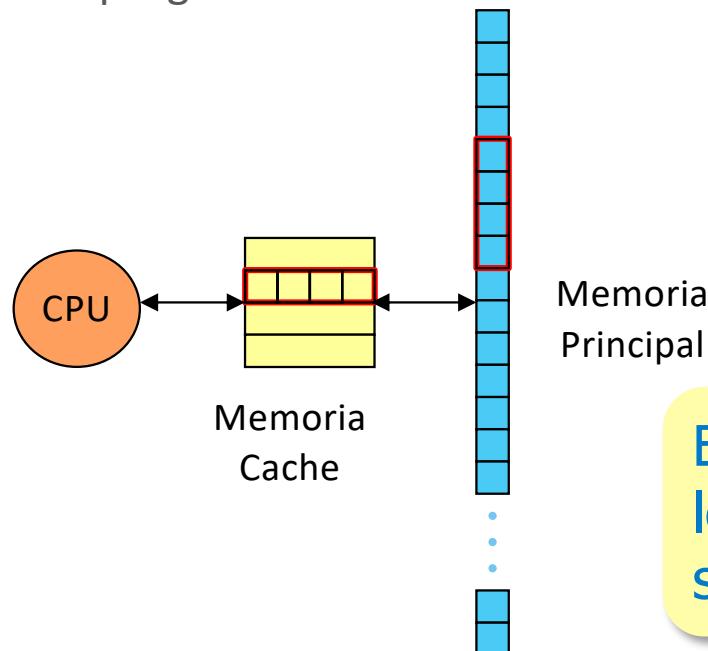
○ Del tamaño de línea

○ De los algoritmos de emplazamiento y reemplazo

○ Del programa evaluado

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.



Tenemos muchos Problemas que resolver todavía:

- Cuando traemos un bloque de MP a MC, ¿dónde lo ponemos?
- Cuando hacemos un acceso a MC, ¿cómo buscamos el dato?
- Cuando traemos un bloque de MP y no cabe en MC, ¿que bloque de MC eliminamos?
- Cuando hacemos un acceso en escritura, dónde lo hacemos, ¿en MC? ¿en MP? ¿en las dos? Al final, las escrituras tienen que ir a MP

El sistema funciona, porque el coste de leer/escribir un bloque de datos de MP es similar al coste de leer/escribir una palabra.

Principios de Funcionamiento de las Memorias Cache

La memoria cache (MC) es una memoria de acceso rápido organizada en líneas (bloques)

xx XX	xx XX	xx XX	xx XX
xx XX	xx XX	xx XX	xx XX
xx XX	xx XX	xx XX	xx XX
xx XX	xx XX	xx XX	xx XX

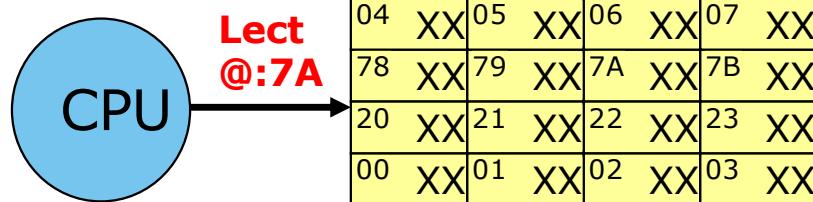
Línea de cache

00 XX	01 XX	02 XX	03 XX
04 XX	05 XX	06 XX	07 XX
08 XX	09 XX	0A XX	0B XX
0C XX	0D XX	0E XX	0F XX
10 XX	11 XX	12 XX	13 XX
14 XX	15 XX	16 XX	17 XX
18 XX	19 XX	1A XX	1B XX

74 XX	75 XX	76 XX	77 XX
78 XX	79 XX	7A XX	7B XX
7C XX	7D XX	7E XX	7F XX

Principios de Funcionamiento de las Memorias Cache

Cuando la CPU realiza un acceso a memoria, en primer lugar se busca el dato en la memoria cache.

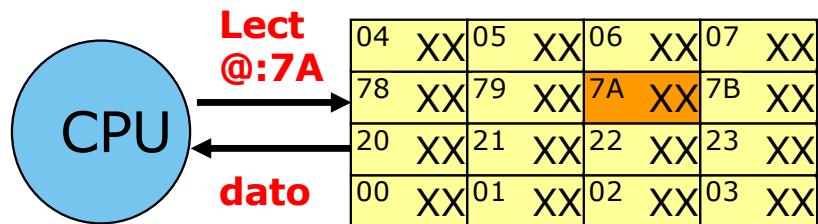


00 XX	01 XX	02 XX	03 XX
04 XX	05 XX	06 XX	07 XX
08 XX	09 XX	0A XX	0B XX
0C XX	0D XX	0E XX	0F XX
10 XX	11 XX	12 XX	13 XX
14 XX	15 XX	16 XX	17 XX
18 XX	19 XX	1A XX	1B XX

74 XX	75 XX	76 XX	77 XX
78 XX	79 XX	7A XX	7B XX
7C XX	7D XX	7E XX	7F XX

Principios de Funcionamiento de las Memorias Cache

Si el dato buscado está en la MC se produce un **ACIERTO** (**hit**) y se pasa el dato a la CPU de forma rápida.

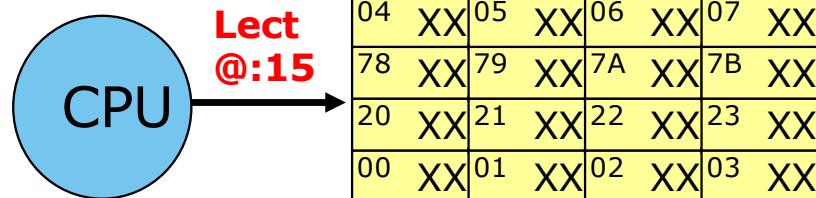


00 XX	01 XX	02 XX	03 XX
04 XX	05 XX	06 XX	07 XX
08 XX	09 XX	0A XX	0B XX
0C XX	0D XX	0E XX	0F XX
10 XX	11 XX	12 XX	13 XX
14 XX	15 XX	16 XX	17 XX
18 XX	19 XX	1A XX	1B XX

74 XX	75 XX	76 XX	77 XX
78 XX	79 XX	7A XX	7B XX
7C XX	7D XX	7E XX	7F XX

Principios de Funcionamiento de las Memorias Cache

Si el dato buscado **NO** está en la MC se produce un **FALLO (miss)** y se ha de buscar en la MP.

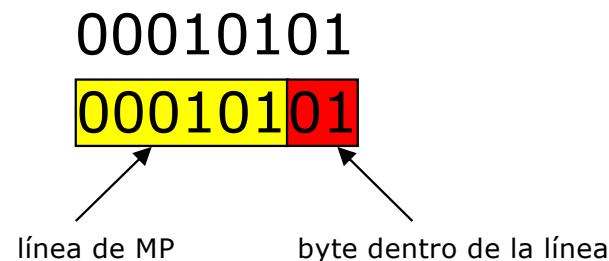
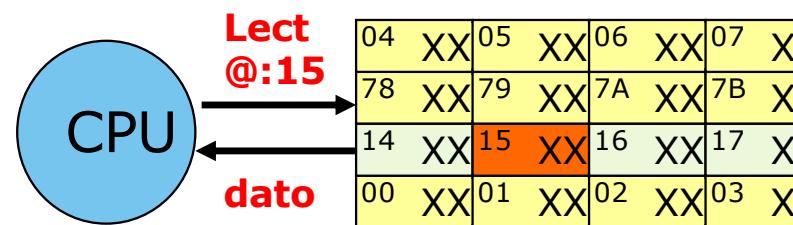


00 XX	01 XX	02 XX	03 XX
04 XX	05 XX	06 XX	07 XX
08 XX	09 XX	0A XX	0B XX
0C XX	0D XX	0E XX	0F XX
10 XX	11 XX	12 XX	13 XX
14 XX	15 XX	16 XX	17 XX
18 XX	19 XX	1A XX	1B XX
.			
.			
.			

74 XX	75 XX	76 XX	77 XX
78 XX	79 XX	7A XX	7B XX
7C XX	7D XX	7E XX	7F XX

Principios de Funcionamiento de las Memorias Cache

Además de leer de MP el dato que provoca el fallo, se lee la línea completa en la que está el dato y se ubica en alguno de los bloques de MC.



00	XX	01	XX	02	XX	03	XX
04	XX	05	XX	06	XX	07	XX
08	XX	09	XX	0A	XX	0B	XX
0C	XX	0D	XX	0E	XX	0F	XX
10	XX	11	XX	12	XX	13	XX
14	XX	15	XX	16	XX	17	XX
18	XX	19	XX	1A	XX	1B	XX
...							

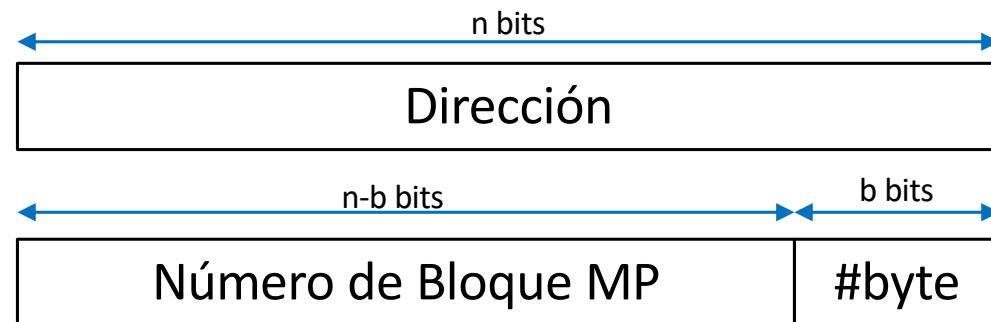
74	XX	75	XX	76	XX	77	XX
78	XX	79	XX	7A	XX	7B	XX
7C	XX	7D	XX	7E	XX	7F	XX

Principios de Funcionamiento de las Memorias Cache

- En cualquier Memoria Cache hay que definir:
 - **Algoritmo de Emplazamiento.** Determina en qué bloques de MC puede colocarse una línea. Determina, también, dónde hay que buscar un dato.
 - **Algoritmo de reemplazo.** Determina qué línea se ha de eliminar de la cache para dejar espacio a una nueva línea.
 - **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.

Organización de la MP

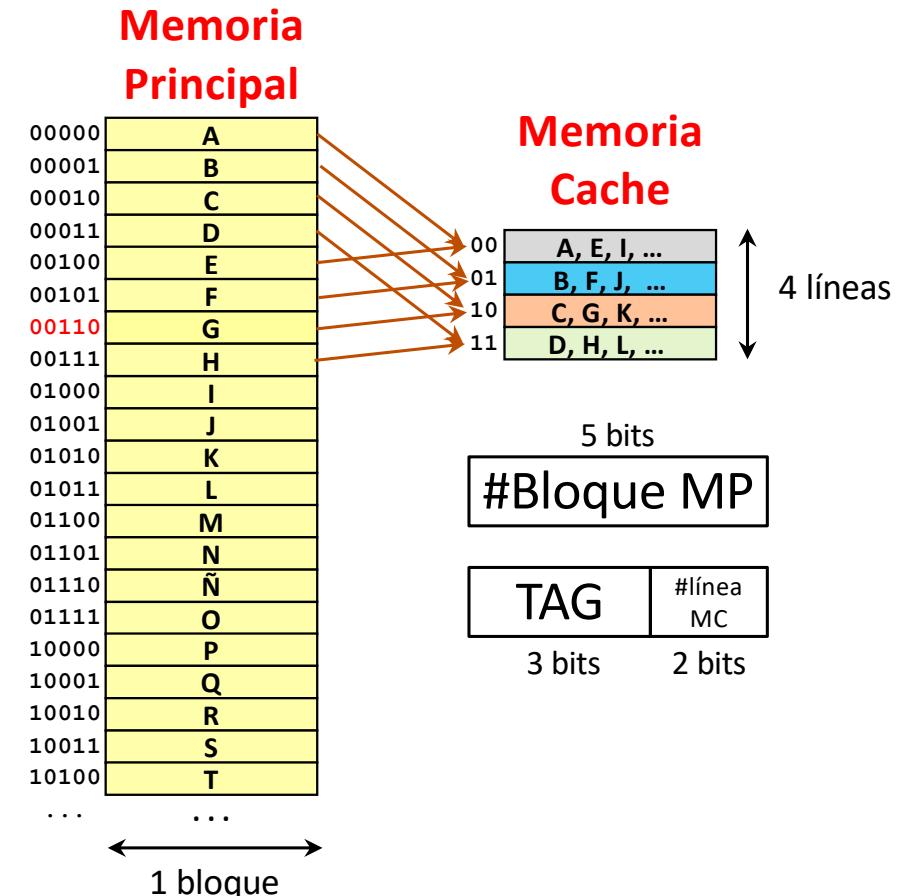
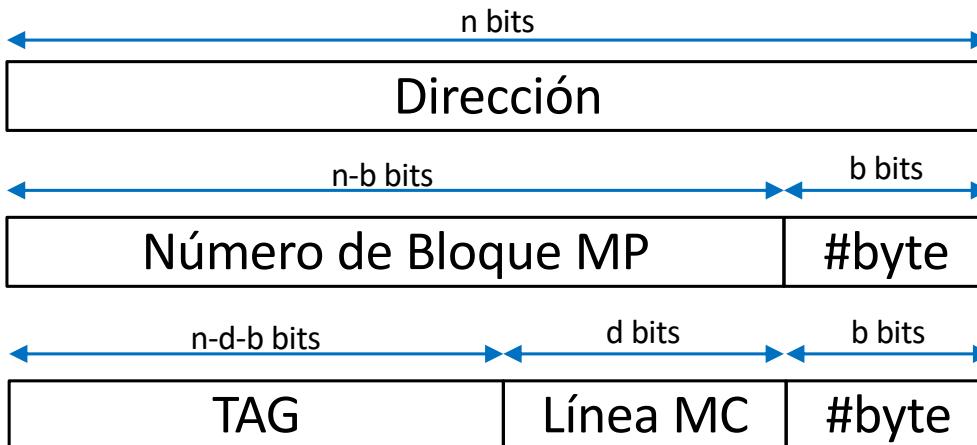
- ❑ La Memoria Principal (MP) se organiza en bloques de tamaño fijo ($B = 2^b$ bytes)
- ❑ Dada una dirección,
 - ¿Cómo averiguamos el número de bloque?
Número de bloque = Dirección / Tamaño bloque
 - ¿Cuál es el byte (offset, desplazamiento) dentro del bloque?
Byte dentro del bloque = Dirección % Tamaño bloque
- ❑ Como el tamaño de bloque siempre es potencia de 2, este cálculo es muy simple:



Dirección	número de bloque
0	0
1	
2	
3	
4	1
5	
6	
7	
8	2
9	
10	
11	
12	3
13	
14	
15	
$2^{32}-1$	

Cache Directa

- En una Memoria Cache, podemos almacenar D bloques de Memoria Principal.
- La forma más sencilla de implementar una Memoria Cache es una **Memoria Cache Directa**
⇒ **Un Bloque de MP se almacena siempre en la misma línea (o bloque) de Memoria Cache**
- Para que sea fácil de implementar, hacemos que el número de bloques de una Memoria Cache Directa sea potencia de dos ($D = 2^d$).



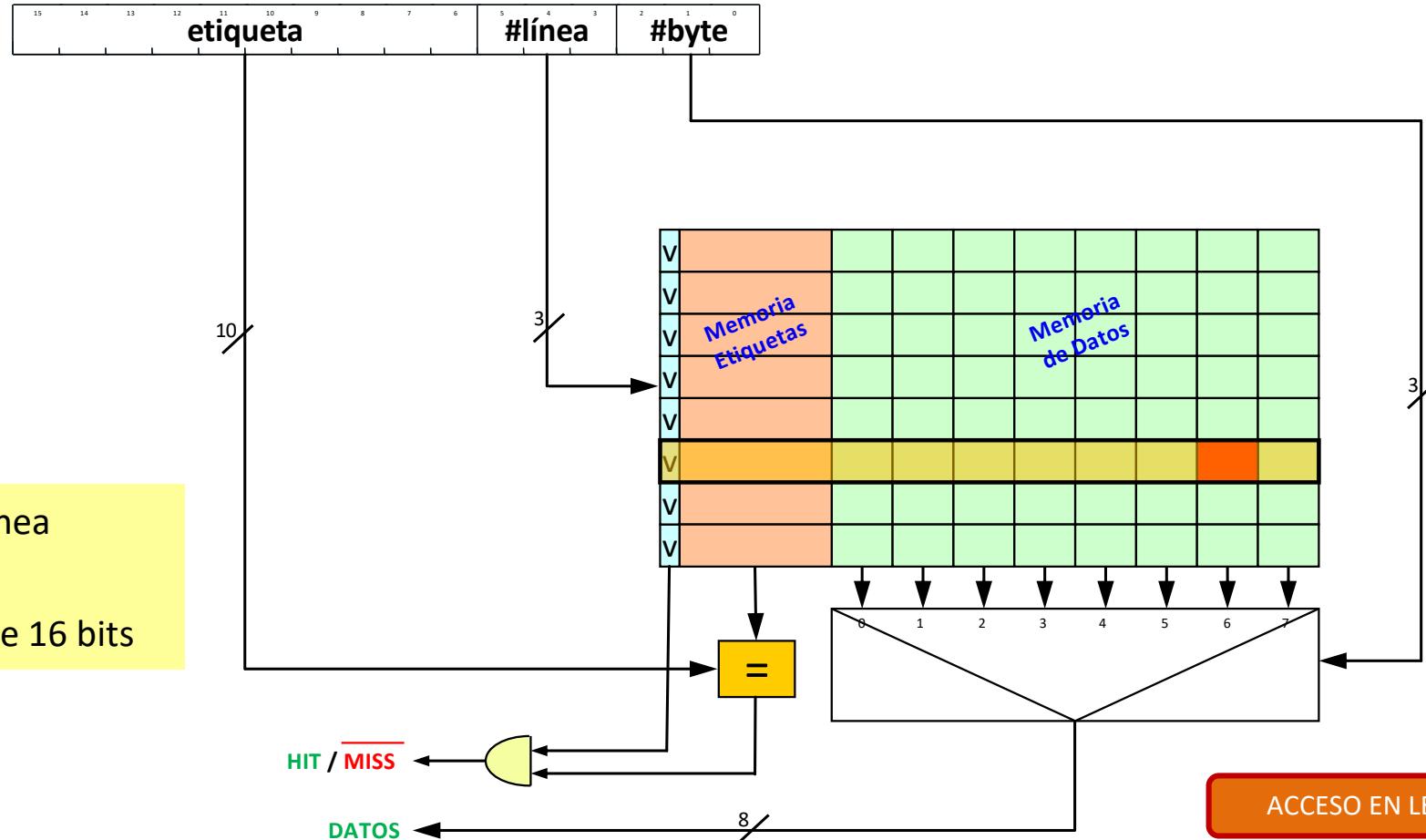
Cache Directa

- ❑ Podemos ver una Memoria Cache Directa como una tabla
- ❑ En cada entrada de la tabla tenemos 1 línea de cache.
- ❑ La información que almacenamos de cada línea es:
 - **DATOS**, contiene los 2^b bytes del bloque
 - **ETIQUETA**, información necesaria para identificar el bloque de MP
 - **BIT de VALIDEZ**, indica si la línea está ocupada

#Línea MC	V	Etiqueta	DATOS
0			
1			
2			
3			

Cache Directa

- 8 bytes por línea
- 8 líneas
- Direcciones de 16 bits



Problema

- Suponiendo un tamaño de bloque $B = 64$ bytes y una cache directa con 32 bloques. Calculad número de línea de MP, la etiqueta, número de línea de MC y desplazamiento para las direcciones de memoria:



- $A = 0x100101C0 = 0001\ 0000\ 0000\ 0001\ 0000\ 0001\ 1100\ 0000 =$
 $= 00\ 0100\ 0000\ 0000\ 0100\ 0000\ 0111\ 00\ 0000 = 0x0400407, 0x00$
 $= 0\ 0010\ 0000\ 0000\ 0010\ 0000\ 0\ 0111\ 00\ 0000 = 0x20020, 0x07, 0x00$
- $A = 0x1001060F = 0001\ 0000\ 0000\ 0001\ 0000\ 0110\ 0000\ 1111 =$
 $= 00\ 0100\ 0000\ 0000\ 0100\ 0001\ 1000\ 00\ 1111 = 0x0400418, 0xF$
 $= 0\ 0010\ 0000\ 0000\ 0010\ 0000\ 1\ 1000\ 00\ 1111 = 0x20020, 0x18, 0xF$

Problema

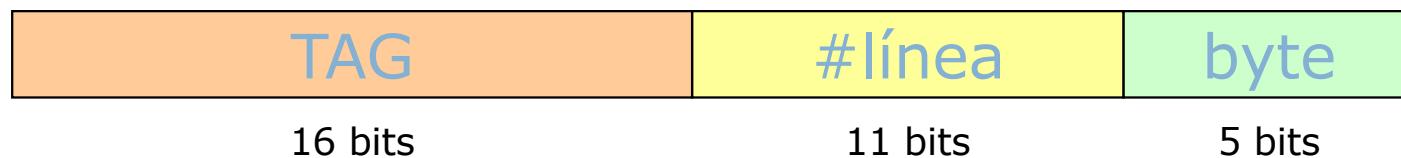
Disponemos de un sistema de Memoria con direcciones de 32 bits y una MC directa de 64Kbytes y 32 bytes por línea.

