NOTAS DE MEMORIA CACHE

ETC- Enero 2018

1. Geometría de la cache

La memoria cache (MC) se define con arreglo a su tamaño (en bytes), número de vías y tamaño de bloque (en bytes). Son lo que se denominan parámetros básicos de la cache.

{Tamaño_MC, num_vías, Tamaño_Bloque}

Los bloques en la cache se agrupan formando conjuntos. El número de vías representa el número de bloques que forman cada uno de los conjuntos y es siempre potencia de 2.

El esquema muestra una MC de 128 Bytes con tamaño de bloque de 16 Bytes y 2 vías, esto es, {128B, 2 vías, 16B}

Conjunto	MC	Bloque	
0	16B	0	128B
	16B	1	
1	16B	2	
	16B	3	
2	16B	4	
	16B	5	
3	16B	6	
	16B	7	

Obsérvese que tanto los bloques como los conjuntos se enumeran de forma consecutiva empezando por "cero". Los bloques de la cache se conocen también como "líneas" de cache

A partir de los parámetros básicos se obtiene un par de características muy importantes de la cache

$$num\_total\_bloques\_MC = \frac{tamaño\_MC}{tamaño\_bloque} \quad (1)$$

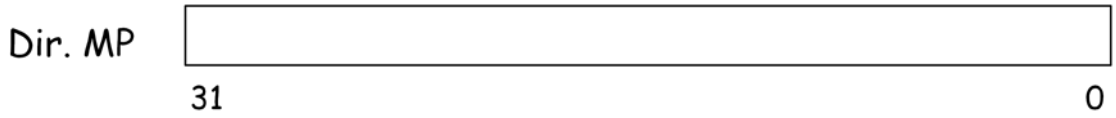
$$num\_total\_conjuntos\_MC = \frac{num\_total\_bloques\_MC}{num\_vias} \quad (2)$$

En el ejemplo anterior, la cache se estructura en 8 bloques de 16B agrupados de dos en dos (número de vías igual a 2) para constituir un total de 4 conjuntos. Sin embargo, obsérvese que existen dos casos extremos, a saber: (1) aquel en el que el número de vías es igual a 1, con lo que el número de conjuntos iguala al número de bloques y (2) aquel en el que el número de vías tiene el máximo valor, esto es, el número de bloques, por lo que sólo existirá un único conjunto que engloba a todos los bloques.

2. Interpretación de la dirección de memoria (1): Cálculo del número de bloque

A partir de la geometría de la MC es posible estructurar en determinados campos la dirección de memoria principal.

Para empezar, como es sabido, el tamaño de la dirección de memoria (número de bits que la componen) depende del tamaño del espacio físico direccionable por el procesador. En MIPS, dicho espacio direccionable es de 4GB, por lo que las direcciones generadas por el procesador tendrán un tamaño de 32 bits.

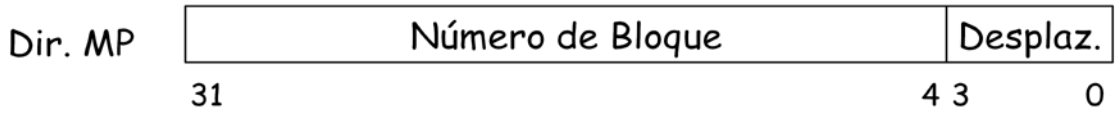


Dado que la memoria principal (MP) se compone de bloques, es posible, a partir de una dirección, identificar fácilmente el bloque (número de bloque) al que pertenece. Para ello, el siguiente esquema muestra cómo se van direccionando los distintos bytes de la MP.

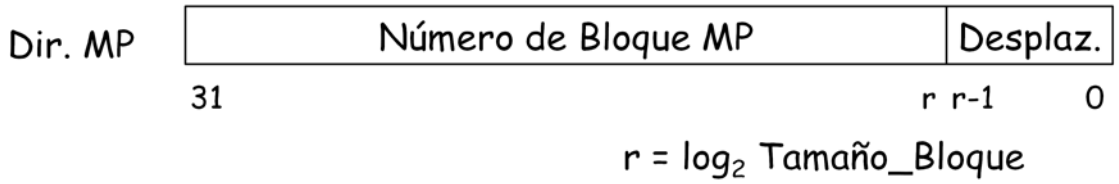
Bloque	MP	Dirección
Bloque 0	Byte 0	0x00000000
	Byte 1	0x00000001
	Byte 2	0x00000002
	……	
	Byte 14	0x0000000E
	Byte 15	0x0000000F
Bloque 1	Byte 16	0x00000010
	Byte 17	0x00000011
	Byte 18	0x00000012
	…..	
	Byte 30	0x0000001E
	Byte 31	0x0000001F
Bloque 2	Byte 32	0x00000020
	Byte 33	0x00000021
	Byte 34	0x00000022
	……	
	Byte 46	0x0000002E
	Byte 47	0x0000002F
	….	

Si se asume la existencia de una MC con tamaño de bloque 16B, cada bloque estará compuesto de 16 bytes de memoria consecutivos, por lo que los bloques resultantes son los que se muestran en el esquema. Se aprecia cómo los 28 bits más significativos (31 …4)

determinan justamente el número de bloque al que pertenece la dirección (campo <Número de Bloque>), mientras que los 4 bits menos significativos (3…0) determinan el desplazamiento del byte dentro del bloque (campo <Desplazamiento>). La dirección de MP queda descompuesta, inicialmente, en los siguientes campos:



¿Cómo se determina el número de bits del campo <Desplazamiento>? Sencillamente, a partir del parámetro Tamaño del Bloque. Como éste es igual a 16B (2^4 bytes), se necesitan 4 bits para codificar todos y cada uno de los bytes que lo integran. Observe que una vez establecido el tamaño del campo <Desplazamiento>, el resto de bits constituyen automáticamente el campo <Número de Bloque>. Generalizando a cualquier tamaño de bloque



En el ejemplo anterior, los Número de Bloque correspondientes a los bloques reseñados serán:

Bloque	**Num_Bloque**
0	0x0000000
1	0x0000001
2	0x0000002
3	0x0000003
…..	…..