1. Dada una dirección de memoria indicad, mediante un dibujo, qué bits se utilizan para seleccionar el byte dentro de la línea, qué bits se utilizan para seleccionar la línea de memoria cache, y qué bits forman la etiqueta.
2. ¿Cuántos bits ocupa la memoria de etiquetas (tags)?

Tamaño línea: 32 bytes por línea (2^5)

Número de líneas: 2048 líneas (2^{11})

Tamaño cache: #(bytes por línea) \times #líneas = 64 Kbytes



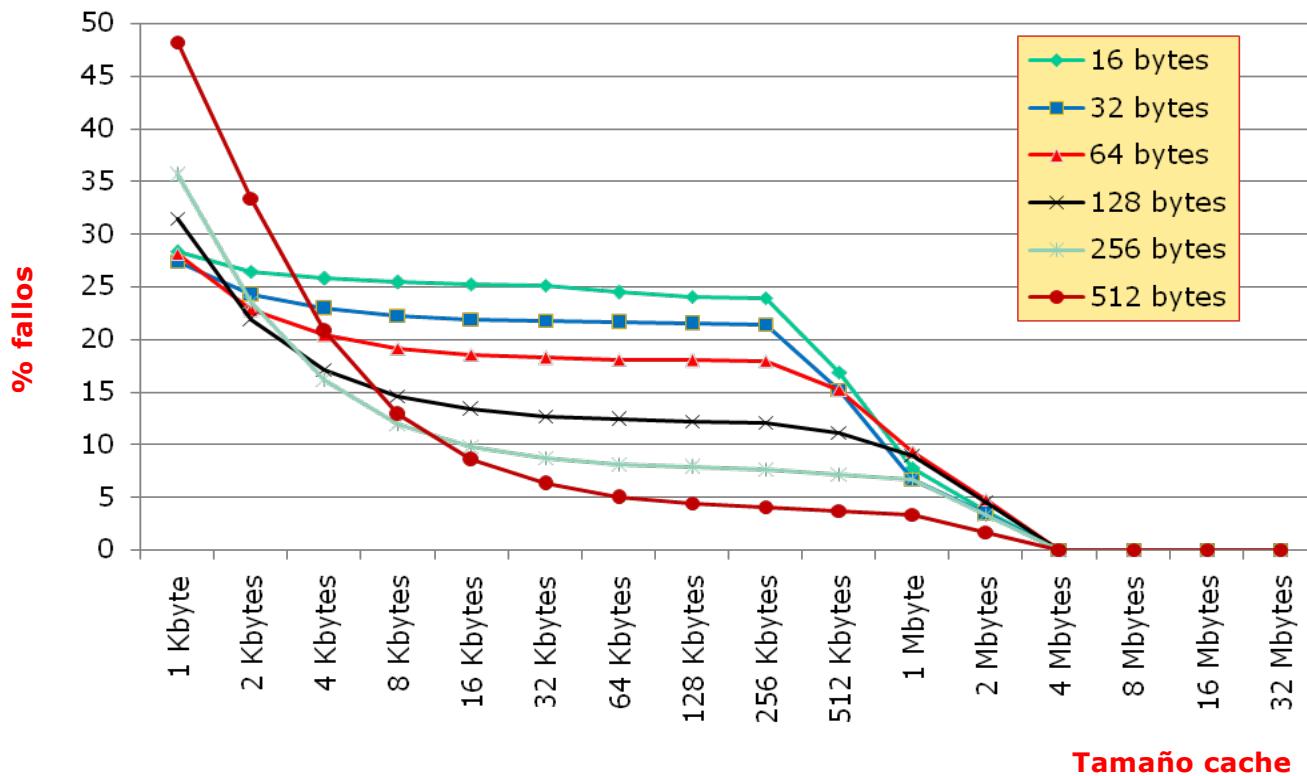
Tamaño Memoria TAGS: 2048 líneas \times 16 bits = 32.768 bits

Influencia del tamaño de línea en la tasa de fallos

- Si mantenemos fijo el tamaño de cache, ¿qué es mejor?
 - Líneas grandes para aprovechar la localidad espacial.
 - Líneas pequeñas para tener menos conflictos

DEPENDE del PROGRAMA

Influencia del tamaño de línea en la tasa de fallos

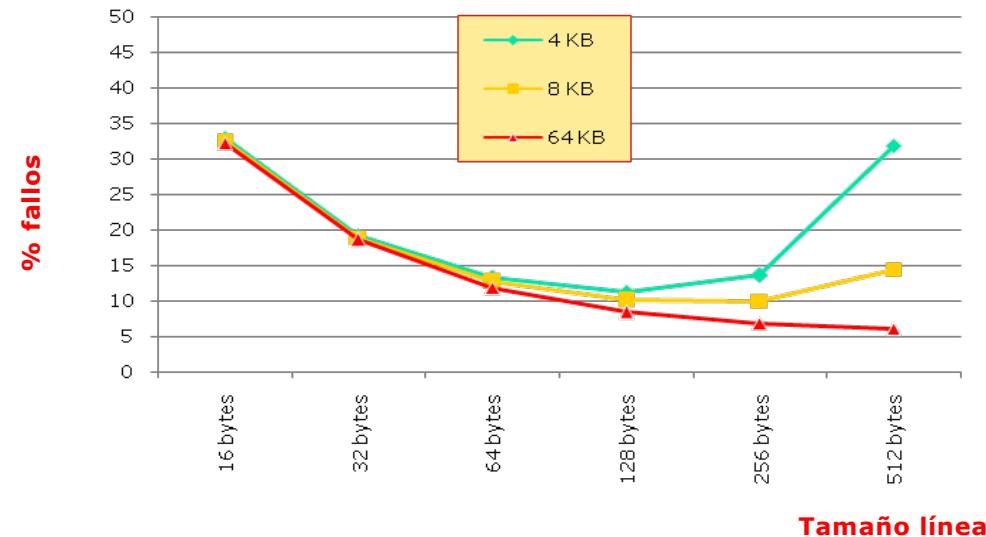


Cache de datos directa.

El tamaño de línea influye, ipero, en ambos sentidos!

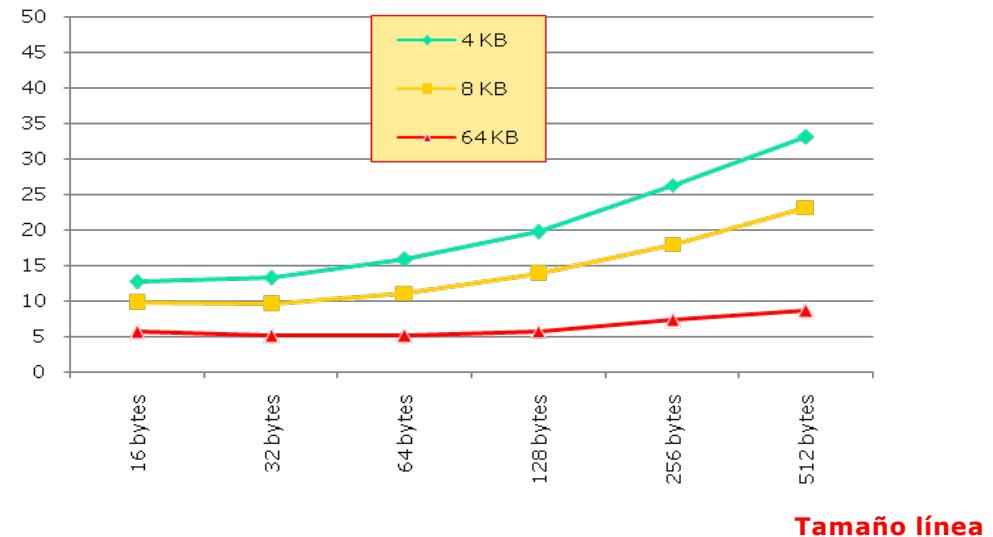
179.art
programa en C
Cálculo científico
Image Recognition

Influencia del tamaño de línea en la tasa de fallos



171.swim

programa en fortran 77, cálculo científico
Shallow Water Modeling



175.vpr

Programa en C, enteros
FPGA Circuit Placement and Routing

Cache de datos directa.

El tamaño de línea influye, ipero, en ambos sentidos!

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 actualizan simultáneamente la MC y la MP.
- El coste del acceso es el tiempo de acceso a MP, pero se puede reducir el tiempo de escritura utilizando buffers.
- La MP siempre está actualizada.

COPY BACK (escritura diferida)

- En una escritura sólo se actualiza MC.
- Para cada línea se añade un bit de control (dirty bit) que indica si la línea ha sido modificada o no.
- Se actualiza la MP cuando la línea (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.

Políticas de Escritura

Premisa: las escrituras, finalmente, se han de hacer en Memoria Principal.

¿Qué hacer en caso de fallo en escritura?

WRITE ALLOCATE (con migración/asignación en caso de fallo)

- Se trae la línea de MP a MC y después se realiza la escritura.

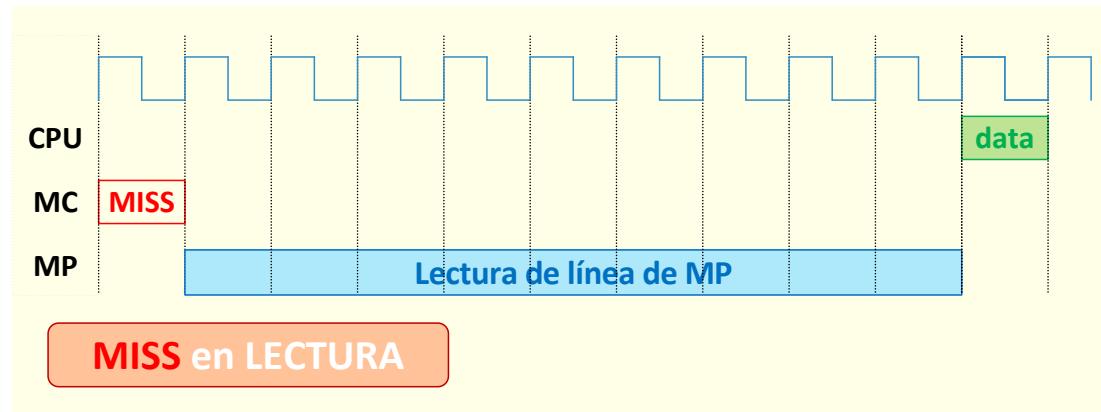
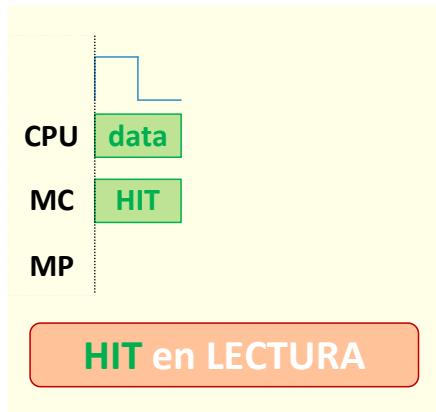
WRITE NO ALLOCATE (sin migración/asignación en caso de fallo)

- La línea NO se trae a MC. Esto obliga a realizar la escritura directamente en MP.

Normalmente se utiliza:

- COPY BACK + WRITE ALLOCATE
- WRITE THROUGH + WRITE NO ALLOCATE

Operaciones a realizar en un acceso a Cache

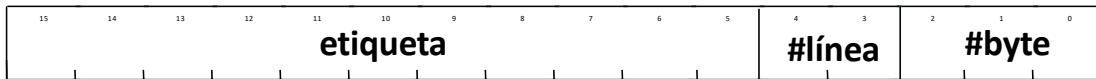


Cache de sólo Lectura, por ejemplo,
una cache de instrucciones

Operaciones a realizar en un acceso a Cache

Cache Directa:

- Direcciones de 16 bits
- Líneas de 8 bytes
- 4 Líneas



Lectura 0x0003
Lectura 0x0004
Lectura 0x0009
Lectura 0x0045
Lectura 0x0047

Bloque MP	DATOS MP								
0x0000	00	03	05	08	F3	87	AD	F4	
0x0001	45	67	89	87	67	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	DA	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
...
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	FF

V	Etiqueta	DATOS MC							
0	0								
1	0								
2	0								
3	0								

Operaciones a realizar en un acceso a Cache

0x0003 =

0000 0000 0000 0011 =
000000000000 00 011

TAG = 0x000

línea = 0x0

byte = 0x3

Bloque MP = 0x0000

Lectura 0x0003: **MISS**

Lectura 0x0004

Lectura 0x0009

Lectura 0x0045

Lectura 0x0047

V	Etiqueta	DATOS MC							
0	0								
1	0								
2	0								
3	0								

Bloque MP	DATOS MP							
0x0000	00	03	05	08	F3	87	AD	F4
0x0001	45	67	89	87	67	90	99	34
0x0002	80	00	23	80	DE	ED	90	54
0x0003	99	AA	DA	80	EF	ED	12	67
0x0004	90	78	AE	10	FE	4E	44	23
0x0005	71	32	F5	10	FE	4E	55	23
0x0006	1A	44	F2	11	FE	64	66	F3
0x0007	1B	23	FF	EE	FF	53	88	7F
0x0008	1C	54	00	00	FF	AA	99	5E
0x0009	BB	34	00	00	34	55	98	4E
0x000A	B4	45	00	00	55	44	45	1E
0x000B	B5	78	00	00	FF	FF	56	12
0x000C	B6	67	56	00	00	CC	DE	E1
...
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF

Operaciones a realizar en un acceso a Cache

$0x0003 =$

$0000 \ 0000 \ 0000 \ 0011 =$
 $000000000000 \ 00 \ 011$

TAG = 0x000

línea = 0x0

byte = 0x3

Bloque MP = 0x0000

Lectura 0x0003: **MISS**

Lectura 0x0004

Lectura 0x0009

Lectura 0x0045

Lectura 0x0047

V	Etiqueta	DATOS MC							
0	1	0x000	00	03	05	08	F3	87	AD
1	0								
2	0								
3	0								

Bloque MP	DATOS MP							
0x0000	00	03	05	08	F3	87	AD	F4
0x0001	45	67	89	87	67	90	99	34
0x0002	80	00	23	80	DE	ED	90	54
0x0003	99	AA	DA	80	EF	ED	12	67
0x0004	90	78	AE	10	FE	4E	44	23
0x0005	71	32	F5	10	FE	4E	55	23
0x0006	1A	44	F2	11	FE	64	66	F3
0x0007	1B	23	FF	EE	FF	53	88	7F
0x0008	1C	54	00	00	FF	AA	99	5E
0x0009	BB	34	00	00	34	55	98	4E
0x000A	B4	45	00	00	55	44	45	1E
0x000B	B5	78	00	00	FF	FF	56	12
0x000C	B6	67	56	00	00	CC	DE	E1
...
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF

Operaciones a realizar en un acceso a Cache

$0x0004 =$

$0000\ 0000\ 0000\ 0100 =$
 $000000000000\ 00\ 100$

$TAG = 0x000$

$\text{l\'ınea} = 0x0$

$\text{byte} = 0x4$

Bloque MP = 0x0000

Lectura 0x0003: **MISS**
 Lectura 0x0004: **HIT**
 Lectura 0x0009
 Lectura 0x0045
 Lectura 0x0047

V	Etiqueta	DATOS MC								
0	1	0x000	00	03	05	08	F3	87	AD	F4
1	0									
2	0									
3	0									

Bloque MP	DATOS MP								
0x0000	00	03	05	08	F3	87	AD	F4	
0x0001	45	67	89	87	67	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	DA	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
...
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Operaciones a realizar en un acceso a Cache

$0x0009 =$

$0000 \ 0000 \ 0000 \ 1001 =$
000000000000 01 001

TAG = 0x000

línea = 0x1

byte = 0x1

Bloque MP = 0x0001

Lectura 0x0003: **MISS**
Lectura 0x0004: **HIT**
Lectura 0x0009: **MISS**
Lectura 0x0045
Lectura 0x0047

V	Etiqueta	DATOS MC								
0	1	0x000	00	03	05	08	F3	87	AD	F4
1	0									
2	0									
3	0									

Bloque MP	DATOS MP								
0x0000	00	03	05	08	F3	87	AD	F4	
0x0001	45	67	89	87	67	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	DA	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
...
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	FF

Operaciones a realizar en un acceso a Cache

$0x0009 =$

$0000 \ 0000 \ 0000 \ 1001 =$
 $000000000000 \ 01 \ 001$

TAG = $0x000$

línea = $0x1$

byte = $0x1$

Bloque MP = $0x0001$

Lectura 0x0003: **MISS**
 Lectura 0x0004: **HIT**
 Lectura 0x0009: **MISS**
 Lectura 0x0045
 Lectura 0x0047

V	Etiqueta	DATOS MC								
0	1	0x000	00	03	05	08	F3	87	AD	F4
1	1	0x000	45	67	89	87	67	90	99	34
2	0									
3	0									

Bloque MP	DATOS MP							
0x0000	00	03	05	08	F3	87	AD	F4
0x0001	45	67	89	87	67	90	99	34
0x0002	80	00	23	80	DE	ED	90	54
0x0003	99	AA	DA	80	EF	ED	12	67
0x0004	90	78	AE	10	FE	4E	44	23
0x0005	71	32	F5	10	FE	4E	55	23
0x0006	1A	44	F2	11	FE	64	66	F3
0x0007	1B	23	FF	EE	FF	53	88	7F
0x0008	1C	54	00	00	FF	AA	99	5E
0x0009	BB	34	00	00	34	55	98	4E
0x000A	B4	45	00	00	55	44	45	1E
0x000B	B5	78	00	00	FF	FF	56	12
0x000C	B6	67	56	00	00	CC	DE	E1
...
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF

Operaciones a realizar en un acceso a Cache

0x0045 =

0000 0000 0100 0101 =
0000000010 00 101

TAG = 0x002

línea = 0x0

byte = 0x5

Bloque MP = 0x0008

Lectura 0x0003: **MISS**
Lectura 0x0004: **HIT**
Lectura 0x0009: **MISS**
Lectura 0x0045: **MISS**
Lectura 0x0047

V	Etiqueta	DATOS MC								
0	1	0x000	00	03	05	08	F3	87	AD	F4
1	1	0x001	45	67	89	87	67	90	99	34
2	0									
3	0									

Bloque MP	DATOS MP								
0x0000	00	03	05	08	F3	87	AD	F4	
0x0001	45	67	89	87	67	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	DA	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Operaciones a realizar en un acceso a Cache

$0x0045 =$

$0000\ 0000\ 0100\ 0101 =$
 $0000000010\ 00\ 101$

TAG = $0x002$

línea = $0x0$

byte = $0x5$

Bloque MP = $0x0008$

Lectura 0x0003: **MISS**
 Lectura 0x0004: **HIT**
 Lectura 0x0009: **MISS**
 Lectura 0x0045: **MISS**
 Lectura 0x0047

V	Etiqueta	DATOS MC								
0	1	0x002	1C	54	00	00	FF	AA	99	5E
1	1	0x000	45	67	89	87	67	90	99	34
2	0									
3	0									

Bloque MP	DATOS MP							
0x0000	00	03	05	08	F3	87	AD	F4
0x0001	45	67	89	87	67	90	99	34
0x0002	80	00	23	80	DE	ED	90	54
0x0003	99	AA	DA	80	EF	ED	12	67
0x0004	90	78	AE	10	FE	4E	44	23
0x0005	71	32	F5	10	FE	4E	55	23
0x0006	1A	44	F2	11	FE	64	66	F3
0x0007	1B	23	FF	EE	FF	53	88	7F
0x0008	1C	54	00	00	FF	AA	99	5E
0x0009	BB	34	00	00	34	55	98	4E
0x000A	B4	45	00	00	55	44	45	1E
0x000B	B5	78	00	00	FF	FF	56	12
0x000C	B6	67	56	00	00	CC	DE	E1

0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF

Operaciones a realizar en un acceso a Cache

$0x0047 =$

$0000\ 0000\ 0100\ 0101 =$
0000000010 00 111

TAG = 0x002

línea = 0x0

byte = 0x7

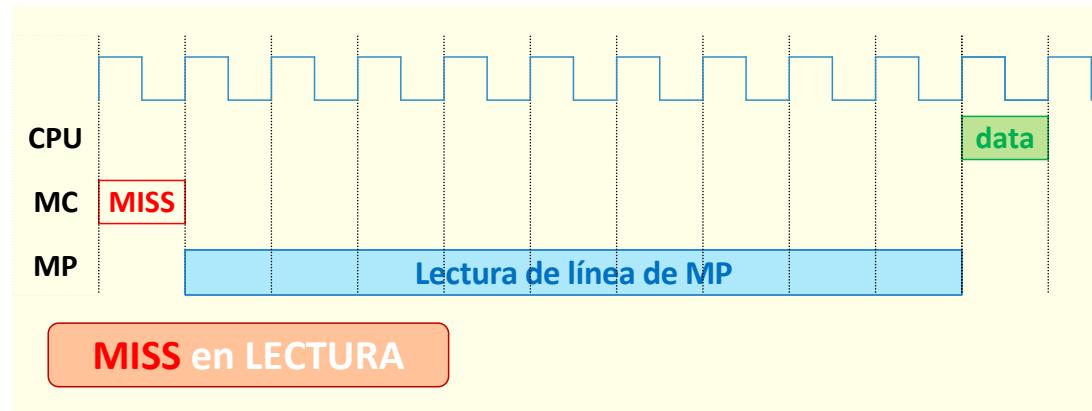
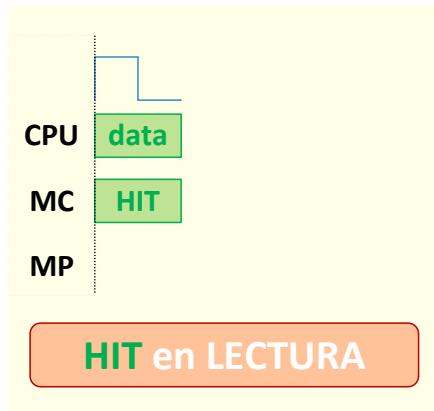
Bloque MP = 0x0008

Lectura 0x0003: **MISS**
Lectura 0x0004: **HIT**
Lectura 0x0009: **MISS**
Lectura 0x0045: **MISS**
Lectura 0x0047: **HIT**

V	Etiqueta	DATOS MC							
0	1	0x002	1C	54	00	00	FF	AA	99
1	1	0x000	45	67	89	87	67	90	99
2	0								
3	0								

Bloque MP	DATOS MP								
0x0000	00	03	05	08	F3	87	AD	F4	
0x0001	45	67	89	87	67	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	DA	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Operaciones a realizar en un acceso a Cache



Write Through +
Write NO Allocate

Operaciones a realizar en un acceso a Cache

$0x0003 =$

$0000 \ 0000 \ 0000 \ 0011 =$
 $000000000000 \ 00 \ 011$

$TAG = 0x000$

$\text{l\'ınea} = 0x0$

$\text{byte} = 0x3$

Bloque MP = 0x0000

Escr 0x0003: **MISS**

Lect 0x0004:

Escr 0x0007:

Lect 0x001F:

Escr 0x001A:

V	Etiqueta	DATOS MC								
0	1	0x002	1C	54	00	00	FF	AA	99	5E
1	1	0x000	45	67	89	87	67	90	99	34
2	0									
3	0									

Bloque MP	DATOS MP								
0x0000	00	03	05	FF	F3	87	AD	F4	
0x0001	45	67	89	87	67	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	DA	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
...	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Write Through + Write NO Allocate



Operaciones a realizar en un acceso a Cache

$0x0004 =$

$0000\ 0000\ 0000\ 0100 =$
 $000000000000\ 00\ 100$

TAG = $0x000$

línea = $0x0$

byte = $0x4$

Bloque MP = $0x0000$

Escr 0x0003: **MISS**

Lect 0x0004: **MISS**

Escr 0x0007:

Lect 0x001F:

Escr 0x001A:

V	Etiqueta	DATOS MC								
0	1	0x002	1C	54	00	00	FF	AA	99	5E
1	1	0x000	45	67	89	87	67	90	99	34
2	0									
3	0									

Bloque MP	DATOS MP								
0x0000	00	03	05	FF	F3	87	AD	F4	
0x0001	45	67	89	87	67	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	DA	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
...	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Write Through + Write NO Allocate

Operaciones a realizar en un acceso a Cache

$0x0004 =$

$0000 \ 0000 \ 0000 \ 0100 =$
 $000000000000 \ 00 \ 100$

TAG = $0x000$

línea = $0x0$

byte = $0x4$

Bloque MP = $0x0000$

Escr 0x0003: **MISS**
 Lect 0x0004: **MISS**
 Escr 0x0007:
 Lect 0x001F:
 Escr 0x001A:

V	Etiqueta	DATOS MC								
0	1	0x000	00	03	05	FF	F3	87	AD	F4
1	1	0x000	45	67	89	87	67	90	99	34
2	0									
3	0									

Bloque MP	DATOS MP							
0x0000	00	03	05	FF	F3	87	AD	F4
0x0001	45	67	89	87	67	90	99	34
0x0002	80	00	23	80	DE	ED	90	54
0x0003	99	AA	DA	80	EF	ED	12	67
0x0004	90	78	AE	10	FE	4E	44	23
0x0005	71	32	F5	10	FE	4E	55	23
0x0006	1A	44	F2	11	FE	64	66	F3
0x0007	1B	23	FF	EE	FF	53	88	7F
0x0008	1C	54	00	00	FF	AA	99	5E
0x0009	BB	34	00	00	34	55	98	4E
0x000A	B4	45	00	00	55	44	45	1E
0x000B	B5	78	00	00	FF	FF	56	12
0x000C	B6	67	56	00	00	CC	DE	E1

0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF

Write Through + Write NO Allocate

Operaciones a realizar en un acceso a Cache

$0x0007 =$

$0000\ 0000\ 0000\ 0111 =$
 $000000000000\ 00\ 111$

TAG = 0x000

línea = 0x0

byte = 0x7

Bloque MP = 0x0000

Escr 0x0003: **MISS**
 Lect 0x0004: **MISS**
 Escr 0x0007: **HIT**
 Lect 0x001F:
 Escr 0x001A:

V	Etiqueta	DATOS MC							
0	1	0x000	00	03	05	FF	F3	87	AD
1	1	0x000	45	67	89	87	67	90	99
2	0								
3	0								

Bloque MP	DATOS MP							
0x0000	00	03	05	FF	F3	87	AD	00
0x0001	45	67	89	87	67	90	99	34
0x0002	80	00	23	80	DE	ED	90	54
0x0003	99	AA	DA	80	EF	ED	12	67
0x0004	90	78	AE	10	FE	4E	44	23
0x0005	71	32	F5	10	FE	4E	55	23
0x0006	1A	44	F2	11	FE	64	66	F3
0x0007	1B	23	FF	EE	FF	53	88	7F
0x0008	1C	54	00	00	FF	AA	99	5E
0x0009	BB	34	00	00	34	55	98	4E
0x000A	B4	45	00	00	55	44	45	1E
0x000B	B5	78	00	00	FF	FF	56	12
0x000C	B6	67	56	00	00	CC	DE	E1
...
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF

Write Through + Write NO Allocate

Operaciones a realizar en un acceso a Cache

$0x001F =$

$0000\ 0000\ 0001\ 1111 =$
 $000000000000\ 11\ 111$

TAG = $0x000$

línea = $0x11$

byte = $0x7$

Bloque MP = $0x0003$

Escr 0x0003: **MISS**
 Lect 0x0004: **MISS**
 Escr 0x0007: **HIT**
 Lect 0x001F: **MISS**
 Escr 0x001A:

V	Etiqueta	DATOS MC								
0	1	0x000	00	03	05	FF	F3	87	AD	00
1	1	0x000	45	67	89	87	67	90	99	34
2	0									
3	0									

Bloque MP	DATOS MP							
0x0000	00	03	05	FF	F3	87	AD	00
0x0001	45	67	89	87	67	90	99	34
0x0002	80	00	23	80	DE	ED	90	54
0x0003	99	AA	DA	80	EF	ED	12	67
0x0004	90	78	AE	10	FE	4E	44	23
0x0005	71	32	F5	10	FE	4E	55	23
0x0006	1A	44	F2	11	FE	64	66	F3
0x0007	1B	23	FF	EE	FF	53	88	7F
0x0008	1C	54	00	00	FF	AA	99	5E
0x0009	BB	34	00	00	34	55	98	4E
0x000A	B4	45	00	00	55	44	45	1E
0x000B	B5	78	00	00	FF	FF	56	12
0x000C	B6	67	56	00	00	CC	DE	E1
...
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF

Write Through + Write NO Allocate



Operaciones a realizar en un acceso a Cache

$0x001F =$

$0000\ 0000\ 0001\ 1111 =$
 $000000000000\ 11\ 111$

TAG = $0x000$

línea = $0x11$

byte = $0x7$

Bloque MP = $0x0003$

Escr 0x0003: **MISS**
 Lect 0x0004: **MISS**
 Escr 0x0007: **HIT**
 Lect 0x001F: **MISS**
 Escr 0x001A:

V	Etiqueta	DATOS MC									
0	1	0x000	00	03	05	FF	F3	87	AD	00	
1	1	0x000	45	67	89	87	67	90	99	34	
2	0										
3	1	0x000	99	AA	DA	80	EF	ED	12	67	

Bloque MP	DATOS MP									
0x0000	00	03	05	FF	F3	87	AD	00		
0x0001	45	67	89	87	67	90	99	34		
0x0002	80	00	23	80	DE	ED	90	54		
0x0003	99	AA	DA	80	EF	ED	12	67		
0x0004	90	78	AE	10	FE	4E	44	23		
0x0005	71	32	F5	10	FE	4E	55	23		
0x0006	1A	44	F2	11	FE	64	66	F3		
0x0007	1B	23	FF	EE	FF	53	88	7F		
0x0008	1C	54	00	00	FF	AA	99	5E		
0x0009	BB	34	00	00	34	55	98	4E		
0x000A	B4	45	00	00	55	44	45	1E		
0x000B	B5	78	00	00	FF	FF	56	12		
0x000C	B6	67	56	00	00	CC	DE	E1		
		
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF		

Write Through + Write NO Allocate



Operaciones a realizar en un acceso a Cache

0x001A =

0000 0000 0001 1010 =
000000000000 11 010

TAG = 0x000

línea = 0x11

byte = 0x2

Bloque MP = 0x0003

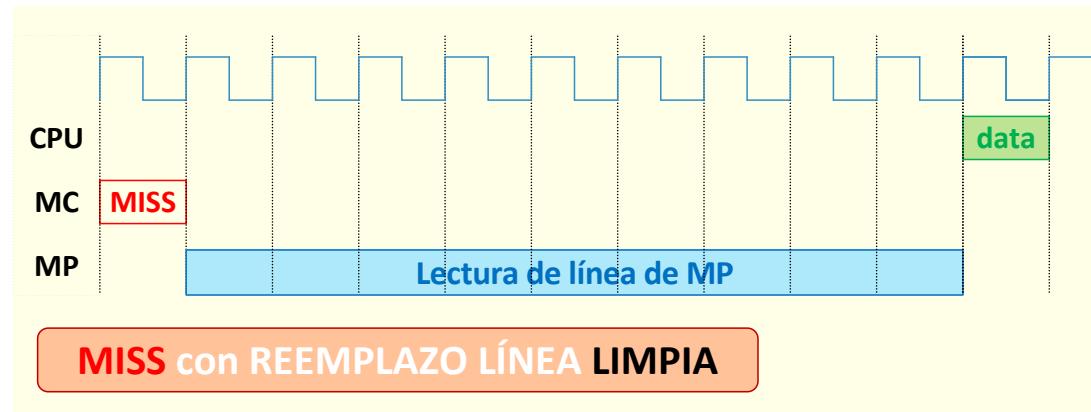
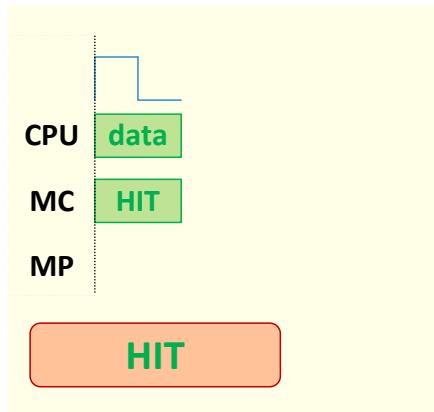
Escr 0x0003: **MISS**
 Lect 0x0004: **MISS**
 Escr 0x0007: **HIT**
 Lect 0x001F: **MISS**
 Escr 0x001A: **HIT**

	V	Etiqueta	DATOS MC							
0	1	0x000	00	03	05	FF	F3	87	AD	00
1	1	0x000	45	67	89	87	67	90	99	34
2	0									
3	1	0x000	99	AA	11	80	EF	ED	12	67

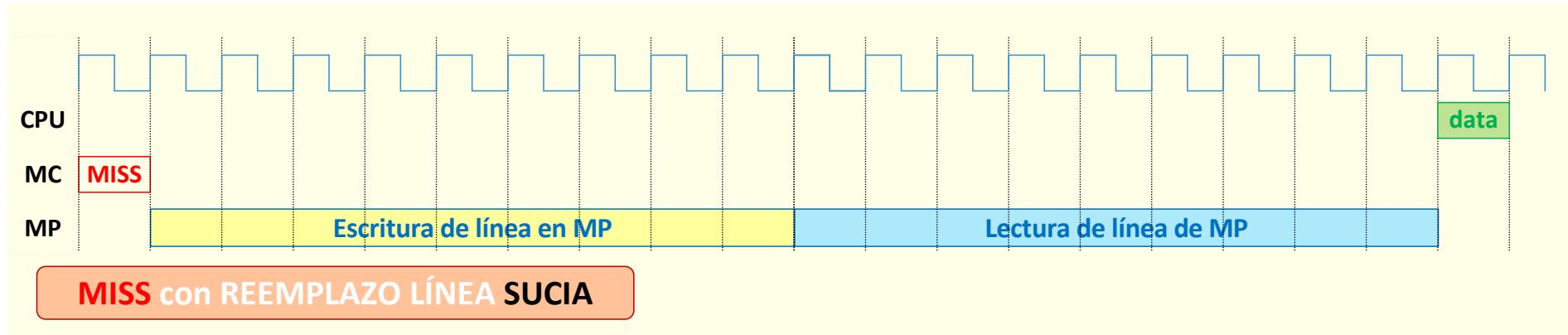
Bloque MP	DATOS MP								
0x0000	00	03	05	FF	F3	87	AD	00	
0x0001	45	67	89	87	67	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	11	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Write Through + Write NO Allocate

Operaciones a realizar en un acceso a Cache



Copy Back +
Write Allocate



Operaciones a realizar en un acceso a Cache

$0x000D =$

$0000\ 0000\ 0000\ 1100 =$
 $000000000000\ 01\ 100$

TAG = $0x000$

línea = $0x01$

byte = $0x4$

Bloque MP = $0x0001$

Escr 0x000D: **HIT**

Lect 0x000A:

Escr 0x002E:

Lect 0x0037:

	V	DB	Etiqueta	DATOS MC							
0	1	0	0x000	00	03	05	FF	F3	87	AD	00
1	1	1	0x000	45	67	89	87	FF	90	99	34
2	0	0									
3	1	0	0x000	99	AA	11	80	EF	ED	12	67

Bloque MP	DATOS MP								
0x0000	00	03	05	FF	F3	87	AD	00	
0x0001	45	67	89	87	67	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	11	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Copy Back + Write Allocate

Operaciones a realizar en un acceso a Cache

0x000A =

0000 0000 0000 1010 =
000000000000 01 010

TAG = 0x000

línea = 0x01

byte = 0x2

Bloque MP = 0x0001

Escr 0x000D: **HIT**

Lect 0x000A: **HIT**

Escr 0x002E:

Lect 0x0037:

	V	DB	Etiqueta	DATOS MC							
0	1	0	0x000	00	03	05	FF	F3	87	AD	00
1	1	1	0x000	45	67	89	87	FF	90	99	34
2	0	0									
3	1	0	0x000	99	AA	11	80	EF	ED	12	67

Bloque MP	DATOS MP								
0x0000	00	03	05	FF	F3	87	AD	00	
0x0001	45	67	89	87	67	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	11	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Copy Back + Write Allocate



Operaciones a realizar en un acceso a Cache

$0x002E =$

$0000\ 0000\ 0010\ 1110 =$
 $00000000001\ 01\ 110$

TAG = $0x001$

línea = $0x01$

byte = $0x6$

Bloque MP = $0x0005$

Escr 0x000D: **HIT**
 Lect 0x000A: **HIT**
 Escr 0x002E: **MISS**
 Lect 0x0037:

			DATOS MC								
V	DB	Etiqueta	00	03	05	FF	F3	87	AD	00	
0	1	0	0x000	00	03	05	FF	F3	87	AD	00
1	1	0	0x000	45	67	89	87	FF	90	99	34
2	0	0									
3	1	0	0x000	99	AA	11	80	EF	ED	12	67

Bloque MP	DATOS MP								
0x0000	00	03	05	FF	F3	87	AD	00	
0x0001	45	67	89	87	67	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	11	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Copy Back + Write Allocate

Operaciones a realizar en un acceso a Cache

0x002E =

0000 0000 0010 1110 =
00000000001 01 110

TAG = 0x001

línea = 0x01

byte = 0x6

Bloque MP = 0x0005

Escr 0x000D: **HIT**
 Lect 0x000A: **HIT**
 Escr 0x002E: **MISS**
 Lect 0x0037:

	V	DB	Etiqueta	DATOS MC							
0	1	0	0x000	00	03	05	FF	F3	87	AD	00
1	1	1	0x000	45	67	89	87	FF	90	99	34
2	0	0									
3	1	0	0x000	99	AA	11	80	EF	ED	12	67

Bloque MP	DATOS MP								
	00	03	05	FF	F3	87	AD	00	
0x0000	00	03	05	FF	F3	87	AD	00	
0x0001	45	67	89	87	FF	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	11	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Copy Back + Write Allocate

Operaciones a realizar en un acceso a Cache

0x002E =

0000 0000 0010 1110 =
00000000001 01 110

TAG = 0x001

línea = 0x01

byte = 0x6

Bloque MP = 0x0005

Escr 0x000D: **HIT**
 Lect 0x000A: **HIT**
 Escr 0x002E: **MISS**
 Lect 0x0037:

	V	DB	Etiqueta	DATOS MC							
0	1	0	0x000	00	03	05	FF	F3	87	AD	00
1	1	1	0x001	71	32	F5	10	FE	4E	55	23
2	0	0									
3	1	0	0x000	99	AA	11	80	EF	ED	12	67

Bloque MP	DATOS MP								
	00	03	05	FF	F3	87	AD	00	
0x0000	00	03	05	FF	F3	87	AD	00	
0x0001	45	67	89	87	FF	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	11	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Copy Back + Write Allocate

Operaciones a realizar en un acceso a Cache

0x0037 =

0000 0000 0011 0111 =
0000000001 10 111

TAG = 0x001

línea = 0x02

byte = 0x7

Bloque MP = 0x0006

Escr 0x000D: **HIT**
 Lect 0x000A: **HIT**
 Escr 0x002E: **MISS**
 Lect 0x0037: **MISS**

	V	DB	Etiqueta	DATOS MC							
0	1	0	0x000	00	03	05	FF	F3	87	AD	00
1	1	1	0x001	71	32	F5	10	FE	4E	55	23
2	0	0									
3	1	0	0x000	99	AA	11	80	EF	ED	12	67