En realidad, los campos <Número de Bloque> y <Desplazamiento> se pueden obtener también a través de la aplicación de simples operaciones aritméticas, a saber:

$$\text{Num\_Bloque} = (\text{Dir\_MP}) \ \text{DIV} \ (\text{Tamaño\_Bloque}) \qquad (3)$$

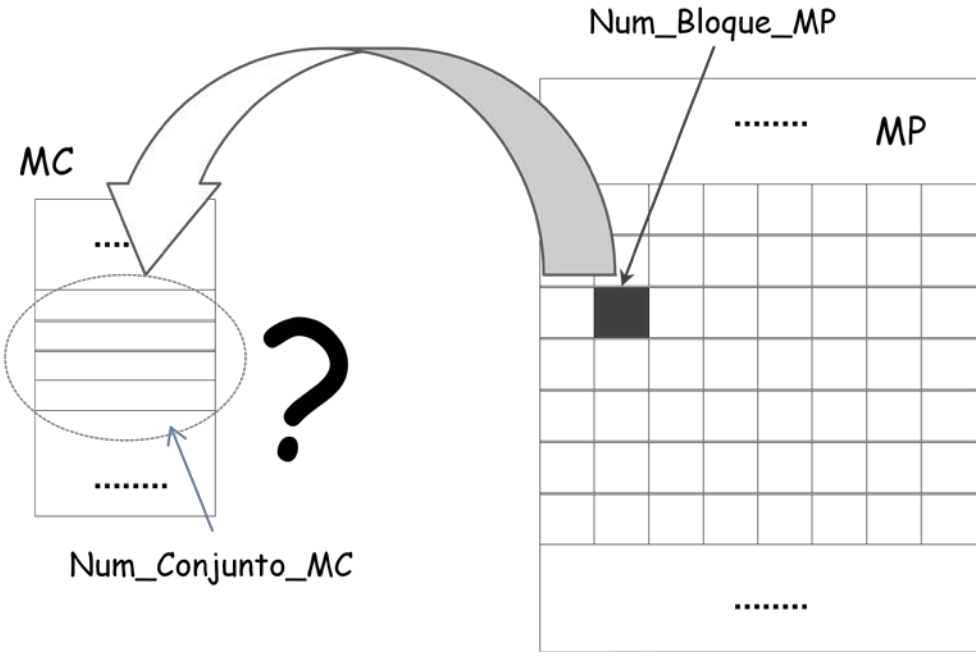
$$\text{Desplazamiento} = (\text{Dir MP}) \ \text{MOD} \ (\text{Tamaño Bloque}) \qquad (4)$$

En consecuencias, los campos <Número de Bloque> y <Desplazamiento> se obtienen como el cociente y el resto, respectivamente, de dividir el valor de la dirección de memoria por el tamaño de bloque.

3. Funciones de correspondencia

Las funciones de correspondencia pretenden dar respuesta a la pregunta de en qué lugar (Conjunto) de la memoria cache se debe almacenar un cierto bloque de memoria principal cuando éste es llevado a la cache.

La correspondencia se establece siempre entre un BLOQUE de la MP y un CONJUNTO de la MC: Num_Bloque_MP → Conjunto_MC



Observe que lo que hace una función de correspondencia es, en realidad, una clasificación de los Bloques de MP; según la clase, se almacenan en un Conjunto u otro. Una vez se determina el Conjunto, el Bloque de MP se puede almacenar, en principio, indistintamente en cualquiera de los bloque/líneas/vías que lo forman.

El tipo de correspondencia depende simplemente del número de vías definidas en la geometría de la MC, al cual se conoce también por "grado de asociatividad". Existen tres tipos de correspondencia, como se define en la siguiente tabla:

Correspondencia	**Num_vías**	**Num_total_conjuntos_MC**
Directa	1	num_total_bloques_MC
Asociativa por conjuntos	n ; $2 \leq n \leq$ num_total_bloques_MC/2	num_total_bloques_MC/n
Totalmente asociativa	num_total_bloques_MC	1

Obsérvese que en el caso de la correspondencia Directa, como el Conjunto contiene una sola línea, la correspondencia se establece finalmente entre un Num_Bloque_MP y un

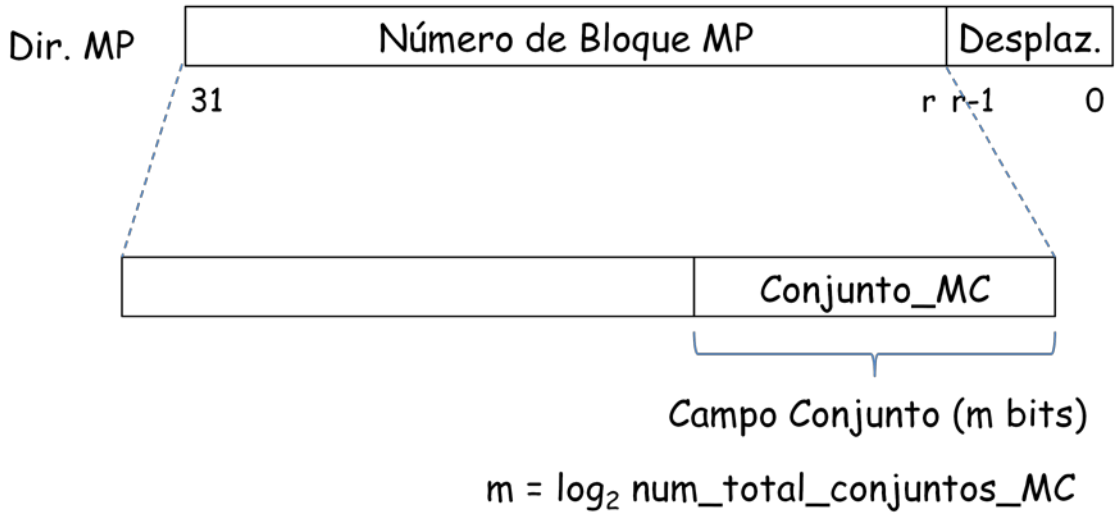
bloque o línea de memoria cache. En el otro caso extremo, correspondencia Totalmente Asociativa, el Bloque de MP se podría almacenar en cualquiera de los bloques de la cache, al estar todos ellos contenidos en el único conjunto existente.