Bloque MP	DATOS MP								
0x0000	00	03	05	FF	F3	87	AD	00	
0x0001	45	67	89	87	FF	90	99	34	
0x0002	80	00	23	80	DE	ED	90	54	
0x0003	99	AA	11	80	EF	ED	12	67	
0x0004	90	78	AE	10	FE	4E	44	23	
0x0005	71	32	F5	10	FE	4E	55	23	
0x0006	1A	44	F2	11	FE	64	66	F3	
0x0007	1B	23	FF	EE	FF	53	88	7F	
0x0008	1C	54	00	00	FF	AA	99	5E	
0x0009	BB	34	00	00	34	55	98	4E	
0x000A	B4	45	00	00	55	44	45	1E	
0x000B	B5	78	00	00	FF	FF	56	12	
0x000C	B6	67	56	00	00	CC	DE	E1	
	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Copy Back + Write Allocate

Operaciones a realizar en un acceso a Cache

0x0037 =

0000 0000 0011 0111 =
0000000001 10 111

TAG = 0x001

línea = 0x02

byte = 0x7

Bloque MP = 0x0006

Escr 0x000D: **HIT**
 Lect 0x000A: **HIT**
 Escr 0x002E: **MISS**
 Lect 0x0037: **MISS**

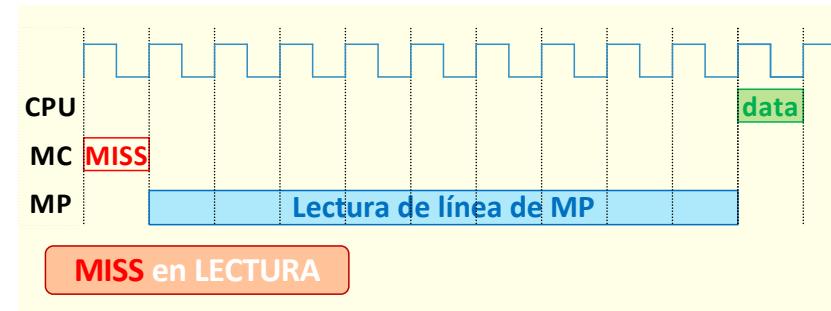
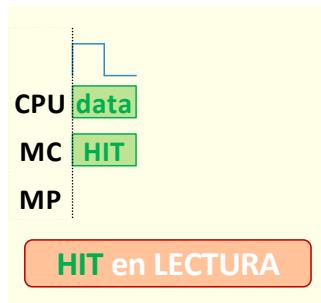
	V	DB	Etiqueta	DATOS MC							
0	1	0	0x000	00	03	05	FF	F3	87	AD	00
1	1	1	0x001	71	32	F5	10	FE	4E	55	23
2	1	0	0x001	1A	44	F2	11	FE	64	66	F3
3	1	0	0x000	99	AA	11	80	EF	ED	12	67

Bloque MP	DATOS MP								
	00	03	05	FF	F3	87	AD	00	
0x0000	45	67	89	87	FF	90	99	34	
0x0001	80	00	23	80	DE	ED	90	54	
0x0002	99	AA	11	80	EF	ED	12	67	
0x0003	90	78	AE	10	FE	4E	44	23	
0x0004	71	32	F5	10	FE	4E	55	23	
0x0005	1A	44	F2	11	FE	64	66	F3	
0x0006	1B	23	FF	EE	FF	53	88	7F	
0x0007	1C	54	00	00	FF	AA	99	5E	
0x0008	BB	34	00	00	34	55	98	4E	
0x0009	B4	45	00	00	55	44	45	1E	
0x000A	B5	78	00	00	FF	FF	56	12	
0x000B	B6	67	56	00	00	CC	DE	E1	
0x000C	
...	FF	FF	FF	FF	FF	FF	FF	FF	
0x1FFF	FF	FF	FF	FF	FF	FF	FF	FF	

Copy Back + Write Allocate

Evaluación

Tiempo de acceso a la cache



$$t_{\text{acceso}} = t_h$$

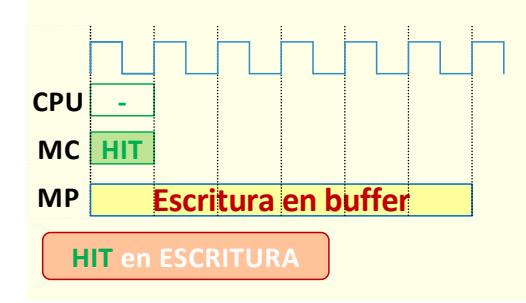
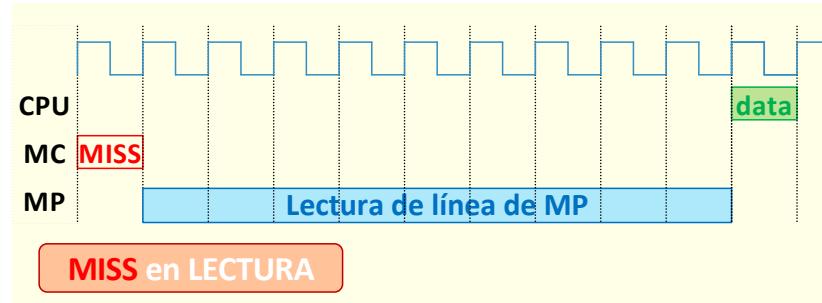
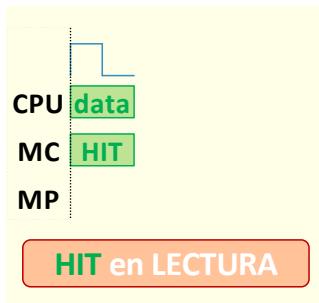
$$t_{\text{acceso}} = t_h + t_p$$

$$t_p = t_{\text{bloq}} + t_h$$

Cache de sólo Lectura

Evaluación

Tiempo de acceso a la cache



$$t_{\text{acceso}} = t_h$$

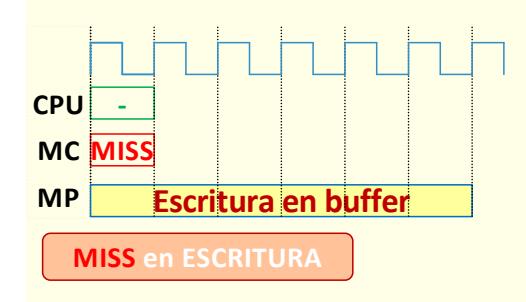
$$t_{\text{acceso}} = t_h + t_p$$

$$t_p = t_{\text{bloq}} + t_h$$

Write Through +
Write NO Allocate

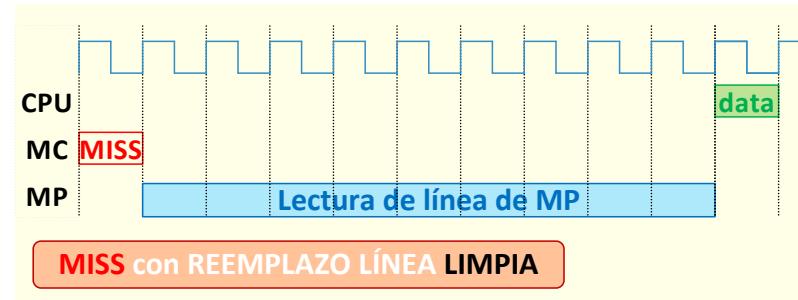
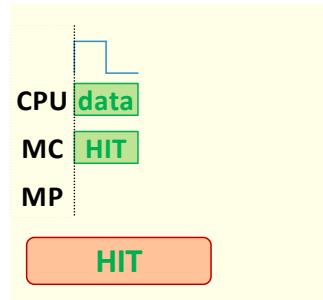
$$t_{\text{acceso}} = t_h$$

Supondremos que todas las escrituras en MP se harán sobre un buffer a coste 0 y sin interferencias



Evaluación

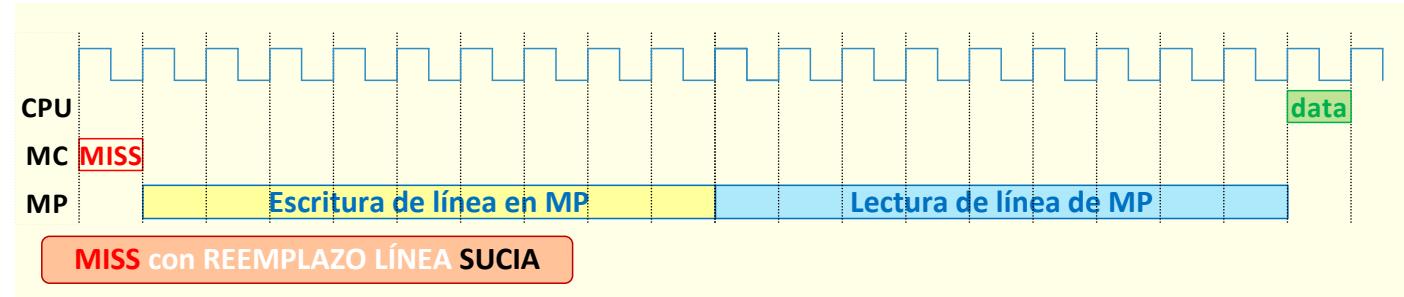
Tiempo de acceso a la cache



$$t_{\text{acceso}} = t_h$$

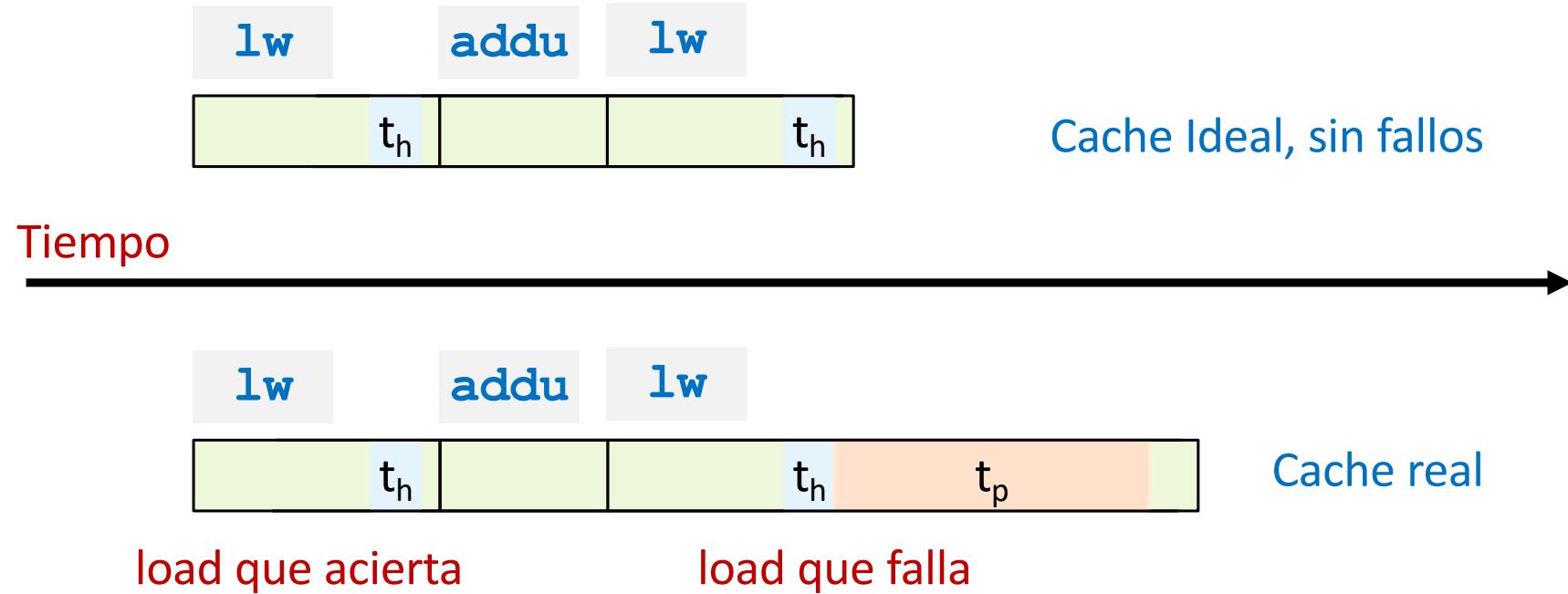
$$t_{\text{acceso}} = t_h + t_p \quad t_p = t_{\text{bloq}} + t_h$$

Copy Back +
Write Allocate



$$t_{\text{acceso}} = t_h + t_p \quad t_p = t_{\text{bloq}} + t_{\text{bloq}} + t_h$$

Impacto de la cache en el Rendimiento



Impacto de la cache en el Rendimiento

- El tiempo de Ejecución de un programa se modelaba así:

$$T_{\text{exe}} = N \cdot CPI \cdot T_c$$

- **N**, número de instrucciones
- **CPI**, ciclos en media por instrucción
- **T_c**, tiempo de ciclo del procesador

- Si tenemos una cache, el modelo cambia:

$$T_{\text{exe}} = N \cdot (CPI_{\text{ideal}} + CPI_{\text{mem}}) \cdot T_c = N \cdot (CPI_{\text{ideal}} + m \cdot t_p \cdot n_r) \cdot T_c$$

- **CPI_{ideal}**, ciclos en media por instrucción suponiendo una cache perfecta (sin fallos)
- **CPI_{mem}**, son los ciclos que perdemos por tener una cache imperfecta ($m \neq 0$)
 - ✓ **m**, tasa de fallos
 - ✓ **t_p**, tiempo de penalización en caso de fallo
 - ✓ **n_r**, número de referencias por instrucción

Suponemos que el fetch de instrucciones se realiza sin fallos de cache.

Problema

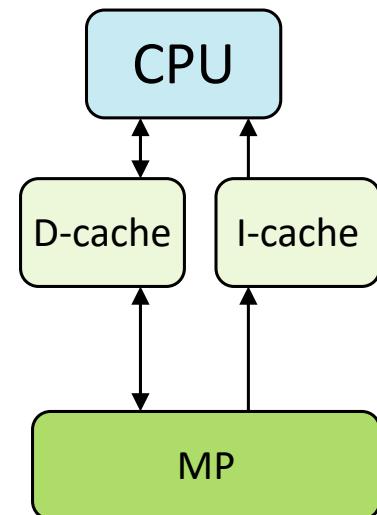
- Dado un sistema con caches separadas para datos e instrucciones:
 - Cache de Instrucciones, $m_i = 2\%$
 - Cache de Datos, $m_d = 4\%$
 - $t_p = 100$ ciclos
 - CPI ideal = 2,5 ciclos/inst
 - 36% de las instrucciones son loads o stores.

Calculad el CPI real.

$$CPI_{memI} = n_r \cdot m_i \cdot t_p = 1 \cdot 0,02 \cdot 100 = 2 \text{ ciclos}$$

$$CPI_{memD} = n_r \cdot m_d \cdot t_p = 0,36 \cdot 0,04 \cdot 100 = 1,44 \text{ ciclos}$$

$$CPI_{real} = CPI_{ideal} + CPI_{mem} = 2,5 + 2 + 1,44 = 5,94 \text{ ciclos}$$



Impacto de la cache en el Rendimiento

- Sabemos que T_{exe} depende de m y t_p . ¿Podemos mejorar el T_{exe} , mejorando estos parámetros?
 - **No es tan sencillo.** Podemos mejorar m con una cache más grande, pero probablemente eso hace que aumente en t_h y T_c .
- Necesitamos una métrica que sólo utilice los parámetros con memoria

$$t_{am} = t_h + m \cdot t_p$$

- **T_{am}**, es el tiempo de acceso medio a memoria
 - Ejemplo, supongamos una MC con
 - ✓ $T_c = 0,75\text{ns}$, $t_h = 1$ ciclo, $t_p = 20$ ciclos, $m = 5\%$
 - ✓ Calculad el tiempo medio de acceso a memoria en ciclos y en ns
$$t_{am} = 1 + 0,05 \cdot 20 = 2 \text{ ciclos}$$
$$t_{am} = 2 \cdot 0,75 = 1,5 \text{ ns}$$

Impacto de la cache en el Rendimiento

$$t_{am} = t_h + m \cdot t_p$$

$$T_{exe} = N \cdot (CPI_{ideal} + m \cdot t_p \cdot n_r) \cdot T_c$$

- ❑ No podemos mejorar sólo un componente (CPU o MC) [Ley del Amdahl]
- ❑ Hay múltiples técnicas para mejorar el rendimiento de la CPU.
- ❑ También existen técnicas para mejorar el rendimiento de la cache.
 - Si queremos mejorar la $m \Rightarrow$ **ASOCIATIVIDAD**
 - Si queremos mejorar $t_p \Rightarrow$ **CACHES MULTINIVEL**

Mejoras en el rendimiento: Asociatividad

```
for (i=0; i<200; i++)  
    sum = sum + V[i];
```

Líneas de cache de 16B

m = 25%

LOCALIDAD
ESPACIAL



- Accedemos a V[0] ⇒ **FALLO**
 - Traemos a MC Bloque de datos: V[0] – V[3] (16 bytes)
- Accedemos a V[1] ⇒ **ACIERTO**
- Accedemos a V[2] ⇒ **ACIERTO**
- Accedemos a V[3] ⇒ **ACIERTO**
- Accedemos a V[4] ⇒ **FALLO**
 - Traemos a MC Bloque de datos: V[4] – V[7] (16 bytes)
- Accedemos a V[5] ⇒ **ACIERTO**
- ...

Mejoras en el rendimiento: Asociatividad

```
for (i=0; i<200; i++)  
    sum = sum + V[i]*W[i];
```

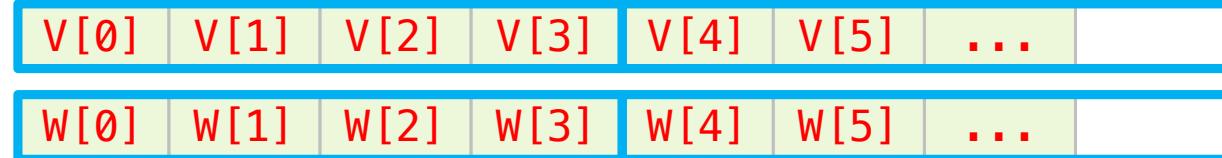
Líneas de cache de 16B

m = 100%

Fallos por CONFLICTO

SOLUCIÓN:

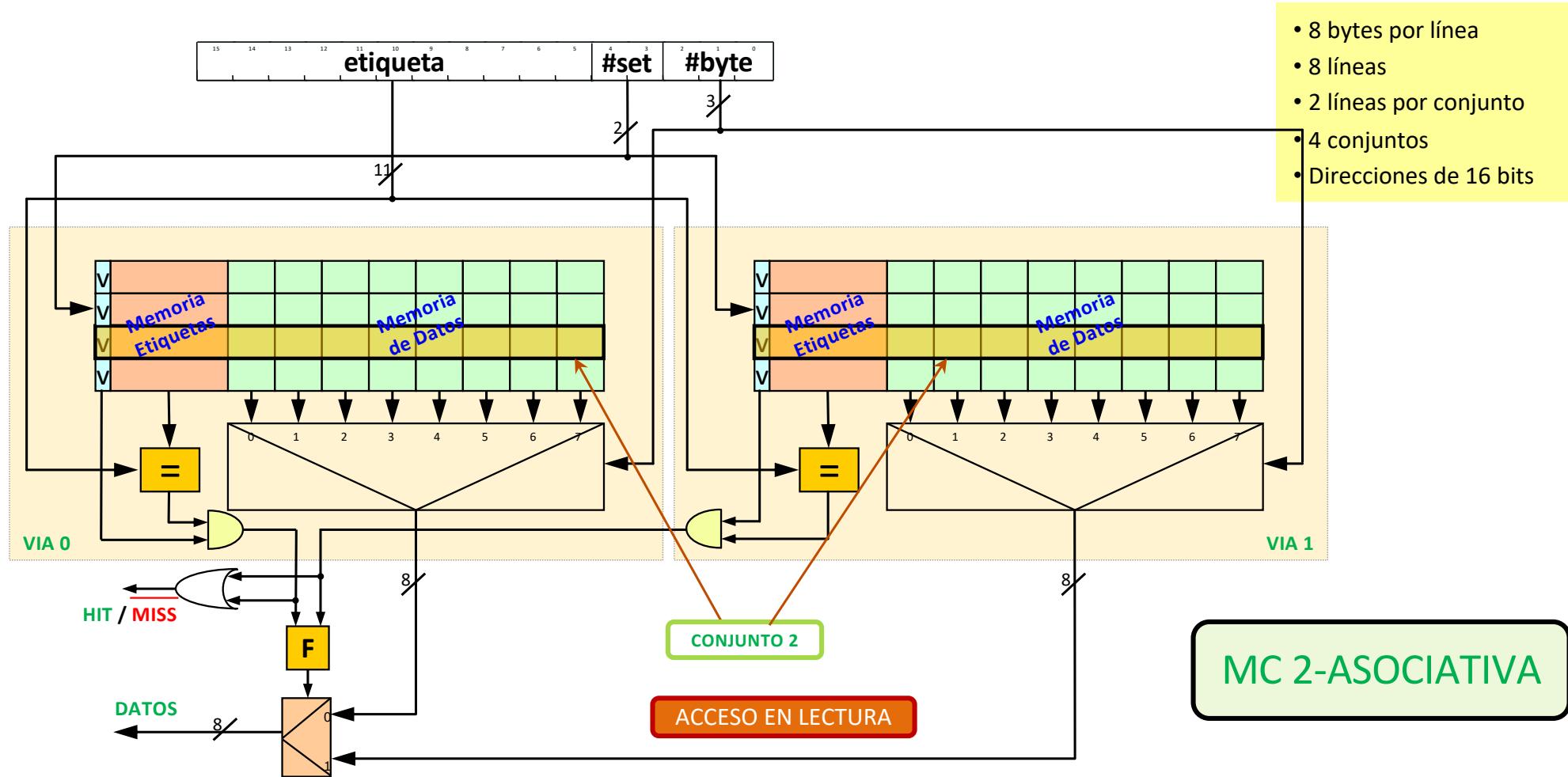
- Permitir que un bloque de MP se pueda alojar en más de 1 línea de MC.
- Aumentar la ASOCIATIVIDAD.