Ahora, la pregunta es ¿cómo clasificamos los Bloques de MP para poder determinar el Conjunto de MC al que corresponde? La respuesta es sencilla, aplicando la operación MOD sobre el Número de Bloque de MP respecto al total de conjuntos de MC, esto es:

$$\text{Conjunto\_MC} = (\text{Num\_Bloque\_MP}) \ \text{MOD} \ (\text{Num\_Total\_Conjuntos\_MC}) \qquad (5)$$

Ello significa que el Conjunto de MC al que se mapea un cierto Bloque de MP corresponde al valor del resto de la división entre el Número de Bloque de MP y el número total de Conjuntos de la MC.

Existe una forma sencilla (es la seguida por el hardware) para determinar el Conjunto de MC a partir de una simple inspección de los bits que componen el Número de Bloque de MP, de forma análoga a como se determinaban los valores del Número de Bloque de MP y del Desplazamiento a partir de la dirección de MP. Ello nos lleva a estructurar, a su vez, el campo Número de Bloque de MP en dos subcampos:



Obsérvese que los m bits menos significativos del campo Número de Bloque de MP se corresponden con el valor del resto de dividir dicho número de bloque entre el total de conjuntos de la MC (operación MOD), lo que equivale en realidad a una clasificación. El campo Conjunto determina el Conjunto de MC al que se habrá de mapear el Bloque de MP.

Ello nos plantea otro problema, como todos los Bloques de MP con el mismo contenido en el campo Conjunto se mapearán al mismo Conjunto de la MC (nunca al mismo tiempo, pues el número de vías y el tamaño de la cache son limitados), la pregunta que nos deberíamos

hacer es ¿cómo es posible identificar el Bloque de MP que está alojado en el Conjunto de MC en un momento dado?

Tratemos de comprenderlo con un sencillo ejemplo. Supóngase que sólo existen 4 conjuntos en la cache. El tamaño del campo conjunto deberá ser de 2 bits (2^2=4). Cuando se direccionan consecutivamente los distintos bloques de MP (direcciones en binario), tal y como se muestra en el esquema, los dos bits menos significativos indican el conjunto (uno entre cuatro) al que se mapea el bloque de MP.

MP	Num_Bloque_MP	Conjunto_MC
Bloque 0	000…000 00	0
Bloque 1	000…000 01	1
Bloque 2	000…000 10	2
Bloque 3	000…000 11	3
Bloque 4	000…001 00	0
Bloque 5	000…001 01	1
Bloque 6	000…001 10	2
Bloque 7	000…001 11	3
Bloque 8	000…010 00	0
Bloque 9	000…010 01	1
Bloque 10	000…010 10	2
Bloque 11	000…010 11	3
……	…	…

Como se puede observar, los bloques 0, 4 y 8 se mapean al conjunto 0, los blques 1, 5 y 9 se mapean al conjunto 1, y así sucesivamente. Ello significa que dichos bloques pueden almacenarse indistintamente en cualquiera de las vías (líneas) del conjunto al que se mapean ¿Cómo se sabe el bloque de MP que se halla actualmente almacenado? Para ello se hace uso del resto de bits del Número de Bloque de MP, los cuales constituyen el denominado campo ETIQUETA. En el esquema se muestra, por ejemplo, cómo los bloques 0, 4 y 8 se pueden distinguir de forma inequívoca mediante sus correspondientes etiquetas, 0, 1 y 2, respectivamente.

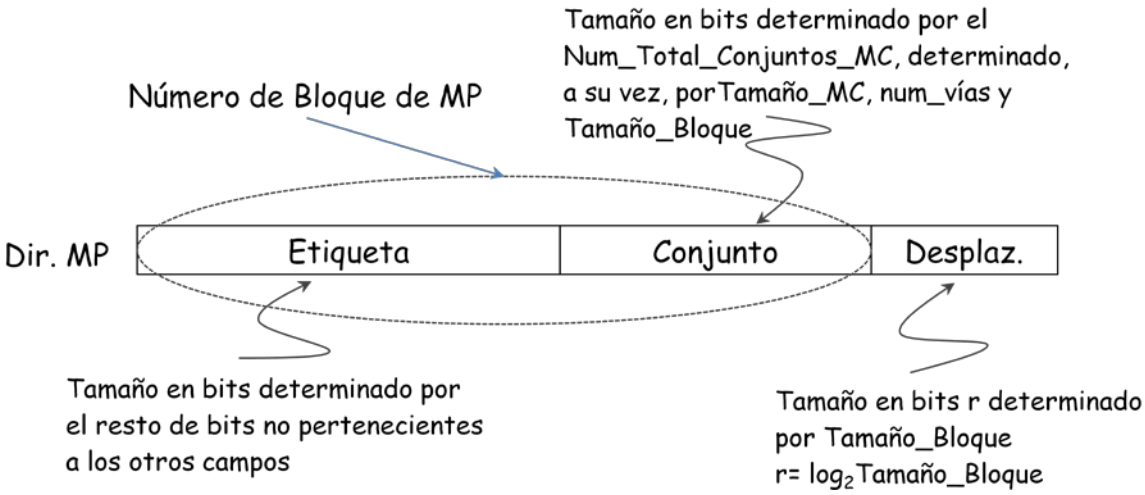
El contenido del campo <Etiqueta> puede determinar también mediante una simple operación aritmética:

$$\text{Etiqueta} = (\text{Num\_Bloque\_MP}) \ \text{DIV} \ (\text{Num\_Total\_Conjuntos\_MC}) \qquad (6)$$

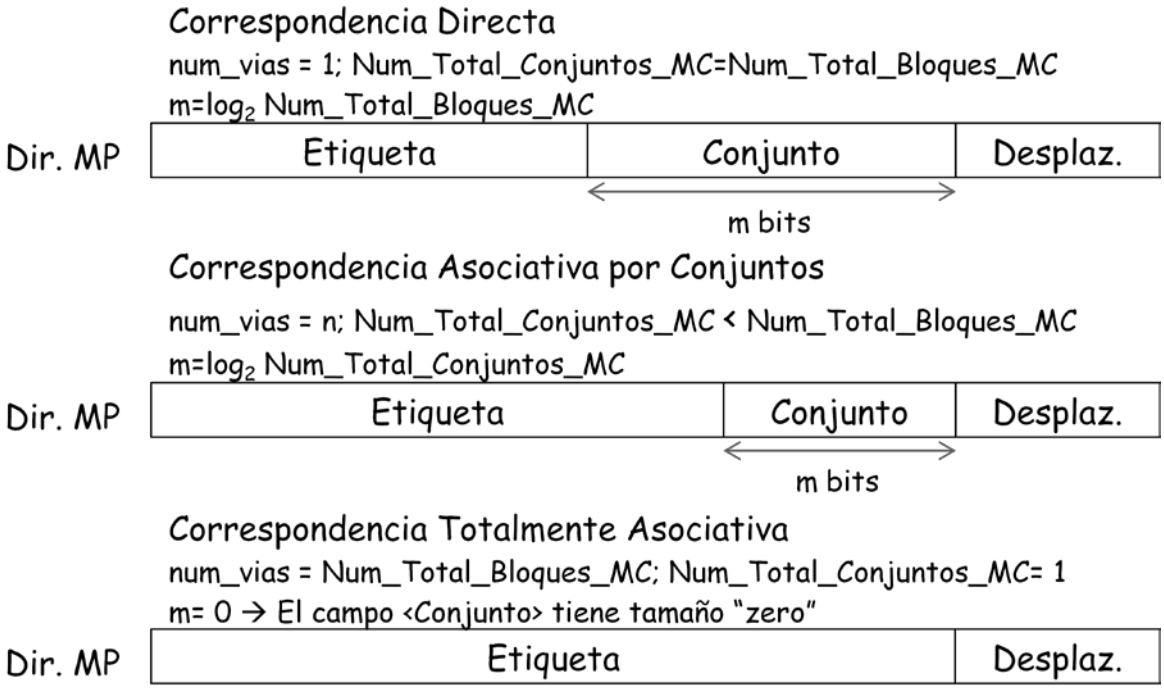
Luego la Etiqueta no es más que el cociente entre el Número de Bloque de MP y el Número Total de Conjuntos de la MC. Recuérdese que el resto de dicha división (operación MOD) correspondía al contenido del campo <Conjunto>.

4. Interpretación de la dirección de memoria (2)

A partir de los parámetros básicos que definen la geometría de la cache, {Tamaño_MC, num_vías, Tamaño_Bloque}, el controlador de la misma hace una interpretación de la dirección de MP que intercepta procedente de la CPU (de su ruta de datos). Según dicha interpretación, se identifican tres campos, como se muestra en el esquema



Obsérvese que, en realidad, para un cierto tamaño de cache y tamaño de bloque, el tamaño de los campos <Conjunto> y <Etiqueta> está determinado por la función de correspondencia aplicada, las cuales difieren únicamente, como se vio anteriormente, en el número de vías que se asumen, como se muestra en el esquema.

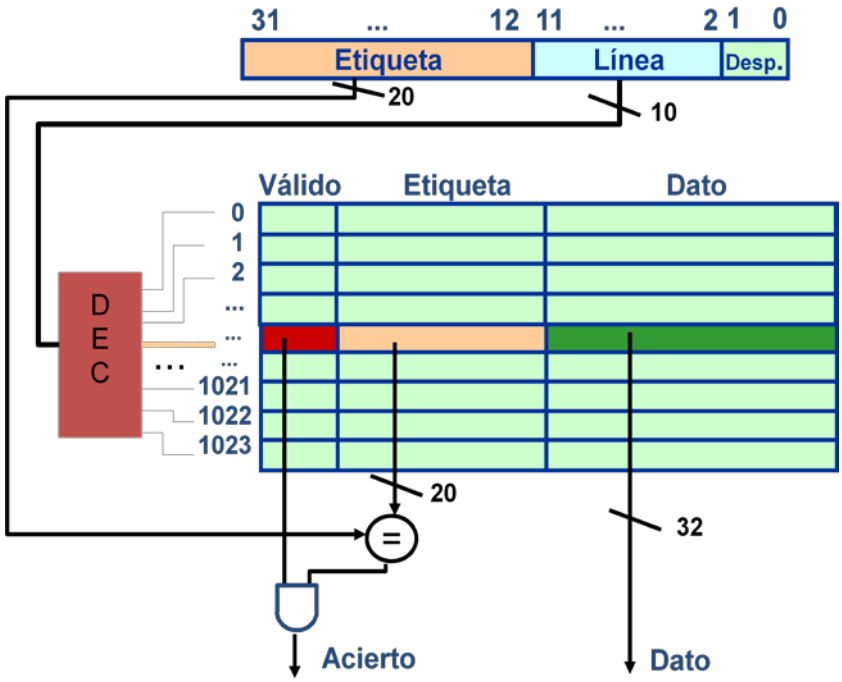


Como se puede apreciar, a medida que aumenta el número de vías, el número de conjuntos disminuye y con ello el tamaño del campo <Conjunto>, disminución que se compensa con el incremento del campo <Etiqueta>. Así, con correspondencia Directa se tiene el mayor tamaño para el campo <Conjunto> y el menor pare el campo <Etiqueta>, mientras que en el otro extremo, con correspondencia Totalmente Asociativa, el tamaño del campo <Etiqueta> se maximiza en detrimento del campo <Conjunto>, el cual desaparece (no se necesita campo para determinar el Conjunto porque solo hay un conjunto). En medio de ambos extremos se encuentra la correspondencia Asociativa por Conjuntos, donde el tamaño del campo <Conjunto> se irá aproximando a uno u otro extremo a medida que se incrementa o disminuye el número de vías (grado de asociatividad).