¿Qué pasaría si V[0] y W[0] fueran a parar a la misma línea de MC

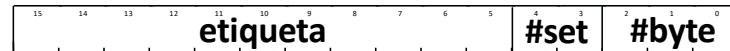
- Accedemos a V[0] ⇒ **FALLO**
 - Traemos a MC Bloque de datos: V[0] – V[3] (16 bytes)
- Accedemos a W[0] ⇒ **FALLO**
 - Traemos a MC Bloque de datos: W[0] – W[3] (16 bytes)
 - Expulsamos de la MC: V[0] – V[3]
- Accedemos a V[1] ⇒ **FALLO**
 - Traemos a MC Bloque de datos: V[0] – V[3] (16 bytes)
- Accedemos a W[1] ⇒ **FALLO**
 - Traemos a MC Bloque de datos: W[0] – W[3] (16 bytes)
 - Expulsamos de la MC: V[0] – V[3]
- ... idem, para todos los demás accesos

Memoria Cache Asociativa por Conjuntos

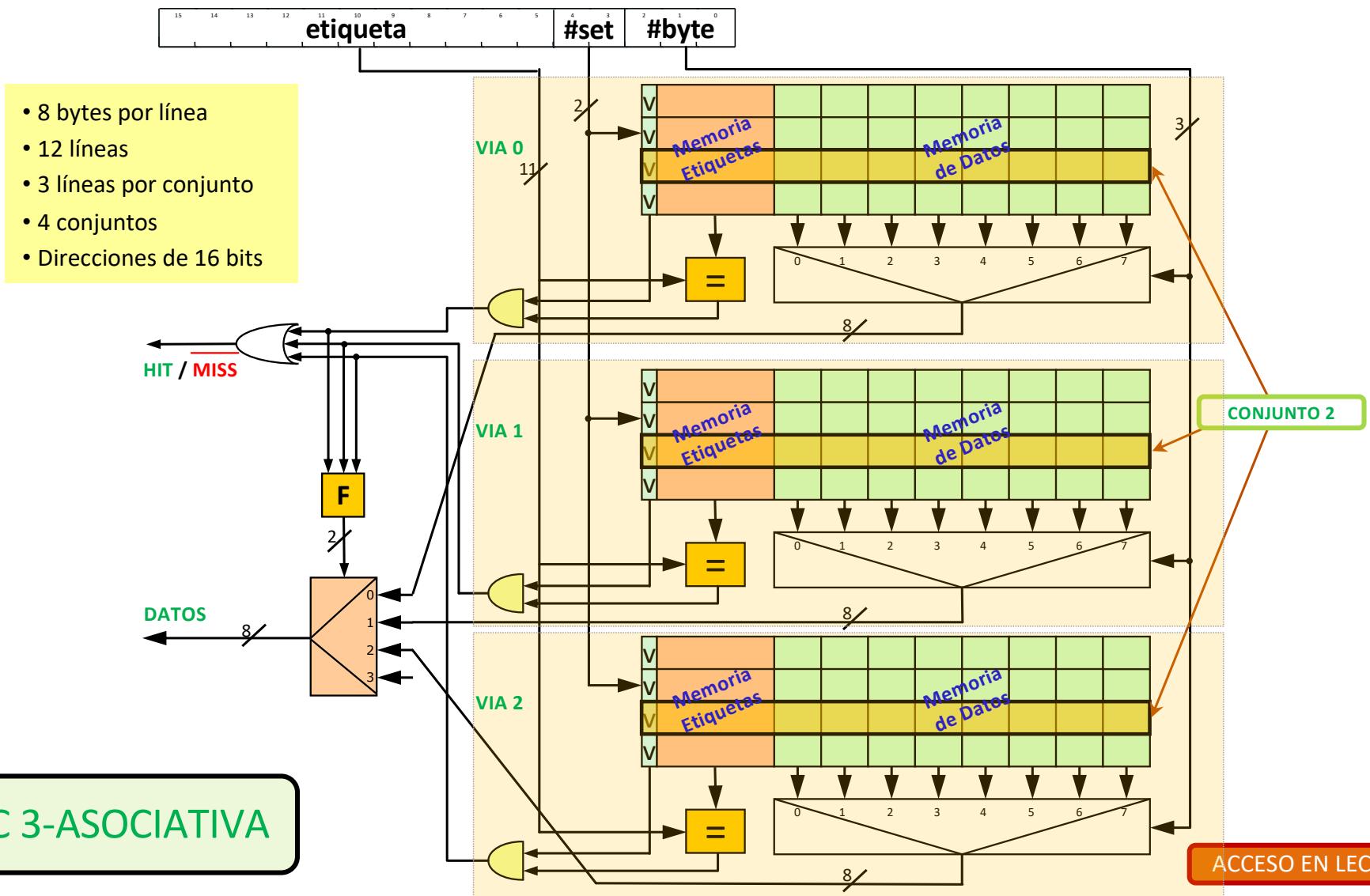


Memoria Cache Asociativa por Conjuntos

- ❑ Una Cache 2-Asociativa (2-way) está compuesta de 2 Vías
- ❑ Cada vía puede verse como una cache directa.
- ❑ La línea X de todas las vías forma el Conjunto X
- ❑ 1 Bloque de MP se guarda en 1 Conjunto de MC, pero dentro del conjunto lo puede hacer en cualquiera de las vías.



- ❑ ¿Cómo seleccionamos la vía? ⇒ **ALGORITMO de REEMPLAZO**
- ❑ El número de conjuntos siempre ha de ser potencia de 2
- ❑ El número de vías puede ser cualquiera.
 - Podemos tener caches: 3-asociativa, 7-asociativa, 12-asociativa, ...



Algoritmos de Emplazamiento

□ Memoria Cache ASOCIATIVA por CONJUNTOS (X-Asociativa, X-way)

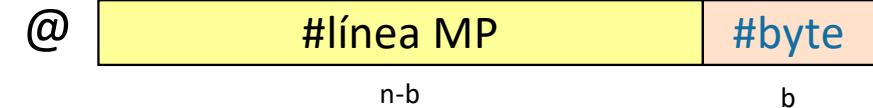
- La Cache tiene S conjuntos ($S=2^s$) y X líneas por Conjunto
- Un Bloque de MP se almacena en 1 Conjunto de MC, pero dentro del Conjunto en cualquier línea

□ Memoria Cache DIRECTA

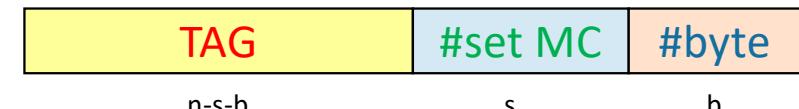
- La Cache tiene D líneas ($D=2^d$)
- Un bloque de MP se almacena en 1 línea de MC

□ Memoria Cache COMPLETAMENTE ASOCIATIVA

- La Cache tiene un solo conjunto de M líneas, M puede ser cualquier valor.
- Un Bloque de MP se almacena en cualquier línea de MC



MC Asociativa por Conjutos



MC Directa



MC Completamente Asociativa



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.
 - Los datos experimentales dicen los algoritmos de reemplazo no influyen sustancialmente en el rendimiento de la MC
- ❑ 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.

Algoritmos de Reemplazo

- ❑ Reemplazo **Aleatorio**
- ❑ Reemplazo **FIFO (First In First Out)**
- ❑ Reemplazo **LRU (Least Recently Used)**
 - De entre todas las líneas candidatas a ser reemplazadas, se selecciona la que lleva más tiempo en la cache sin ser utilizada.
 - 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 sólo se mantiene parte de esa información.

Ejemplo de comportamiento

Características de la Memoria Cache

- ❑ Asociativa por Conjuntos
- ❑ Tamaño de Cache: 128 bytes
- ❑ Tamaño de línea: 16 bytes
- ❑ Número de Conjuntos: 4
- ❑ Líneas por Conjunto: 2
- ❑ Algoritmo de reemplazo: LRU
- ❑ Política de escritura: WT + WnA
- ❑ Direcciones de 16 bits



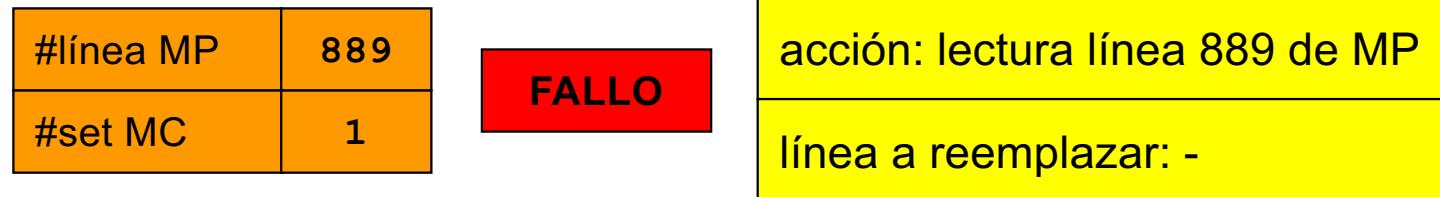
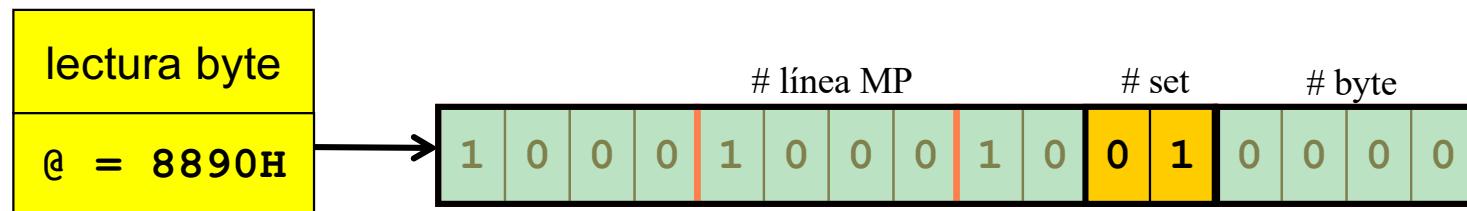
Contenido inicial de la Memoria Cache

set 0	set 1	set 2	set 3
EC8 1	EC5 1	EC6 0	EC7 1
AB4 0	- 0	AB2 1	- 0

Por simplificación el contenido de la cache es el número de línea, en vez de la etiqueta.

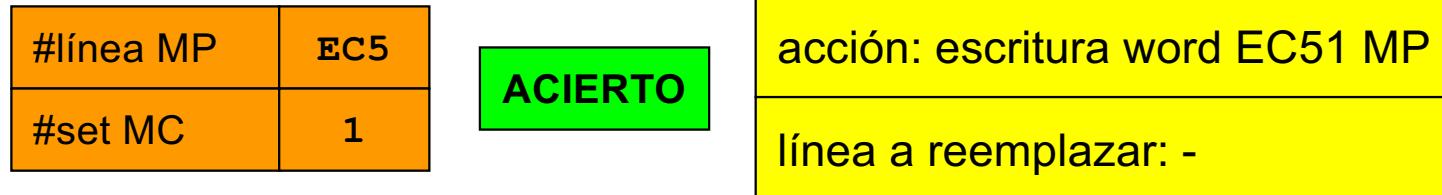
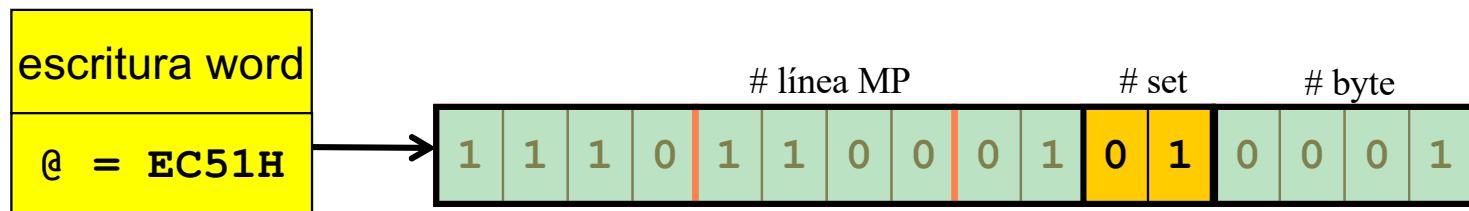
El bit a 1, indica la última línea referenciada en ese conjunto

	set 0	set 1	set 2	set 3			
EC8	1	EC5	0	EC6	0	EC7	1
AB4	0	889	1	AB2	1	-	0



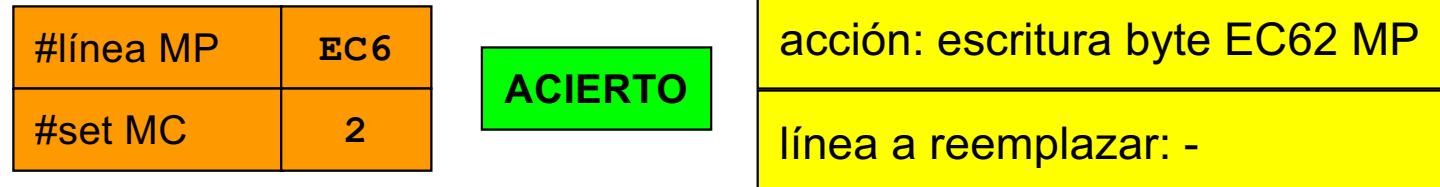
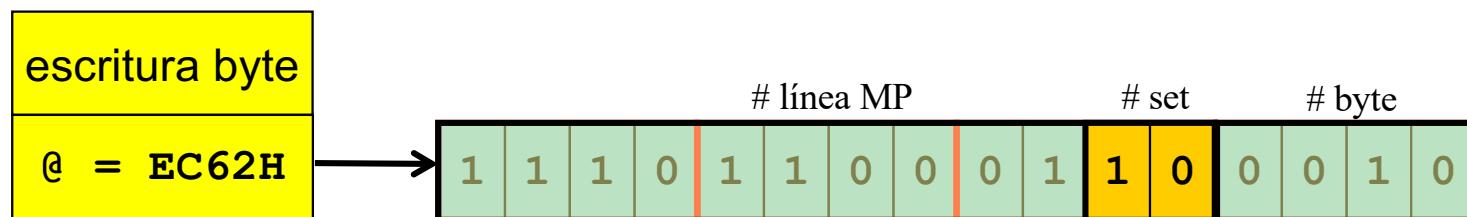
# accesos	1	tasa fallos	1.0	#bytes leídos MP	16
# fallos	1	tasa aciertos	0.0	#bytes escritos MP	0

	set 0	set 1	set 2	set 3
EC8	1	EC5	1	EC6
AB4	0	889	0	AB2



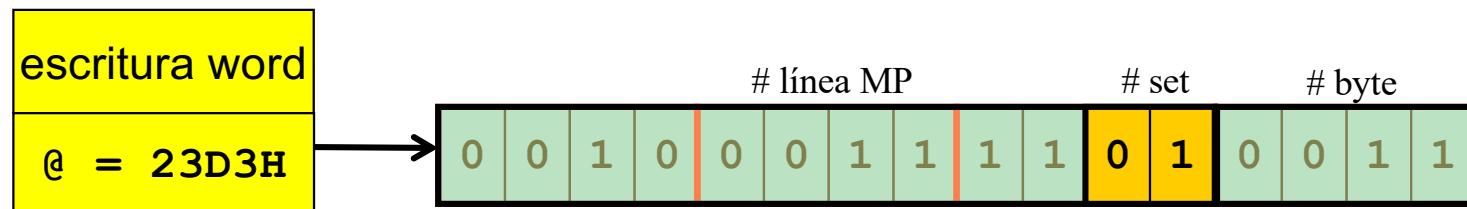
# accesos	2	tasa fallos	0.5	#bytes leídos MP	16
# fallos	1	tasa aciertos	0.5	#bytes escritos MP	2

	set 0	set 1	set 2	set 3	
EC8	1	EC5	1	EC6	1
AB4	0	889	0	AB2	0



# accesos	3	tasa fallos	0.33	#bytes leídos MP	16
# fallos	1	tasa aciertos	0.67	#bytes escritos MP	3

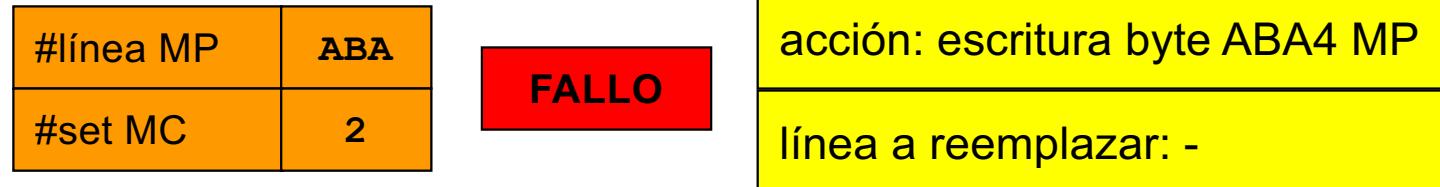
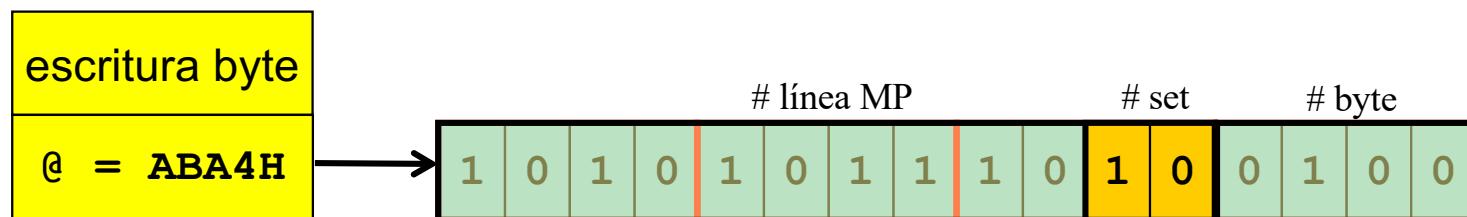
	set 0	set 1	set 2	set 3
EC8	1	EC5	1	EC6
AB4	0	889	0	AB2



#línea MP	23D	FALLO	acción: escritura word 23D3 MP
#set MC	1		línea a reemplazar: -