5. Organización e indexado de la cache

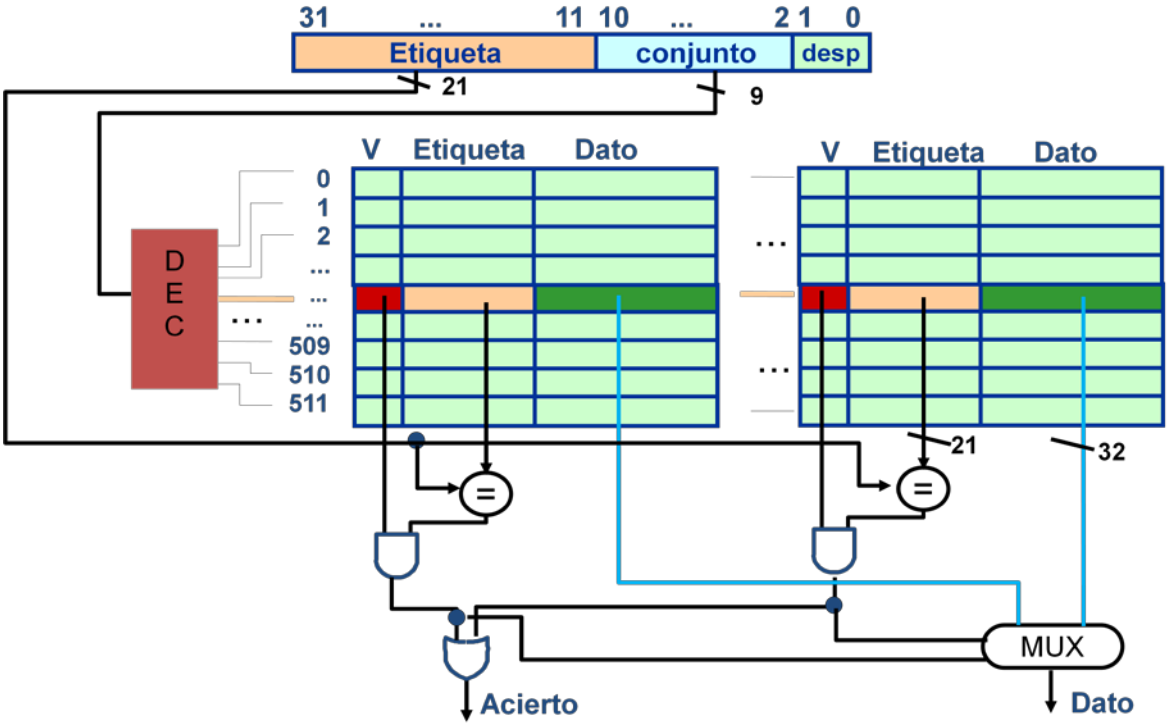
La aplicación de cada una de las funciones de correspondencia introducidas en la sección 3 asume la existencia de una organización particular de la cache. En concreto, las organizaciones posibles serían: directa, organización asociativa por conjuntos y totalmente asociativa. La diferencia fundamental entre las mismas es el número de vías. El conocimiento de dichas organizaciones nos permite comprender cómo se realiza el indexado o direccionamiento de la cache, así como tener una idea aproximada de su complejidad. Las organizaciones que se ilustran a continuación asumen un tamaño de bloque de una palabra.

a) Cache directa



Como se aprecia, la cache se indexa al nivel de línea de cache, empleando para ello el campo línea de la dirección de memoria. La decodificación de dicho campo permite seleccionar una única línea de cache junto a su etiqueta asociada. Para determinar si hay acierto o fallo se ha de comparar el campo etiqueta de la dirección con la etiqueta asociada a la línea de cache. Basta para ello con disponer de un único comparador en la cache. Si las etiquetas coinciden y el bit de válido (bit V) es igual a uno, habrá acierto. En caso contrario, habrá fallo. La comprobación del bit V se realiza con una simple puerta AND. El decodificador tendrá tantas entradas como bits contenga el campo línea de la dirección y tantas salidas como líneas/bloques contenga la cache.

b) Cache asociativa por conjuntos



Como se aprecia, la cache se indexa al nivel de conjunto de cache, empleando para ello el campo conjunto de la dirección de memoria. La decodificación de dicho campo permite seleccionar al mismo tiempo todas las líneas de cache o vías que componen el conjunto seleccionado, junto a sus etiqueta asociada. Para determinar si hay acierto o fallo se ha de comparar el campo etiqueta de la dirección con todas las etiquetas asociadas a las líneas de cache del conjunto. Para ello se requiere disponer de un comparador por cada una de las vías del conjunto. Si hay coincidencia con algunas de las etiquetas y el correspondiente bit V es igual a uno, habrá acierto. En caso contrario, habrá fallo. La comprobación de los bits V se realiza con una simple puerta AND por cada una de las vías del conjunto más una puerta OR que

agrupa las salidas de las anteriores. El decodificador tendrá tantas entradas como bits contenga el campo conjunto de la dirección y tantas salidas como conjuntos contenga la cache. A diferencia de la cache directa, en este caso se precisa de un multiplexor a la salida para volcar al exterior, en caso de acierto, la línea de cache cuya etiqueta coincide con el campo etiqueta de la dirección. La complejidad y retardo del multiplexor se incrementa a medida que aumenta el número de vías.

c) Cache totalmente asociativa

El esquema organizativo sería, en realidad, un caso particular del anterior, en el que sólo existiría un conjunto, el cual tendría el total de líneas de la cache como número de vías. Al existir un solo conjunto se eliminaría el decodificador, con lo que el indexado de la cache sería trivial (ausencia de campo conjunto en la dirección de memoria). La consecuencia en el hardware sería el empleo de tantos comparadores como líneas tiene la cache y un multiplexor de mucho mayor tamaño. El balance, no obstante, es un incremento de la complejidad y del tiempo de acceso. Este último, a consecuencia del mayor retardo del multiplexor.

6. Algoritmo de Reemplazo

En caso de correspondencia con grado de asociatividad (num_vias ≠ 1) se requiere un mecanismo para determinar, en caso de fallo, qué bloque debe ser reemplazado, de entre los pertenecientes a un cierto conjunto, por el bloque que originó el fallo y que se ha de almacenar en dicho conjunto de la cache según la función de correspondencia.

Algoritmo LRU

si Fallo entonces
reemplazar línea(i) / contador(i)= 2^n-1
contador(j)++ ∀j≠i
contador(i)= 0
si Acierto entonces
contador(j)++ ∀j / contador(j)<contador(i)
contador(i)= 0

El algoritmo comúnmente empleado es el algoritmo LRU, el cual selecciona como bloque/línea a ser reemplazado dentro del conjunto a aquel que hace más tiempo que no se ha accedido (el menos recientemente usado). Dicho algoritmo requiere asociar un contador con cada una de las líneas/bloques de la cache. Como la aplicación del algoritmo LRU se hace al nivel del conjunto al que se mapea el bloque que originó el fallo, el tamaño (número

de bits) del contador será igual a $\log_2$ num_vias. De esta forma, y gracias a la mecánica del algoritmo, se garantiza que cada línea/vía del conjunto tenga un valor distinto en su contador LRU asociado. El máximo valor del contador será, pues, igual a num_vias-1.

Cuando se produce un fallo, aquel bloque/línea cuyo contador posea el máximo valor será seleccionado para ser reemplazado. Una vez reemplazado por el nuevo bloque, su contador se pone a cero y se incrementa en una unidad el resto de contadores.

Cuando se produce acierto, el contador del bloque referenciado se pone a cero y se incrementa en una unidad el contador de los bloques/líneas cuyo contador posee un valor inferior al del bloque referenciado (antes de ser inicializado a cero).

…									
a/f	2	3	3	0	1	2	3	3	← Contador LRU
b/c	0	1	0	1	2	3	0	1	
c/e	3	0	1	2	0	1	2	2	
d/a	1	2	2	3	3	0	1	0	
…		e	b	f	e	a	c	a	← Bloque accedido
		F	A	F	A	F	F	A	←**A**cierto/**F**allo

En el esquema se muestra cómo evolucionan los contadores LRU asociados a las 4 vías de un conjunto de la memoria cache. Se parte del estado inicial en el que el conjunto alberga los bloques a, b, c y d. Los contadores evolucionan a medida que se van accediendo a bloques y en función de que se produzca Acierto ó Fallo. Al final el conjunto contendrá los bloquea f, c, e y a. Obsérvese que la mecánica del algoritmo evita que los contadores lleguen a desbordar.

7. Políticas de escritura

A diferencia de las lecturas, en las que en caso de fallo la única opción prevista es copiar el bloque de memoria a la cache, las operaciones de escritura, tanto en caso de fallo como de acierto, son susceptibles de adoptar distintas políticas.

A) Fallo en escritura

En caso de fallo al ir a escribir cabe adoptar dos posibles políticas:

- Ubicar el bloque en la cache (*write allocate*). Es la misma política seguida en el caso de una operación de lectura.

- NO Ubicar el bloque en la cache (*write no-allocate*). En este caso de opta por no copiar el bloque a la cache y escribir directamente sobre la memoria principal. ¿Qué sentido tiene hacer esto? Pues se evita tener que reemplazar otros bloques de la cache que posiblemente estén siendo accedidos, lo que evita incurrir en posteriores fallos. Además, en escritura no se explota tanto el principio de localidad, como sí ocurre, por el contrario, en el caso de lectura, por lo que son dudosas las ventajas que se obtendrían alojando los bloque en la cache. Como inconveniente estaría la ralentización de la operación de escritura, sin embargo ello se soluciona con el empleo de los denominados buffers de escritura. La CPU escribe directamente sobre dicho buffer y éste ya se encarga de ir transfiriendo su contenido a memoria al ritmo que esta última establezca.

B) Acierto en escritura

En caso de acierto al ir a escribir cabe adoptar también dos posibles políticas, las cuales se conocen como **políticas de actualización** en escritura:

- Escritura directa (*write through*). El byte o la palabra se actualizan al mismo tiempo en la cache y en la memoria principal. Como la memoria está actualizada en todo momento, en caso de reemplazo del bloque en la cache bastará con eliminarlo almacenando encima el nuevo bloque, lo que agiliza la resolución del fallo. Requiere el empleo de buffer de escritura para evitar ralentizar la operación de escritura, si bien incrementa el tráfico hacia la memoria.
- Escritura posterior (write back). El byte o la palabra se actualizan únicamente en la cache, por lo que en caso de reemplazo de un bloque que haya sido modificado se requerirá su copia a memoria antes de ser eliminado por el almacenamiento del nuevo bloque. Dicha operación puede alargar el tiempo de resolución del fallo. Para saber si un bloque ha sido modificado se requiere de un bit M (bit de modificado) asociado a cada bloque. Si M=0, el bloque en memoria se mantiene coherente con la cache, por lo que no se requiere actualización previa al reemplazo de dicho bloque. Si M=1, los contenidos del bloque en la cache y en la memoria principal difieren, por lo que el bloque deberá ser copiado a memoria antes de ser reemplazado.

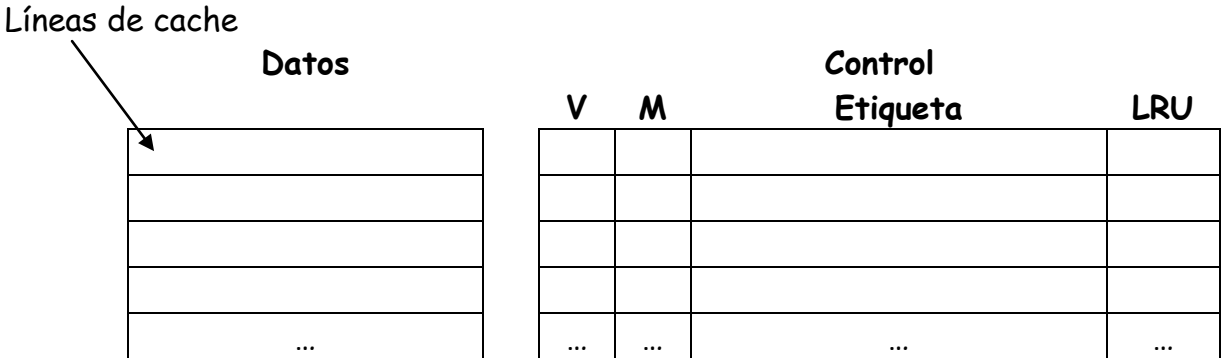
8. Memoria de Control

La cache no está únicamente constituida por la memoria que alberga los datos (bloques de memoria), sino que junto a ella se halla siempre lo que se denomina Memoria de Control, la cual se necesita para almacenar información necesaria para poder identificar qué bloques de memoria se hallan almacenados en la cache y gestionar el reemplazo de los mismos cuando estos deben ser desalojados de la cache ante la llegada de un nuevo bloque.

La pregunta que nos hacemos es ¿qué contenido se almacena en la Memoria de Control y cómo se halla estructurado? La Memoria de Control debe contener una entrada de información por cada línea/bloque de la cache. La estructura de cada entrada es la que se muestra en el esquema

V	M	Etiqueta	LRU

V: Bit de válido
M: Bit de modificado
Etiqueta: Bits del campo etiqueta de la direccción del bloque almacenado
LRU: Contador para alg. de reemplazo (tamaño= $\log_2$ num_vias)



El bit V indica si la línea de cache asociada tiene almacenado un bloque (V=1). El bit M existe sólo si se aplica política de escritura write_back (copy_back). Dicho bit indica si el bloque que se halla almacenado ha sido modificado (M=1), en cuyo caso deberá ser copiado a memoria antes de ser reemplazado. La Etiqueta sirve para identificar qué bloque se halla almacenado en la línea de cache asociada, entre los potenciales bloques de MP que se mapean al conjunto al que pertenece dicha línea de cache. Nótese que el hardware es el encargado de inicializar este campo, extraído de la propia dirección de memoria, al mismo tiempo que se almacena el bloque en la cache. El tamaño del campo Etiqueta será, pues, coincidente, con el campo <Etiqueta> de la dirección de memoria, cuyo tamaño será igual al resultado de eliminar de la dirección de memoria los bits de los campos <Conjunto> y <Desplazamiento>. Finalmente, el campo LRU existe únicamente cuando se aplica una

correspondencia que exhibe algún grado de asociatividad (num_vias ≠ 1). En consecuencia, con correspondencia Directa no existe campo LRU. Este campo consiste en un contador necesario para aplicar el algoritmo de reemplazo LRU. El tamaño de dicho campo será igual a $\log_2$ num_vias.

El tamaño de la Memoria de Control será, pues, el resultado de multiplicar el número de bits de cada una de las entradas por el total de líneas o bloques de MC, esto es;

Num_Total_Bloques_MC x [1+(1)+Tamaño_<Etiqueta>+($\log_2$ num_vias)]

Donde entre paréntesis se indican aquellos campos que pueden o estar en función de la política de escritura o la función de correspondencia. La parte de la memoria de control que alberga las etiquetas también se la conoce con el nombre de DIRECTORIO.

9. Tipos de fallo

Los fallos en la cache se pueden clasificar en tres tipos, lo que se conoce comúnmente como CCC (Cold, Capacity, and Conflict), a saber:

a) Fallos de inicialización o arranque (Cold misses). Son los fallos que se producen cada vez que un bloque se memoria se referencia por vez primera y se lleva a la cache. Son fallos que no se pueden evitar en modo alguno y que dependen únicamente del número de bloques referenciados

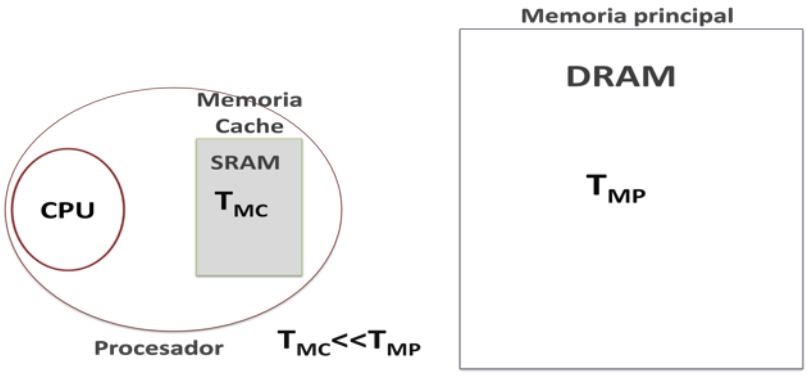
b) Fallos de capacidad (Capacity misses). Son fallos que se producen a consecuencia del tamaño limitado de la cache. Con una cache ideal de tamaño infinito se eliminarían por completo dichos fallos. Estos fallos acontecen cuando se referencia un bloque que estuvo anteriormente en la cache pero que no lo está en ese momento por haber sido reemplazado en algún momento anterior a consecuencia de la capacidad limitada de la cache. Lógicamente, con el incremento del tamaño de la cache se contribuye a reducir (no eliminar) el número de estos fallos.

c) Fallos de conflicto o colisión (Conflict misses). Son fallos que se producen como consecuencia del grado limitado de asociatividad que puede presentar el esquema de correspondencia empleado. Estos fallos acontecen cuando se referencia un bloque que estuvo anteriormente en la cache pero que no lo está en ese momento por haber sido reemplazado en algún momento anterior como consecuencia del conflicto originado con otros bloques que se mapean al mismo conjunto de la cache dado el número limitado de vías del conjunto (grado de asociatividad). A medida que

aumenta el número de vías se van reduciendo estos fallos. En concreto, con una cache completamente asociativa se eliminarían por completo los fallos de conflicto. A menudo, en el caso de una cache con correspondencia directa o asociativa por conjuntos, no es fácil distinguir si un fallo es de conflicto o de capacidad. Para ello bastaría con averiguar si dicho fallo se habría producido en caso de aplicar una correspondencia totalmente asociativa con una cache del mismo tamaño. En caso de no producirse, el fallo sería de conflicto. En caso contrario, sería de capacidad.

10. Prestaciones de la memoria cache

El acceso a memoria principal (T_{MP}) es actualmente lento en comparación al ciclo de reloj al que trabaja la CPU, por lo que tanto la búsqueda de instrucciones como la ejecución de *loads* y *stores* para el acceso a datos penalizan muy seriamente la velocidad de ejecución de los programas, pudiendo llegar hacerlos inoperativos. Para solucionar el grave problema, los arquitectos de computadores propusieron la técnica de memoria cache, cuyo objetivo es reducir significativamente el tiempo de acceso a memoria, llegando a crear la virtualidad, a ojos de la CPU, de que toda la memoria es muy rápida.