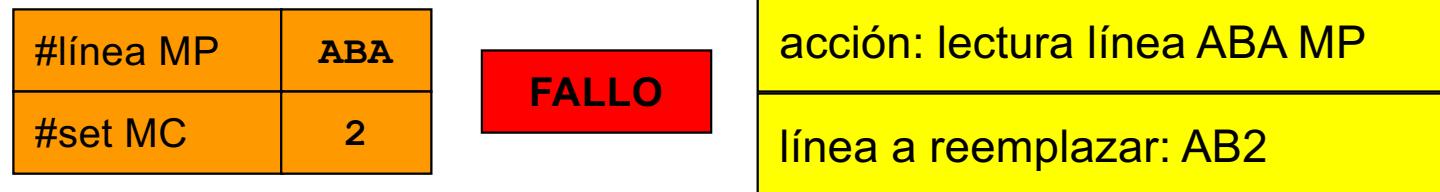
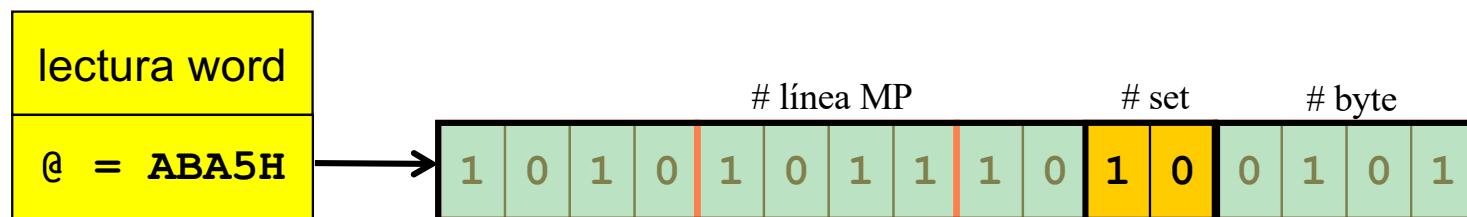
# accesos	4	tasa fallos	0.5	#bytes leídos MP	16
# fallos	2	tasa aciertos	0.5	#bytes escritos MP	5

	set 0	set 1	set 2	set 3	
EC8	1	EC5	1	EC6	1
AB4	0	889	0	AB2	0



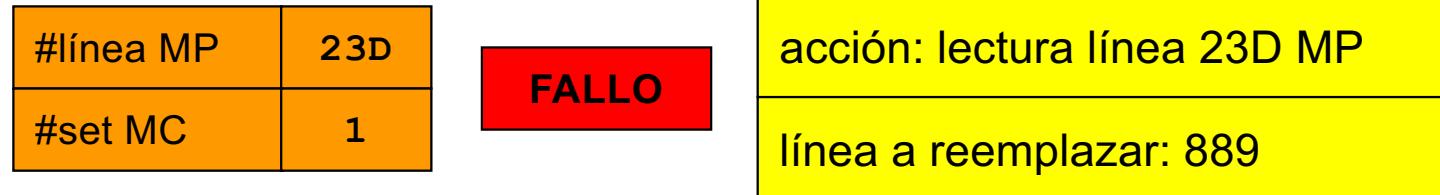
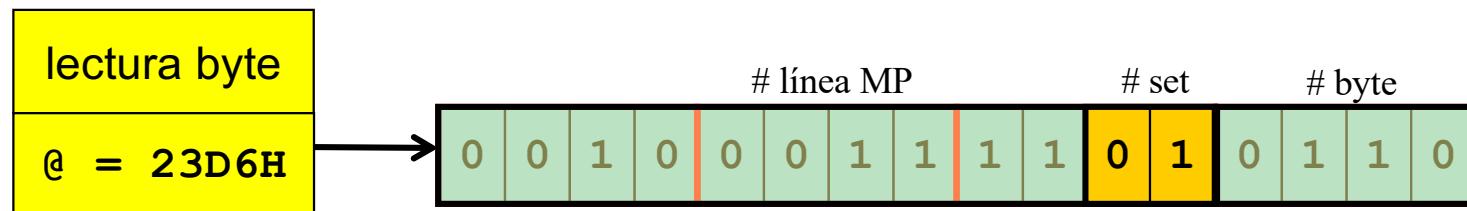
# accesos	5	tasa fallos	0.6	#bytes leídos MP	16
# fallos	3	tasa aciertos	0.4	#bytes escritos MP	6

	set 0	set 1	set 2	set 3	
EC8	1	EC5	1	EC6	0
AB4	0	889	0	ABA	1



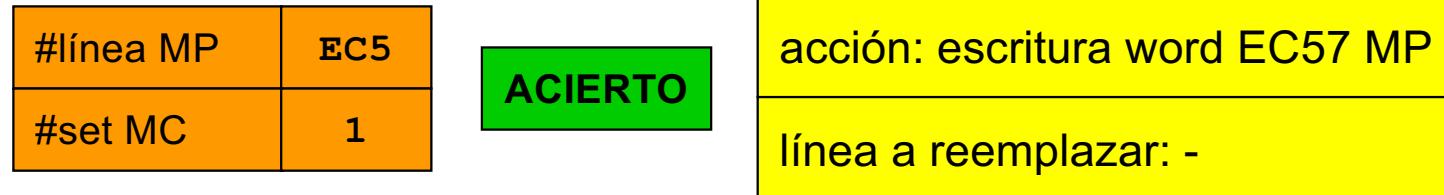
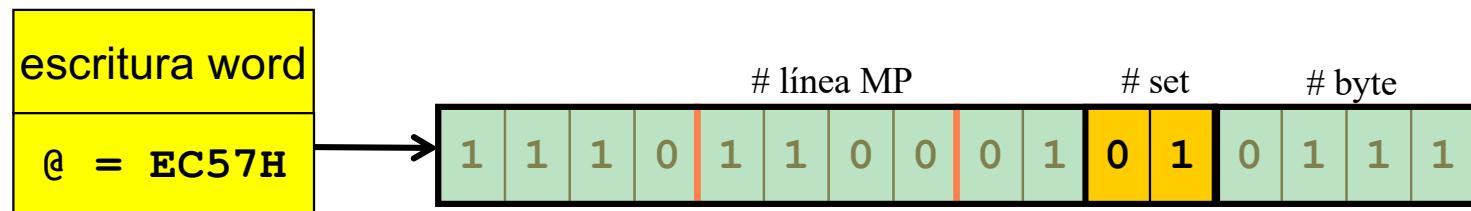
# accesos	6	tasa fallos	0.66	#bytes leídos MP	32
# fallos	4	tasa aciertos	0.34	#bytes escritos MP	6

	set 0	set 1	set 2	set 3			
EC8	1	EC5	0	EC6	0	EC7	1
AB4	0	23D	1	ABA	1	-	0



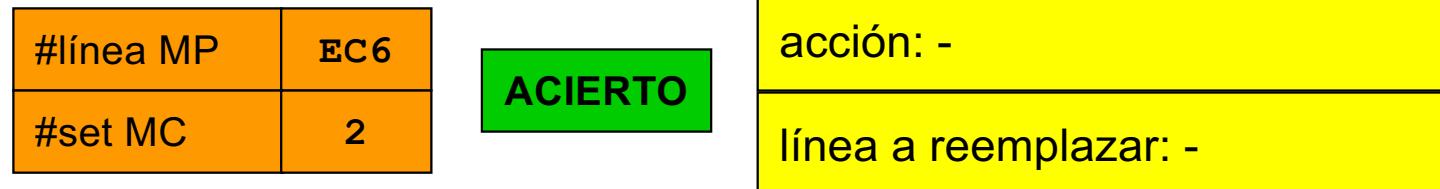
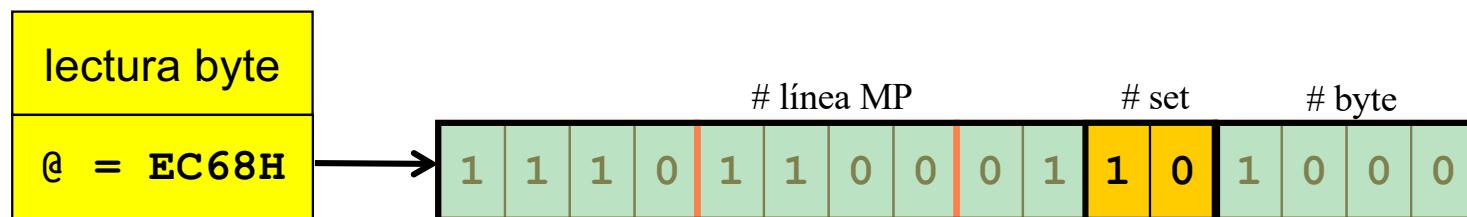
# accesos	7	tasa fallos	0.71	#bytes leídos MP	48
# fallos	5	tasa aciertos	0.29	#bytes escritos MP	6

	set 0	set 1	set 2	set 3
EC8	1	EC5	1	EC6
AB4	0	23D	0	ABA



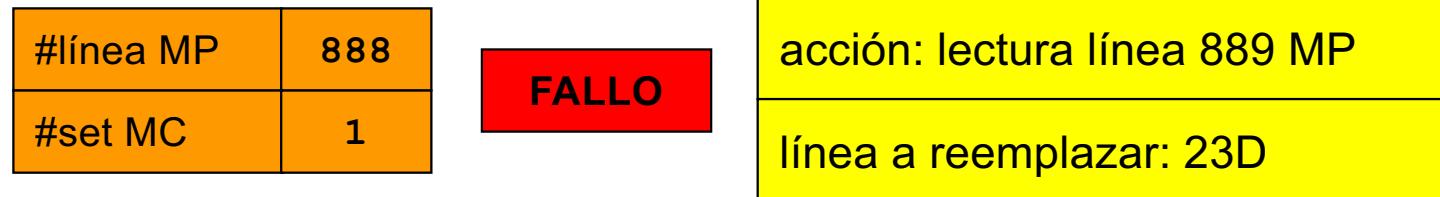
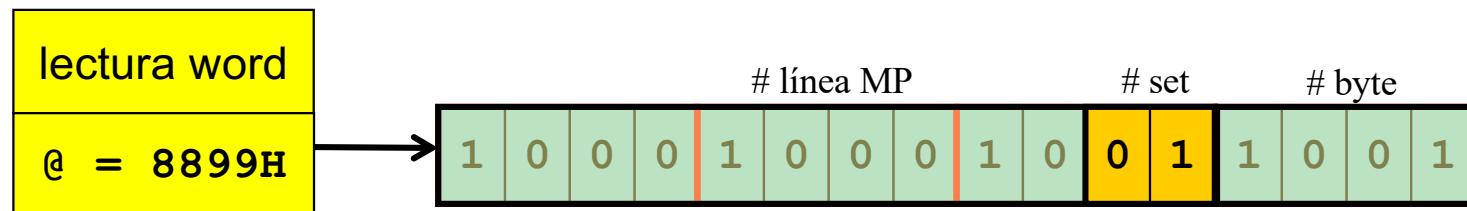
# accesos	8	tasa fallos	0.62	#bytes leídos MP	48
# fallos	5	tasa aciertos	0.38	#bytes escritos MP	8

	set 0	set 1	set 2	set 3	
EC8	1	EC5	1	EC6	1
AB4	0	23D	0	ABA	0



# accesos	9	tasa fallos	0.55	#bytes leídos MP	48
# fallos	5	tasa aciertos	0.45	#bytes escritos MP	8

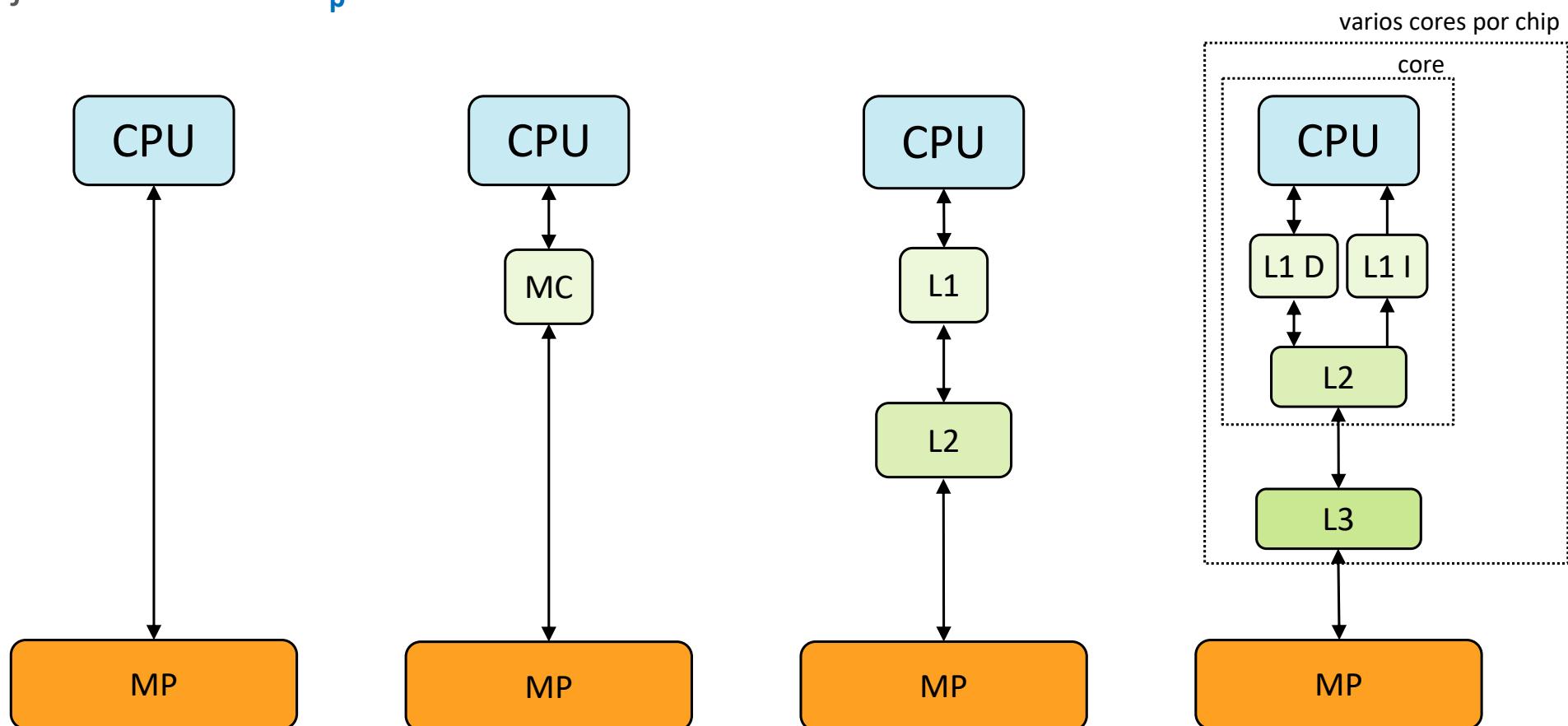
	set 0	set 1	set 2	set 3			
EC8	1	EC5	0	EC6	1	EC7	1
AB4	0	889	1	ABA	0	-	0



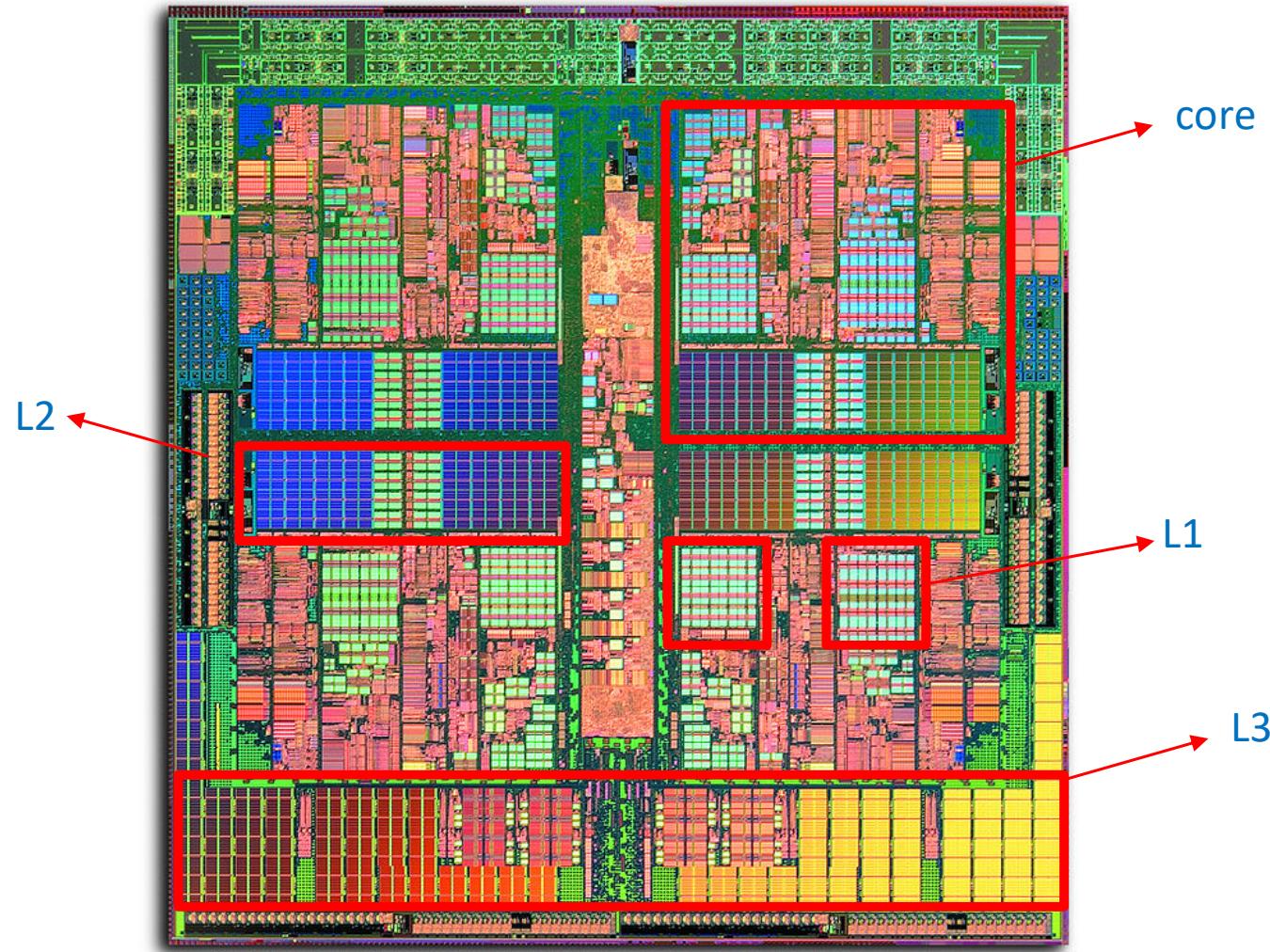
# accesos	10	tasa fallos	0.6	#bytes leídos MP	64
# fallos	6	tasa aciertos	0.4	#bytes escritos MP	8

Caches multinivel

❑ Objetivo: **REDUCIR t_p**



QuadCore AMD Opteron



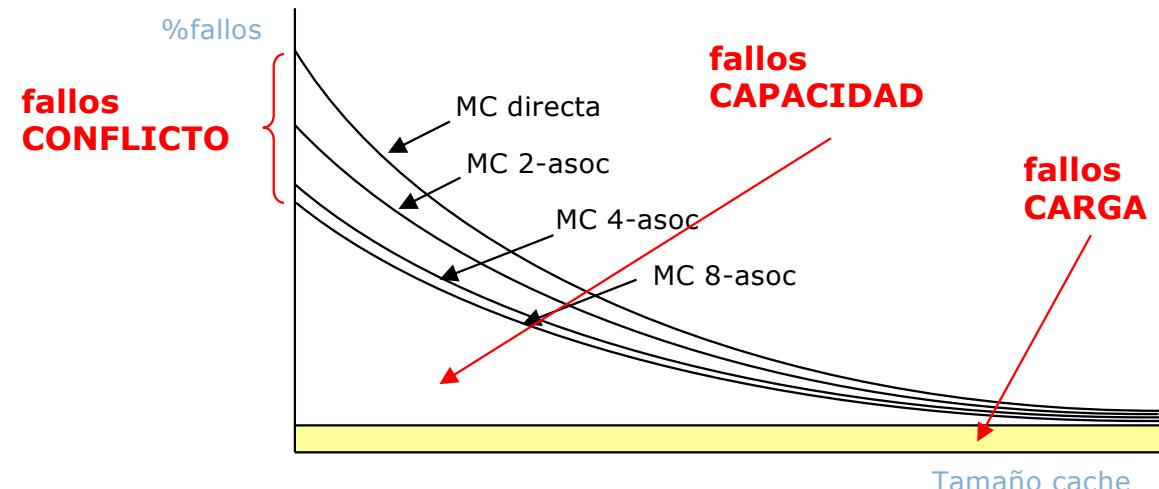
Tipos de Fallos

Los fallos de cache pueden dividirse en tres categorías:

- ❑ **CARGA** (compulsory). Se producen la primera vez que se accede a una posición de memoria.
- ❑ **CONFLICTO**. Se producen cuando varias líneas se mapean en el mismo lugar de la MC (sólo aplica en MC directas y asociativas por conjuntos, en una MC Completamente Asociativa no puede haber fallos por conflicto, por definición).
- ❑ **CAPACIDAD**. Todas las líneas que necesita un programa no caben en la Memoria Cache.