La cache consiste en una memoria estática (SRAM) de pequeño tamaño y tiempo de acceso (T_{MC}) muy reducido en comparación al de la memoria principal ($T_{MC}<<T_{MP}$). La idea es que todos los accesos a memoria sean interceptados por la cache, de modo que si el dato accedido se encuentra en la cache (ACIERTO), éste será accedido en tiempo T_{MC}, mientras que en caso contrario (FALLO), éste será accedido en un tiempo T_{MP}, muy superior. Se definen las Tasas de Aciertos y Fallos como sigue:

$$Tasa\ de\ Aciertos\ (H) = \frac{N\acute{u}mero\ de\ ACIERTOS}{Total\ de\ Acessos\ a\ memoria}$$

$$Tasa\ de\ Fallos = 1 - H$$

En consecuencia, el tiempo medio de acceso a memoria corresponderá a la media de los tiempos T_{MC} y T_{MP}, ponderándolos con arreglo a las Tasas de Aciertos y de Fallos:

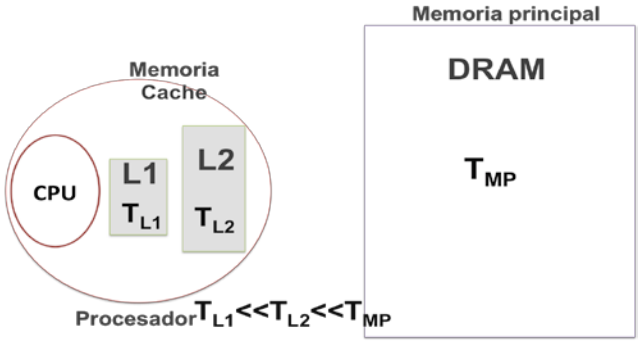
$$Tm = H \times T_{MC} + (1 - H) \times T_{MP}$$

Cuanto mayor sea H, el tiempo medio de acceso a memoria (T_m) estará más próximo al tiempo de acceso a la cache (T_{MC}), que es el objetivo. Actualmente se suelen alcanzar Tasas de Aciertos muy superiores a 0,9 (90%), lo que significa que de cada 100 accesos a memoria, más de 90 son ACIERTOS en la cache.

¿Cómo es posible que se alcancen Tasas de Aciertos tan elevadas teniendo en cuenta el reducidísimo tamaño de la cache en comparación con el de la memoria principal? o lo que es lo mismo ¿cómo es posible encontrar casi siempre en memoria cache la información que se necesita? La respuesta se halla en el principio de localidad tanto espacial como temporal que afortunadamente explotan habitualmente los programas:

> *Principio de localidad espacial:* Al acceder a memoria a la dirección n, existe una elevada probabilidad de que también se acceda a la dirección n+1
>
> *Principio de localidad temporal:* Al acceder a memoria en tiempo t a cierta información, existe una elevada probabilidad de que se vuelva a acceder a dicha información en un tiempo cercano posterior.

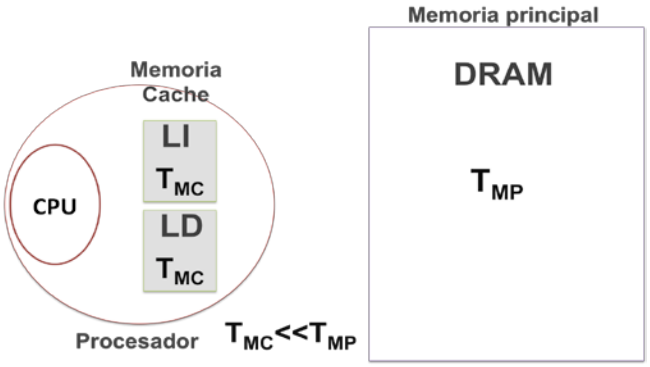
Para mejorar la prestaciones (incrementar la tasa de aciertos y reducir así el tiempo de acceso) sin incrementa el tamaño de la cache (que incrementaría su tiempo de acceso y afectaría negativa a las prestaciones) los procesadores actuales organizan su sistema de cache en varios niveles (L1, L2, L3, …), lo que se conoce como cache multinivel. Cada nivel es de mayor tamaño que el anterior y con un tiempo de acceso mayor. De esta forma, los fallos en L1 podrán ser resueltos muy probablemente en los niveles inferiores (L2, L3, …), evitando la penalización que supone el acceso a memoria principal. Para el cálculo del tiempo medio de acceso a memoria se deberán tener en cuenta las tasas de aciertos de cada nivel y sus tiempos de acceso.



En el caso de dos niveles (L1 y L2) se tomarán en consideración las tasas de aciertos de L1 (H_{L1}) y de L2 (H_{L2}), así como sus tiempos de acceso, T_{L1} y T_{L2}, respectivamente:

$$Tm = H_{L1} \times T_{L1} + (1 - H_{L1}) \times \{H_{L2} \times T_{L2} + (1 - H_{L2}) \times T_{MP}\}$$

Los procesadores actuales emplean *caches segregadas* (o *cache dual*) para Instrucciones (LI) y para Datos (LD) en el nivel L1, y cada vez más también en el nivel L2. Las caches segregadas tienen la ventaja de que, a igualdad de tamaño total, al ser el tamaño de cada una por separado la mitad del tamaño de la cache en la configuración unificada (una única cache tanto para Instrucciones como para Datos), sus tiempos de acceso serán inferiores, lo que contribuirá a reducir el tiempo de medio de acceso a memoria. Otra ventaja radica en la posibilidad de simultanear los accesos a instrucciones y a datos, explotando así el paralelismo en el acceso a memoria.



Suponiendo la existencia de un único nivel de cache L1(I+D), existirán tasas de acierto diferenciadas para Instrucciones (H_I) y para Datos (H_D), de donde se puede derivar una tasa de aciertos promedio (H), resultado de promediar las tasas H_I y H_D, ponderándolas con arreglo a las fracciones de accesos a Instrucciones (%I) y a Datos (%D):

$$H = H_I \times \%I + H_D \times \%D$$

En el cálculo del tiempo medio de acceso a memoria se podrá hacer directamente uso de la tasa de aciertos promedio H calculada previamente:

$$Tm = H \times T_{MC} + (1 - H) \times T_{MP}$$