¿Cómo se calculan?

Ayuda a entenderlo



Ejemplo de tipos de fallos

```
int A[8], B[8];
int i, s=0;
for (i=0; i<8; i++)
    s = s + A[i] + B[i];
```

- Cache Directa
- Líneas de 8 bytes (2 words)
- 4 Líneas
- Procesador 32 bits

MC

V	Etiquetas	Datos	
0	0		
1	0		
2	0		
3	0		

MP	#bloque
A[0]	0
A[1]	1
A[2]	
A[3]	
A[4]	
A[5]	
A[6]	
A[7]	
B[0]	4
B[1]	5
B[2]	
B[3]	
B[4]	
B[5]	
B[6]	
B[7]	

Ejemplo de tipos de fallos

```
int A[8], B[8];
int i, s=0;
for (i=0; i<8; i++)
    s = s + A[i] + B[i];
```

- Cache Directa
- Líneas de 8 bytes (2 words)
- 4 Líneas
- Procesador 32 bits

MC

V	Etiquetas	Datos	
0	1	A[0]	A[1]
1	0		
2	0		
3	0		

MISS CARGA → A:

MP	#bloque
A[0]	0
A[1]	1
A[2]	2
A[3]	3
A[4]	4
A[5]	5
A[6]	6
A[7]	7
B[0]	4
B[1]	5
B[2]	6
B[3]	7
B[4]	4
B[5]	5
B[6]	6
B[7]	7

B:

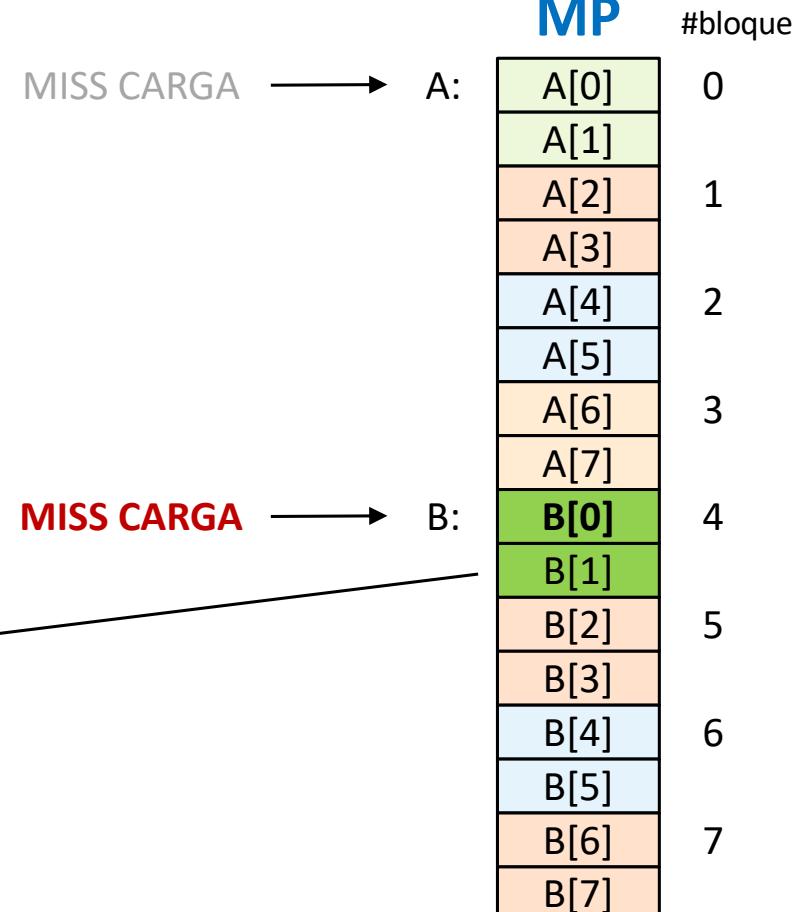
Ejemplo de tipos de fallos

```
int A[8], B[8];
int i, s=0;
for (i=0; i<8; i++)
    s = s + A[i] + B[i]; ;
```

- Cache Directa
- Líneas de 8 bytes (2 words)
- 4 Líneas
- Procesador 32 bits

MC

V	Etiquetas	Datos	
0	1	B[0]	B[1]
1	0		
2	0		
3	0		

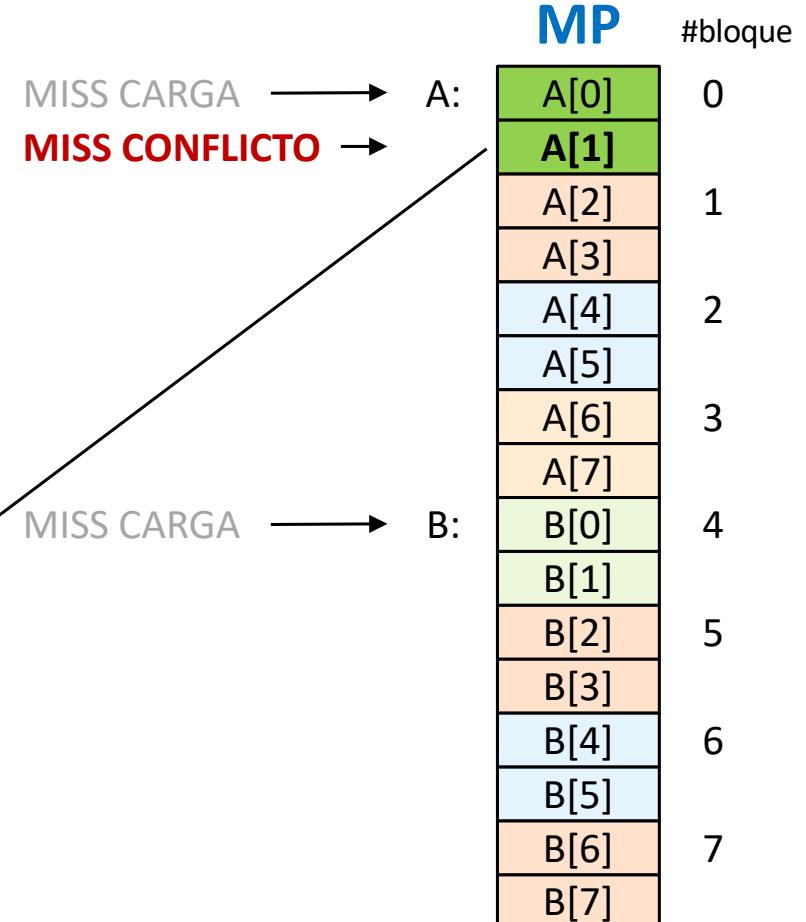


Ejemplo de tipos de fallos

```
int A[8], B[8];
int i, s=0;
for (i=0; i<8; i++)
    s = s + A[i] + B[i]; ;
```

- MC
- Cache Directa
 - Líneas de 8 bytes (2 words)
 - 4 Líneas
 - Procesador 32 bits

V	Etiquetas	Datos	
0	1	A[0]	A[1]
1	0		
2	0		
3	0		



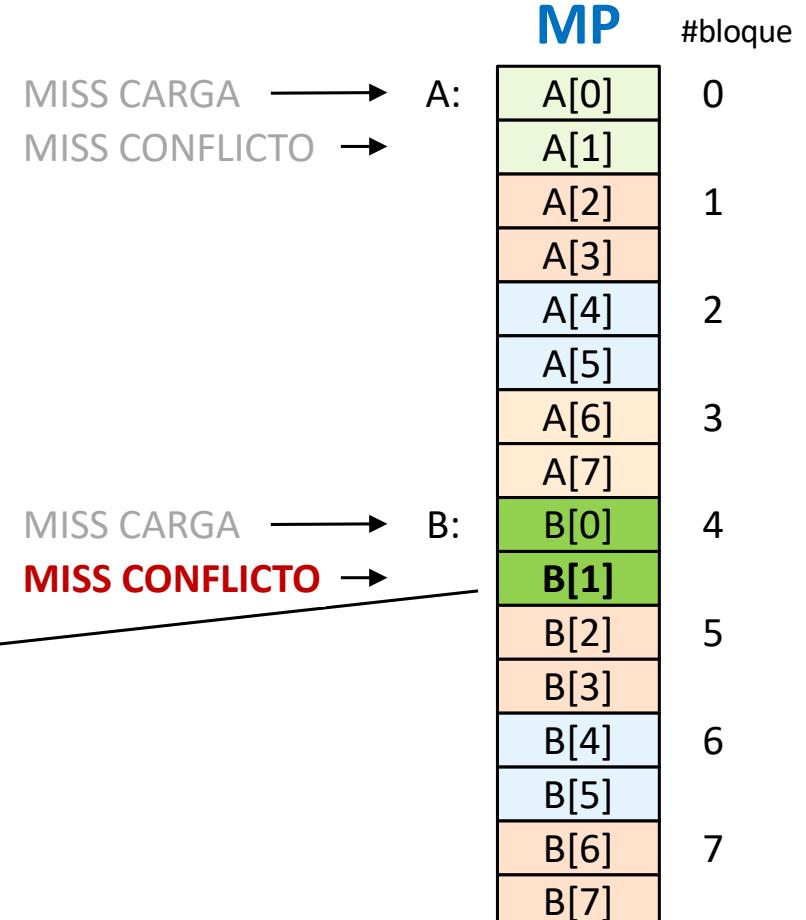
Ejemplo de tipos de fallos

```
int A[8], B[8];
int i, s=0;
for (i=0; i<8; i++)
    s = s + A[i] + B[i]; ;
```

- Cache Directa
- Líneas de 8 bytes (2 words)
- 4 Líneas
- Procesador 32 bits

MC

V	Etiquetas	Datos	
0	1	B[0]	B[1]
1	0		
2	0		
3	0		



Ejemplo de tipos de fallos

```
int V[16], i, s=0;  
for (i=0; i<16; i++)  
    s = s + V[i];  
for (i=0; i<16; i++)  
    s = s + V[i]*3;
```

- Cache Completamente Asociativa
- 4 Líneas de 8 bytes (2 words)
- Procesador 32 bits

MC

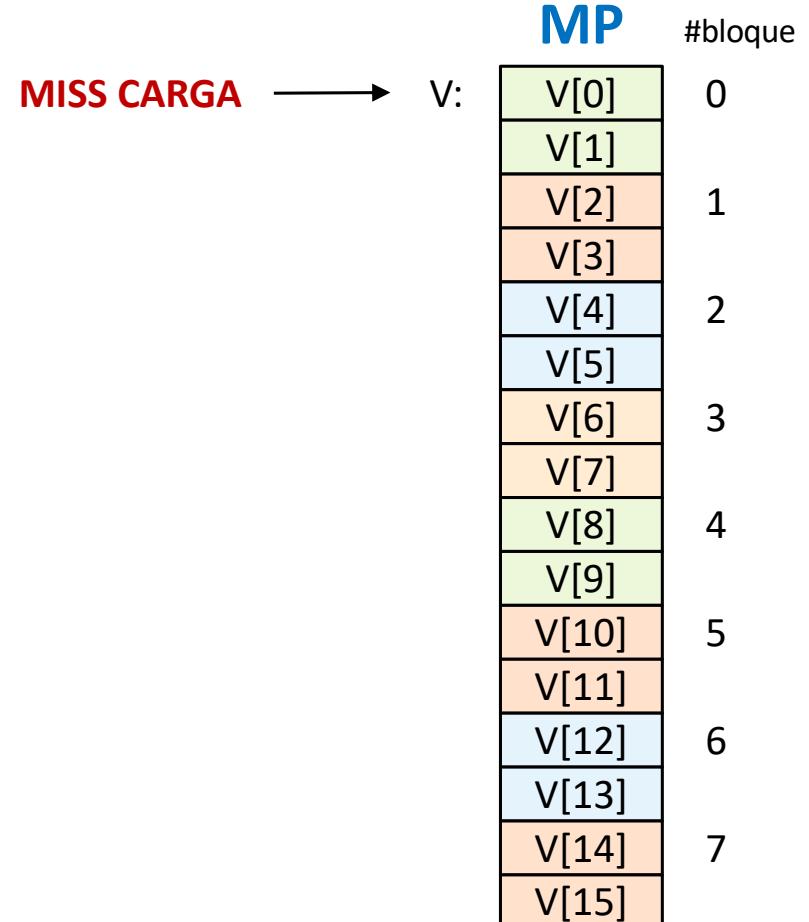
V	Etiquetas	Datos	
0	0		
1	0		
2	0		
3	0		

MP	#bloque
V[0]	0
V[1]	1
V[2]	1
V[3]	2
V[4]	2
V[5]	3
V[6]	3
V[7]	4
V[8]	4
V[9]	5
V[10]	5
V[11]	6
V[12]	6
V[13]	7
V[14]	7
V[15]	7

Ejemplo de tipos de fallos

```
int V[16], i, s=0;  
for (i=0; i<16; i++)  
    s = s + V[i];  
for (i=0; i<16; i++)  
    s = s + V[i]*3;
```

- MC
- Cache Completamente Asociativa
 - 4 Líneas de 8 bytes (2 words)
 - Procesador 32 bits



V	Etiquetas	Datos	
0	1	V[0]	V[1]
1	0		
2	0		
3	0		

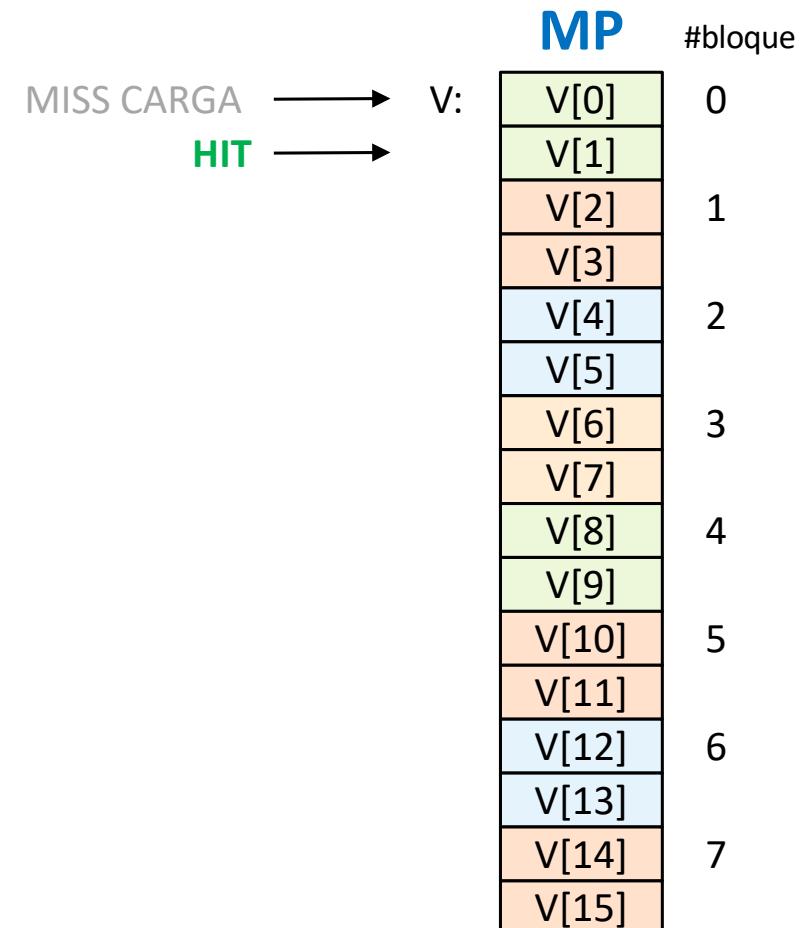
Ejemplo de tipos de fallos

```
int V[16], i, s=0;  
for (i=0; i<16; i++)  
    s = s + V[i];  
for (i=0; i<16; i++)  
    s = s + V[i]*3;
```

- Cache Completamente Asociativa
- 4 Líneas de 8 bytes (2 words)
- Procesador 32 bits

MC

V	Etiquetas	Datos	
0	1	V[0]	V[1]
1	0		
2	0		
3	0		



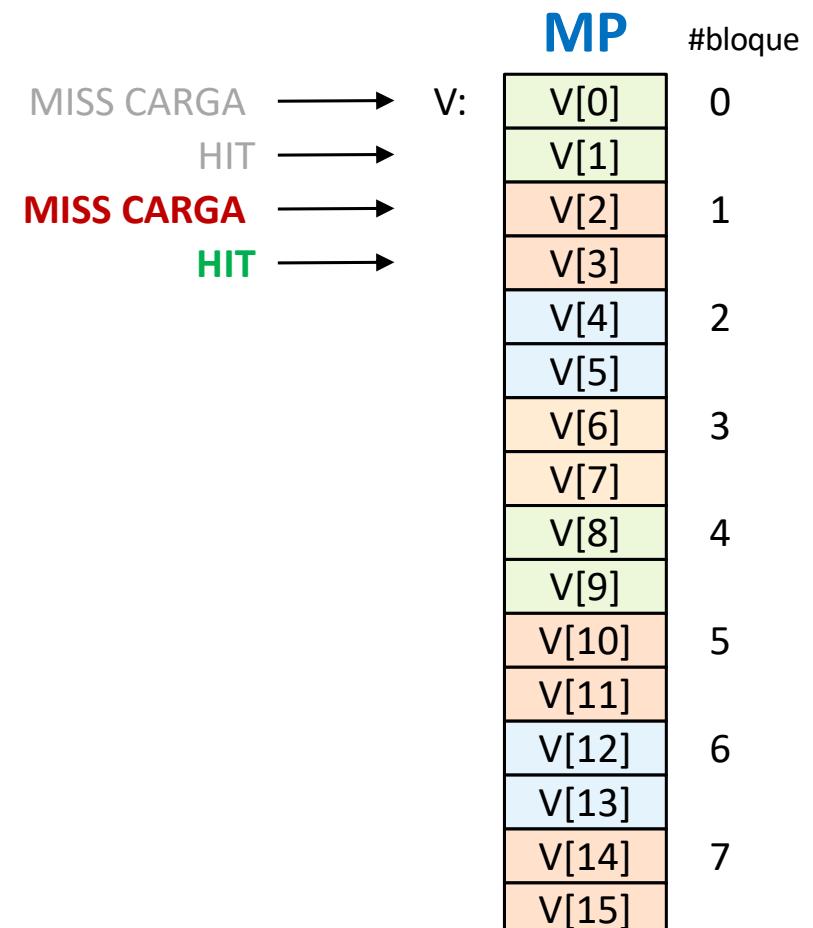
Ejemplo de tipos de fallos

```
int V[16], i, s=0;  
for (i=0; i<16; i++)  
    s = s + V[i];  
for (i=0; i<16; i++)  
    s = s + V[i]*3;
```

- Cache Completamente Asociativa
- 4 Líneas de 8 bytes (2 words)
- Procesador 32 bits

MC

V	Etiquetas	Datos	
0	1	V[0]	V[1]
1	1	V[2]	V[3]
2	0		
3	0		



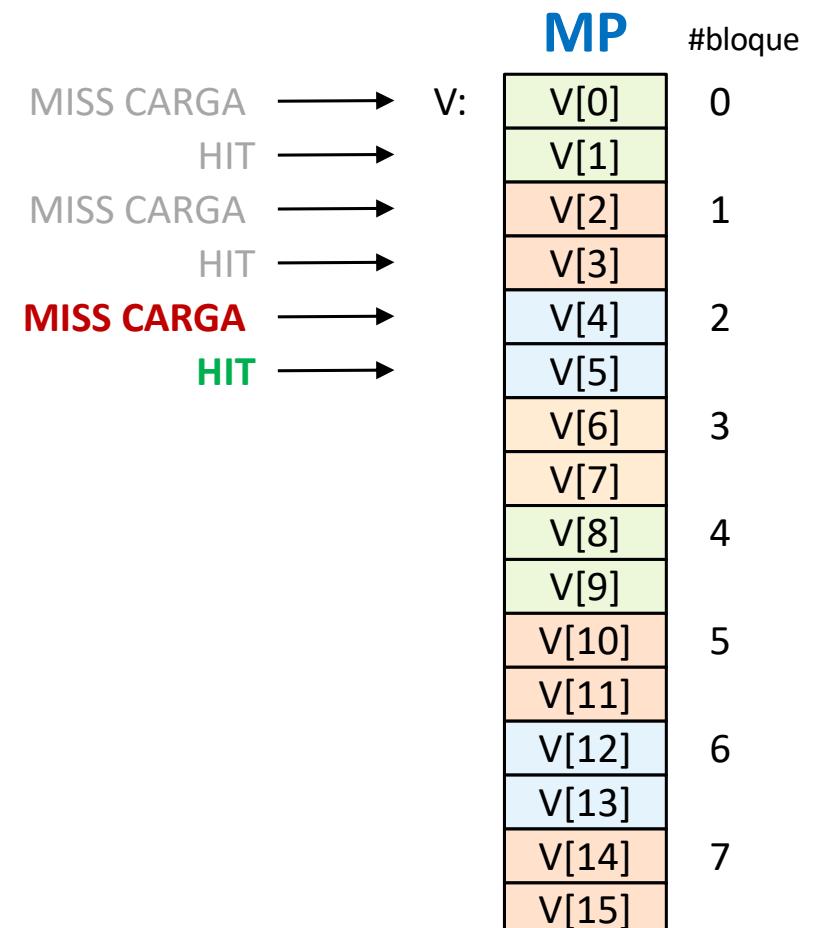
Ejemplo de tipos de fallos

```
int V[16], i, s=0;  
for (i=0; i<16; i++)  
    s = s + V[i];  
for (i=0; i<16; i++)  
    s = s + V[i]*3;
```

- Cache Completamente Asociativa
- 4 Líneas de 8 bytes (2 words)
- Procesador 32 bits

MC

V	Etiquetas	Datos	
0	1	V[0]	V[1]
1	1	V[2]	V[3]
2	1	V[4]	V[5]
3	0		



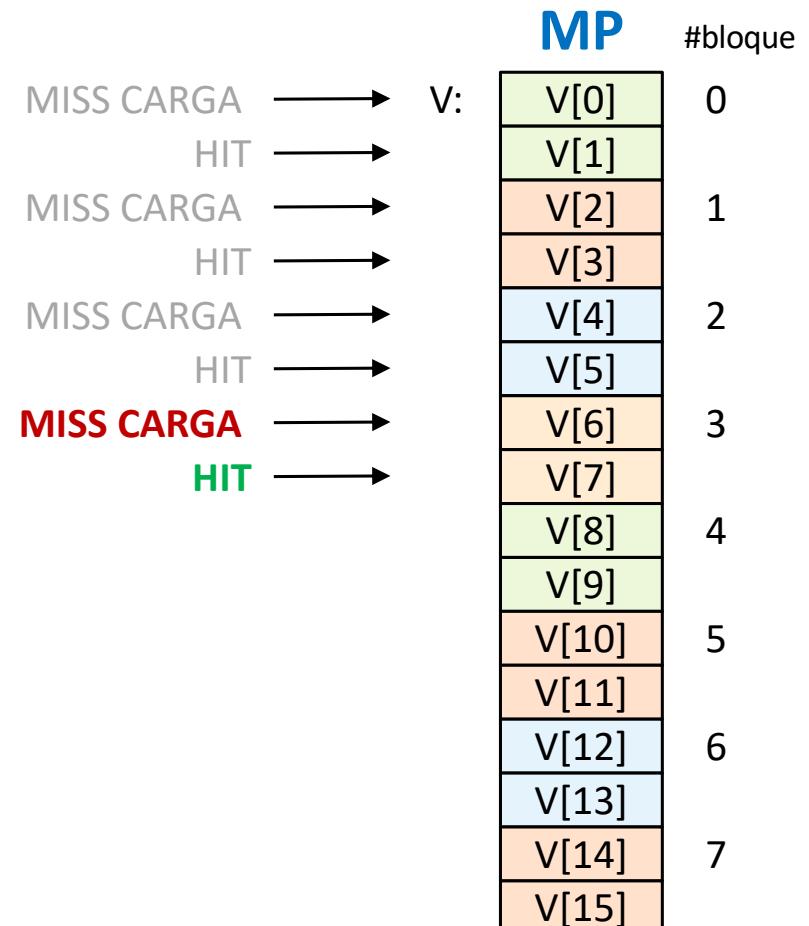
Ejemplo de tipos de fallos

```
int V[16], i, s=0;  
for (i=0; i<16; i++)  
    s = s + V[i];  
for (i=0; i<16; i++)  
    s = s + V[i]*3;
```

- Cache Completamente Asociativa
- 4 Líneas de 8 bytes (2 words)
- Procesador 32 bits

MC

V	Etiquetas	Datos	
0	1	V[0]	V[1]
1	1	V[2]	V[3]
2	1	V[4]	V[5]
3	0	V[6]	V[7]



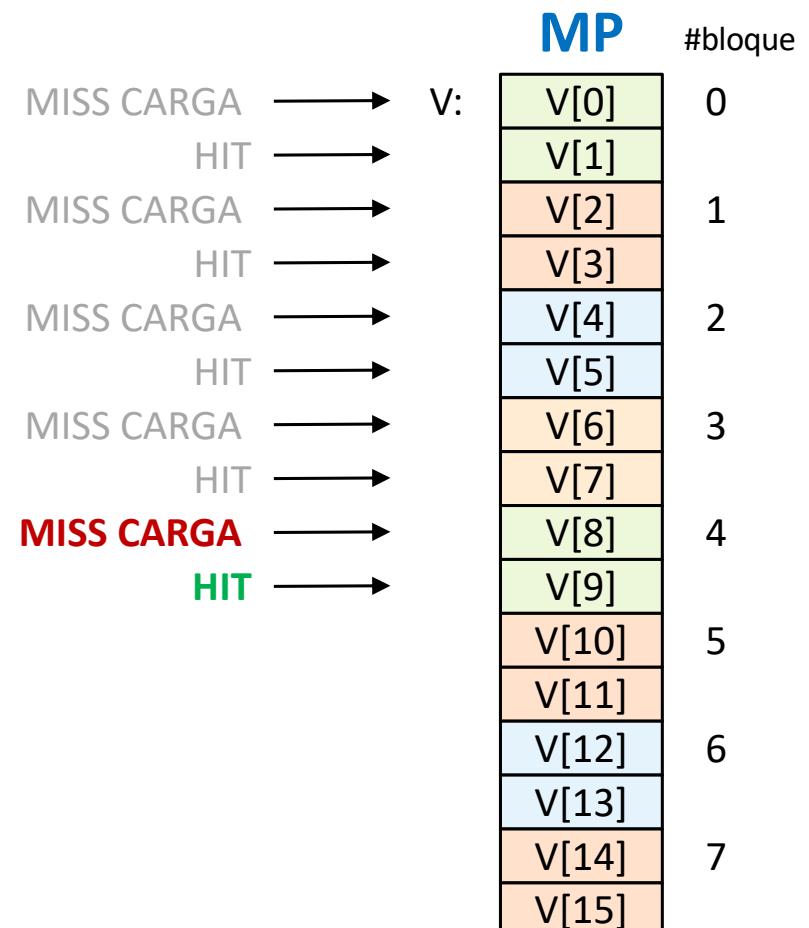
Ejemplo de tipos de fallos

```
int V[16], i, s=0;  
for (i=0; i<16; i++)  
    s = s + V[i];  
for (i=0; i<16; i++)  
    s = s + V[i]*3;
```

- Cache Completamente Asociativa
 - 4 Líneas de 8 bytes (2 words)
 - Procesador 32 bits

MC

V	Etiquetas	Datos	
0	1	V[8]	V[9]
1	1	V[2]	V[3]
2	1	V[4]	V[5]
3	1	V[6]	V[7]



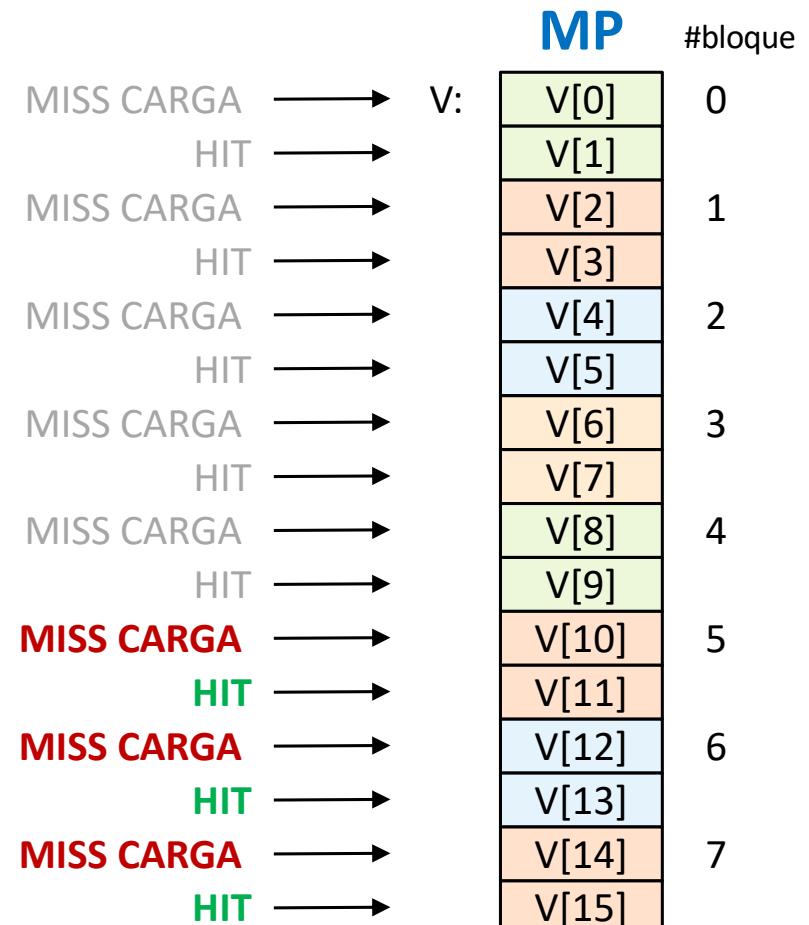
Ejemplo de tipos de fallos

```
int V[16], i, s=0;  
for (i=0; i<16; i++)  
    s = s + V[i];  
for (i=0; i<16; i++)  
    s = s + V[i]*3;
```

- Cache Completamente Asociativa
- 4 Líneas de 8 bytes (2 words)
- Procesador 32 bits

MC

V	Etiquetas	Datos	
1	1	V[8]	V[9]
2	1	V[10]	V[11]
3	1	V[12]	V[13]
4	1	V[14]	V[15]



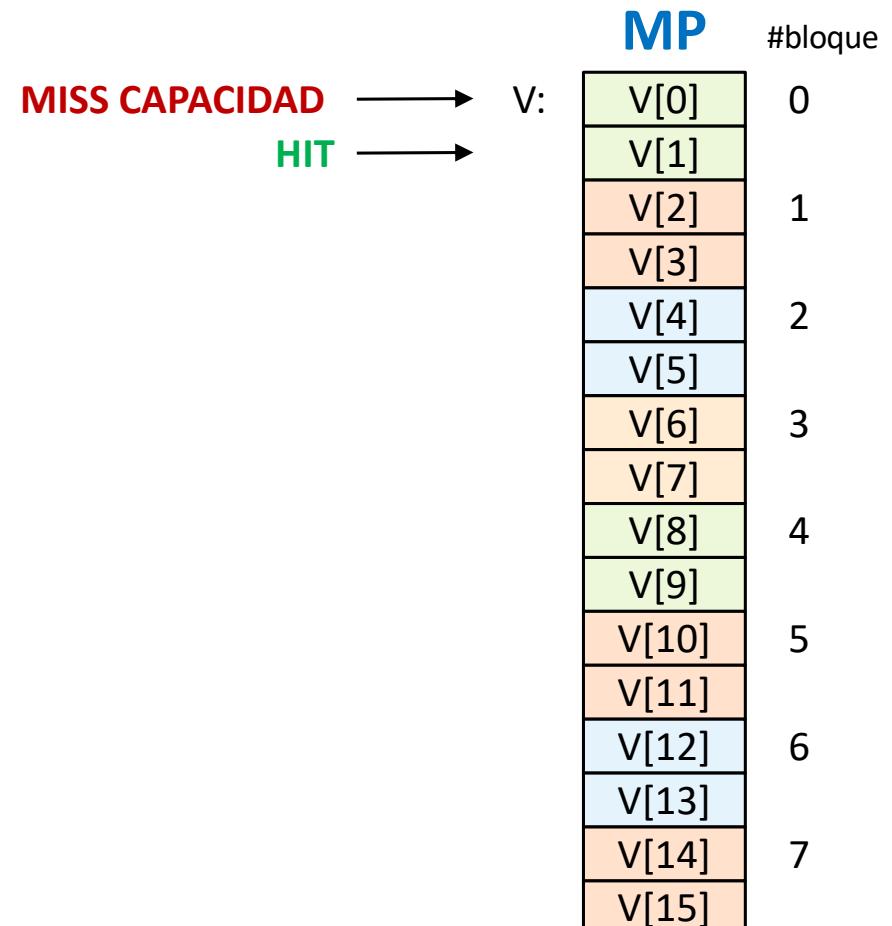
Ejemplo de tipos de fallos

```
int V[16], i, s=0;  
for (i=0; i<16; i++)  
    s = s + V[i];  
for (i=0; i<16; i++)  
    s = s + V[i]*3;
```

- Cache Completamente Asociativa
- 4 Líneas de 8 bytes (2 words)
- Procesador 32 bits

MC

V	Etiquetas	Datos	
1	1	V[0]	V[1]
2	1	V[10]	V[11]
3	1	V[12]	V[13]
4	1	V[14]	V[15]



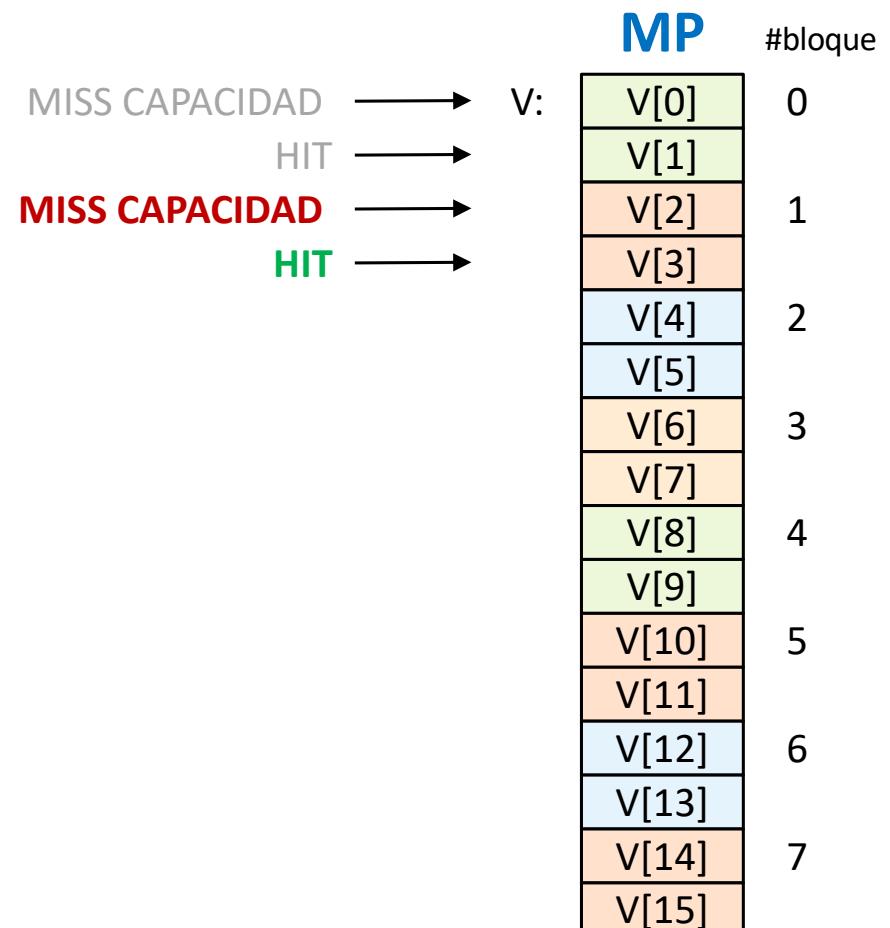
Ejemplo de tipos de fallos

```
int V[16], i, s=0;  
for (i=0; i<16; i++)  
    s = s + V[i];  
for (i=0; i<16; i++)  
    s = s + V[i]*3;
```

- Cache Completamente Asociativa
- 4 Líneas de 8 bytes (2 words)
- Procesador 32 bits

MC

V	Etiquetas	Datos	
1	1	V[0]	V[1]
2	1	V[2]	V[3]
3	1	V[12]	V[13]
4	1	V[14]	V[15]



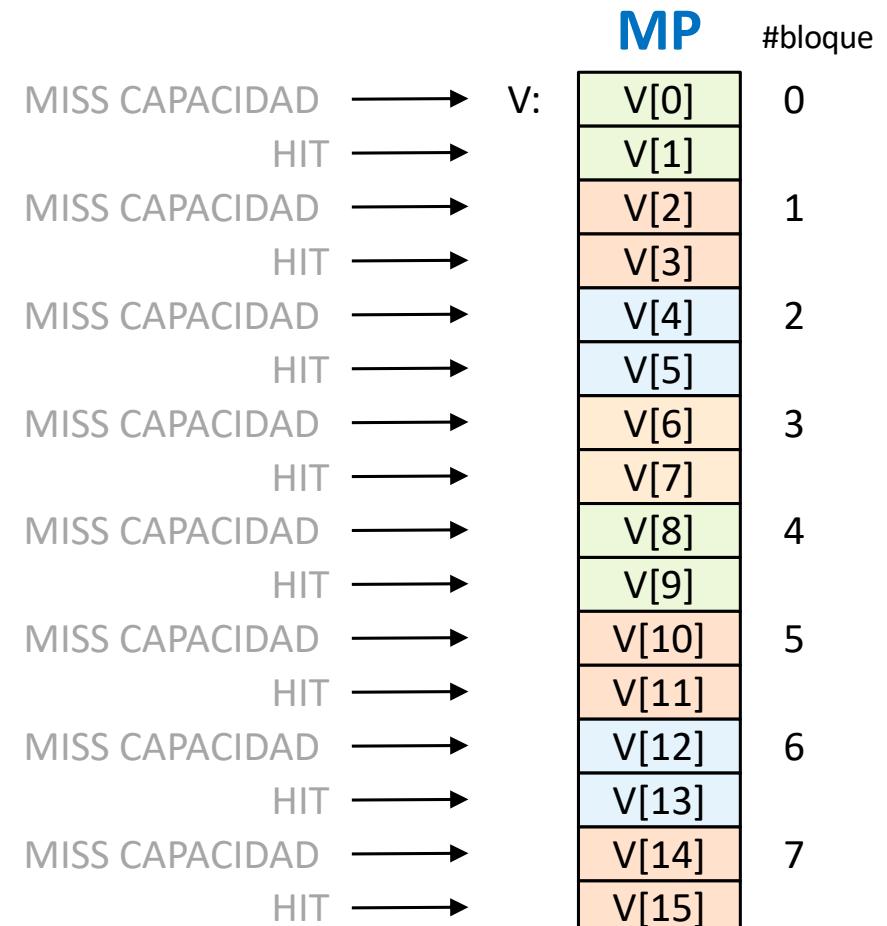
Ejemplo de tipos de fallos

```
int V[16], i, s=0;  
for (i=0; i<16; i++)  
    s = s + V[i];  
for (i=0; i<16; i++)  
    s = s + V[i]*3;
```

- Cache Completamente Asociativa
- 4 Líneas de 8 bytes (2 words)
- Procesador 32 bits

MC

V	Etiquetas	Datos	
1	1	V[8]	V[9]
2	1	V[10]	V[11]
3	1	V[12]	V[13]
4	1	V[14]	V[15]





UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament d'Arquitectura de Computadors

Estructura de Computadores

Tema 6: Memoria Cache

Agustín Fernández

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya

