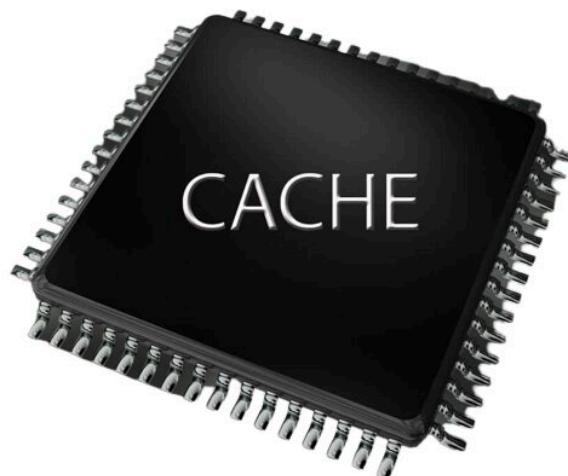


La memoria Caché



1. ¿Qué es la memoria Caché?

La memoria caché es una memoria de acceso rápido que se utiliza para almacenar temporalmente los datos que se usan con mayor frecuencia por el procesador. Está diseñada para reducir el tiempo de acceso a los datos que provienen de la memoria principal (RAM), ya que acceder a la RAM es más lento en comparación con la caché.

La caché se encuentra más cerca del procesador y actúa como una especie de intermedio entre el procesador y la RAM. Los datos que se encuentran en la caché pueden ser recuperados mucho más rápido que si tuvieran que ser leídos desde la memoria principal.

Existen varios niveles de caché, siendo los más comunes L1, L2 y L3, con diferentes tamaños y velocidades:

- Caché L1: Es la más rápida y pequeña, ubicada directamente en el procesador.
- Caché L2: Es más grande y un poco más lenta que L1, pero aún más rápida que la RAM.
- Caché L3: Es la más grande y lenta de todas las cachés, pero sigue siendo más rápida que la memoria principal.

El funcionamiento básico de la memoria caché se basa en aprovechar la "localidad de referencia", que indica que las mismas ubicaciones de memoria tienden a ser accedidas repetidamente en un corto periodo de tiempo (localidad temporal) y que las ubicaciones cercanas a las accedidas recientemente también tienden a ser utilizadas (localidad espacial).

El computador Hack, al ser un diseño con fines educativos, no incluye memoria caché.

2. localidad de referencia

La localidad de referencia es un principio clave que describe el comportamiento típico de los programas al acceder a la memoria, y es crucial para entender por qué la memoria caché es tan efectiva. Este concepto se divide en dos conceptos principales: localidad temporal y localidad espacial.

Localidad Temporal

La localidad temporal se refiere a la tendencia de un programa a acceder repetidamente a los mismos datos o instrucciones en un corto periodo de tiempo. Esto significa que si un dato ha sido accedido recientemente, es probable que se vuelva a acceder a ese mismo dato en un futuro cercano.

Ejemplo práctico: Imagina que un programa realiza una serie de cálculos sobre un mismo conjunto de datos, como una variable que se utiliza varias veces dentro de un bucle. En este caso, los datos de esa variable son accedidos repetidamente, y almacenarlos en la memoria caché permite un acceso mucho más rápido en cada iteración del bucle.

Localidad Espacial

La localidad espacial se refiere a la tendencia de un programa a acceder a datos que están físicamente cercanos entre sí en la memoria. Si un programa accede a una posición de memoria, es probable que pronto acceda a posiciones cercanas.

Ejemplo práctico: Supongamos que un programa está recorriendo un vector de datos. Si se accede a la posición i de ese vector, es muy probable que el siguiente acceso sea a la posición $i+1$. La memoria caché, al almacenar no solo el dato que fue solicitado, sino también un bloque de datos contiguos, mejora el rendimiento de estos accesos secuenciales.

3. Caché de datos vs. de instrucciones

La caché de datos y la caché de instrucciones son dos tipos específicos de memoria caché que almacenan distintos tipos de información que el procesador necesita para su ejecución.

Caché de Instrucciones

- Propósito: Almacena las instrucciones que el procesador probablemente va a ejecutar.
- Función: El objetivo de la caché de instrucciones es minimizar el tiempo de espera cuando el procesador necesita acceder a las instrucciones del programa. En lugar de buscarlas en la memoria principal, puede encontrarlas rápidamente en la caché de instrucciones.
- Uso: Esta caché es útil para programas que tienden a ejecutar secuencias repetitivas de instrucciones, como bucles o funciones llamadas con frecuencia, así como en el funcionamiento normal de un programa donde la siguiente instrucción a ser ejecutada es la almacenada en la posición consecutiva en la memoria RAM con respecto a la actual.

Caché de Datos

- Propósito: Almacena los datos que el procesador es más probable que necesite para realizar operaciones. Estos datos pueden ser variables, matrices, o cualquier valor que el programa manipule.
- Función: La caché de datos reduce el tiempo que tarda el procesador en acceder a los datos, guardando temporalmente aquellos que son más probables de ser reutilizados.
- Uso: Es especialmente útil en programas que manipulan grandes conjuntos de datos o realizan operaciones repetitivas sobre los mismos datos, aprovechando la localidad temporal y espacial de los datos. Por ejemplo, en un cálculo iterativo sobre un vector, la caché de datos almacenará partes de ese vector, de modo que los accesos posteriores sean mucho más rápidos, evitando la lentitud de la RAM.

Caché unificada vs. caché separada

Algunos procesadores utilizan un diseño de caché unificada, donde los datos y las instrucciones comparten el mismo espacio de caché, mientras que otros usan un diseño de caché separada, con una caché para datos y otra para instrucciones. Los procesadores modernos, en muchos casos, tienen cachés separadas en el nivel L1, y una caché unificada para L2 o L3.

4. Jerarquía de memoria

La jerarquía de memoria es una estructura que organiza los diferentes tipos de memoria utilizados en un computador, con el fin de equilibrar tres factores clave: velocidad, capacidad y costo. La idea principal es que las memorias más rápidas y costosas (como los registros y la caché) son más pequeñas y cercanas al procesador, mientras que las memorias más lentas y económicas (como la RAM y el almacenamiento secundario) tienen mayor capacidad y están más alejadas. La Figura 1 muestra la jerarquía de memoria de un computador común.

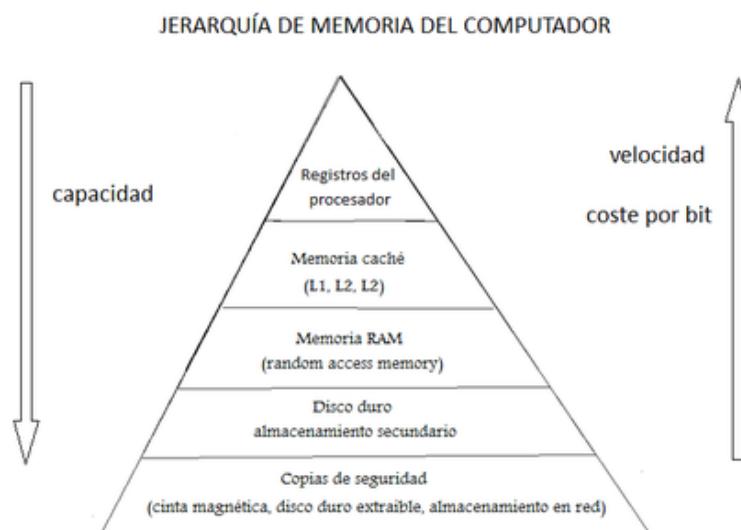


Figura 1: La jerarquía de memoria. Fuente:
https://es.wikipedia.org/wiki/Jerarqu%C3%ADa_de_memoria

La jerarquía de memoria está diseñada para maximizar el rendimiento del sistema, permitiendo que los datos más frecuentemente utilizados estén accesibles rápidamente para el procesador, mientras que los datos menos utilizados se guardan en niveles de memoria más lento pero de mayor capacidad. A medida que los datos suben en la jerarquía (desde el almacenamiento secundario hacia la caché), el sistema se vuelve más eficiente.

Esta jerarquía es fundamental para el diseño de los sistemas modernos porque permite aprovechar la alta velocidad de los procesadores sin estar limitado por la latencia de la memoria más lenta, manteniendo un equilibrio entre costo y rendimiento.

Un símil en el mundo de los videojuegos podría ser el Baldur's Gate 3 (ver Figura 2). En la parte inferior de la pantalla hay un panel y a su derecha hay un pequeño espacio donde el jugador puede poner los hechizos y pociones que es más probable que necesite. Esto sería una memoria en los niveles superiores de la jerarquía, como por ejemplo una memoria caché. Sin embargo, el jugador tiene muchas más pociones y hechizos que puede consultar si accede al inventario. Pero para ello, el jugador tiene que abrir el inventario y “copiar” la poción en el panel de acceso rápido. Esto lleva un tiempo. El inventario se podría equivaler con la memoria RAM. Por último, el jugador puede volver a su campamento donde hay un cofre donde puede guardar todas las pociones y hechizos que no le caben en el inventario. El cofre es muy grande y caben muchas cosas, pero se pierde mucho tiempo cada vez que queremos acceder a él. El símil, en este caso, sería un disco duro.

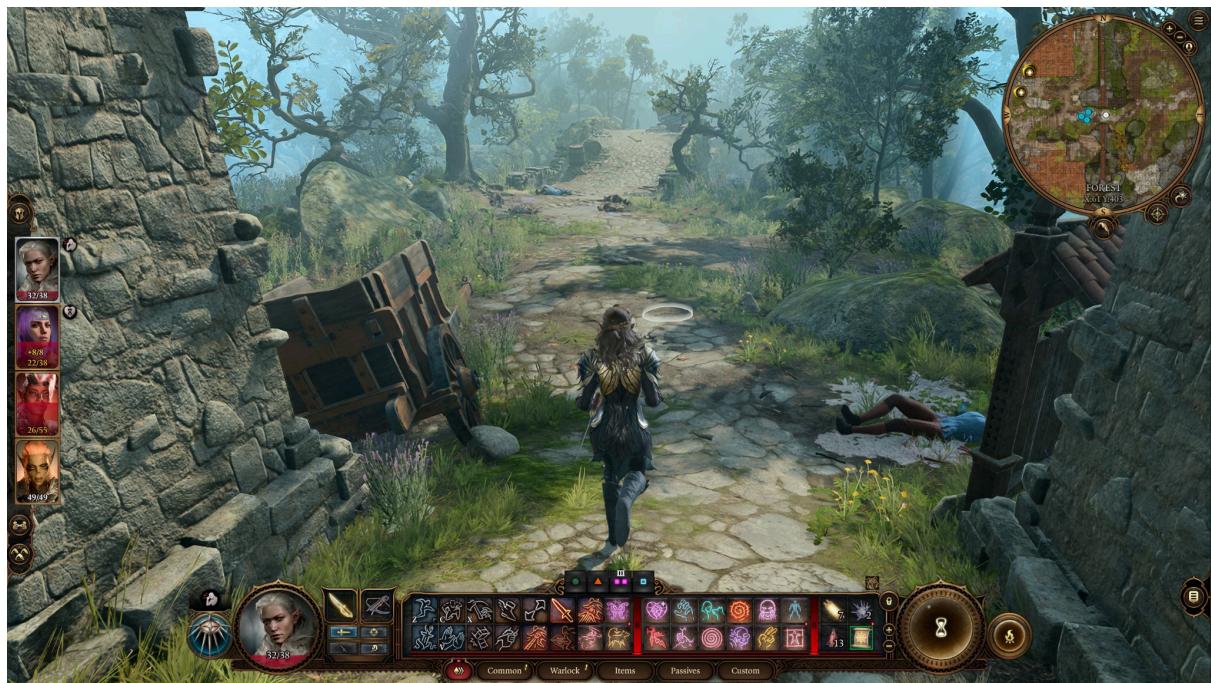


Figura 2: Captura de pantalla del juego Baldur's Gate 3. En la parte inferior derecha se encuentra el acceso rápido a los hechizos y pociones que el jugador puede usar.

¿Cuál es la forma óptima de organizar los hechizos y las pociones? Un jugador inteligente tendría los que es muy probable que necesite en la siguiente batalla en el panel, otros menos probables en el inventario y el resto en el cofre del campamento.

Niveles de la jerarquía de memoria

1. Registros:

- Ubicación: Dentro del procesador.
- Velocidad: Son los más rápidos de todos los tipos de memoria, ya que están directamente integrados en la unidad de procesamiento.
- Capacidad: Muy pequeña (en el orden de unos pocos bytes). En el computador Hack, tienen 16 bits.
- Uso: Los registros almacenan datos inmediatos que la unidad de procesamiento necesita de manera urgente para ejecutar instrucciones. La CPU puede acceder a ellos en un solo ciclo de reloj. En el computador Hack, son los registros D y A.

2. Memoria Caché:

- Ubicación: Cercana al procesador, a menudo integrada en él o en el mismo paquete.
- Velocidad: Más lenta que los registros pero mucho más rápida que la RAM.
- Capacidad: Limitada (en el rango de kilobytes a megabytes, dependiendo del nivel).
- División en niveles:
 - Caché L1: Es la más rápida y pequeña, normalmente específica para cada núcleo del procesador.
 - Caché L2: Un poco más lenta que L1, pero más grande. Puede estar compartida entre varios núcleos en algunos procesadores.
 - Caché L3: Más lenta y grande que L2, generalmente compartida por todos los núcleos del procesador.
- Uso: La caché almacena tanto datos como instrucciones que son frecuentemente accedidos, acelerando el acceso a la información respecto a la RAM.

El computador Hack no tiene implementada la memoria caché.

3. Memoria Principal (RAM):

- Ubicación: Fuera del procesador, pero accesible directamente por la CPU.
- Velocidad: Más lenta que la caché, pero más rápida que los dispositivos de almacenamiento.
- Capacidad: Mucho mayor que la caché (en el orden de gigabytes). En el computador hack, al ser un ejemplo educativo, la memoria RAM tiene únicamente 34Kb.
- Uso: La RAM almacena programas y datos en uso activo que no caben en la caché. La CPU puede acceder directamente a ella, pero con mayor latencia comparada con la caché. Es una memoria volátil, lo que significa que los datos se pierden al apagar el sistema.

El computador Hack tiene implementada una memoria RAM de 32K donde la mitad (16K) se usa para datos. La otra mitad se usa para mapear la E/S de la pantalla y teclado. El computador Hack tiene también una memoria ROM para instrucciones de 32K.

4. Almacenamiento Secundario (Disco Duro o SSD):

- Ubicación: Externo al procesador y a la RAM.
- Velocidad: Mucho más lenta que la RAM y la caché.
- Capacidad: Muy grande (en el orden de terabytes).
- Uso: El almacenamiento secundario se utiliza para guardar datos a largo plazo y programas que no están siendo ejecutados activamente. A menudo, los sistemas operativos utilizan una porción de este almacenamiento como memoria virtual o swap, para expandir la RAM disponible, aunque con una penalización de velocidad considerable.

El computador Hack no tiene disco duro.

5. Almacenamiento Externo (CD, DVD, Blu-ray, USB, etc.):

- Ubicación: Externo al sistema, puede ser removible.
- Velocidad: Mucho más lenta que el almacenamiento secundario.
- Capacidad: Varía ampliamente (desde megabytes hasta terabytes).
- Uso: Se utiliza principalmente para copias de seguridad, transporte de datos, y almacenamiento de archivos a largo plazo.

Características principales de la jerarquía de memoria

- Velocidad decreciente: A medida que nos alejamos del procesador en la jerarquía, la velocidad de acceso a la memoria disminuye. Los registros y la caché son extremadamente rápidos, mientras que el almacenamiento secundario (como un disco duro) es mucho más lento.
- Capacidad creciente: En general, cuanto más baja en la jerarquía se encuentra un nivel de memoria, mayor es su capacidad. Esto permite que grandes cantidades de datos y programas sean almacenados en niveles más bajos (RAM y almacenamiento secundario).
- Costo creciente por bit: La memoria más rápida (como los registros y la caché) es también la más costosa por bit. Por esta razón, los sistemas se diseñan para tener una pequeña cantidad de memoria rápida y costosa cerca del procesador, y una mayor cantidad de memoria más lenta y barata más lejos del procesador.

5. Conceptos relacionados con la memoria caché

Bloque

Un bloque es la unidad básica de almacenamiento en la memoria caché. Cuando se transfiere información entre la caché y la memoria principal, se hace en bloques. El tamaño de un bloque puede variar, pero comúnmente oscila entre 32 y 128 bytes. Un bloque contiene un conjunto de datos que se considera que están relacionados, aprovechando así la localidad espacial.

Conjunto

También llamado línea, es una estructura de datos que se usa para construir la memoria caché. Un conjunto puede tener de 1 o varios bloques (que es lo habitual).

Tasa de Acierto (Hit Rate)

La tasa de acierto es la proporción de accesos a la caché que resultan en un acierto (hit). Es decir, es la fracción de veces que los datos solicitados están presentes en la caché. Una tasa de acierto alta indica que la caché está funcionando de manera efectiva, reduciendo el tiempo de acceso a los datos y aumentando el rendimiento general del sistema.

Tasa de Fallo (Miss Rate)

La tasa de fallo es la proporción de accesos a la caché que resultan en un fallo (miss). Es decir, es la fracción de veces que los datos solicitados no están presentes en la caché y deben ser leídos de la memoria principal. Una tasa de fallo baja es deseable, ya que indica que el procesador tiene acceso rápido a los datos requeridos.

Ciclo de Tiempo de Acierto (Hit Time)

Es el tiempo que tarda el procesador en acceder a un dato que se encuentra en la caché. Un tiempo de acierto bajo es crítico para el rendimiento general del sistema, ya que significa que los datos se pueden recuperar rápidamente sin esperar un acceso más lento a la memoria principal.

Ciclo de Tiempo de Fallo (Miss Penalty)

Es el tiempo adicional que se requiere para recuperar datos de la memoria principal cuando ocurre un fallo en la caché. Un ciclo de tiempo de fallo alto puede afectar significativamente el rendimiento del sistema, ya que implica que el procesador debe esperar más tiempo para obtener los datos necesarios.

6. Caché de correspondencia directa

Las cachés de correspondencia directa (o cachés directamente mapeadas) son una forma simple y eficiente de organizar la memoria caché. Las memorias de correspondencia directa se organizan en varios conjuntos (o líneas de caché) y utilizan un esquema de mapeo directo donde cada bloque de memoria principal se asigna a un único conjunto determinado por unos bits de la dirección de memoria que se desea saber si está o no en la caché. Aunque su diseño es simple y proporciona tiempos de acceso rápidos, pueden enfrentar problemas de conflictos que afectan su rendimiento. Por lo tanto, su implementación es ideal en situaciones donde la localidad de referencia es alta y se minimizan los conflictos.

La caché se divide en varios conjuntos, donde cada uno de los cuales contiene un bloque de datos de la memoria principal y algunos bits de control, como la etiqueta y el bit de validez. La etiqueta (o tag) actúa como identificador y se utiliza para saber si en el bloque se encuentra o no la dirección buscada. El bit de validez indica si el bloque contiene un valor válido (1) o no (0). En términos coloquiales podemos decir que cuando está a 1 el bloque está "lleno" y cuando está a 0, el bloque está "vacío". Al principio de la ejecución del computador, el bit de validez de todos los bloques de la caché está a cero.

Un bloque puede almacenar un dato o varios, siendo esto último lo más común.

Proceso de Acceso a la Caché

1. Descomposición de la Dirección: Cuando el procesador solicita un dato, la dirección de memoria se descompone en etiqueta, índice y posición dentro del bloque.

Por ejemplo, si las direcciones de memoria son de 16 bits, la caché tiene 4 conjuntos de un bloque cada uno y en cada bloque se pueden almacenar 1 dato de 16 bits, la dirección se descompondrá de la siguiente manera (ver Figura 3):

- Etiqueta: bits 15 al 2.
- Índice: bits 1 y 0. Necesitamos 2 bits pues tenemos 4 conjuntos.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	0	1	1	0

Figura 3: Descomposición de una dirección de 16 bits cuando la caché tiene 4 conjuntos y cada bloque puede almacenar 1 palabras.

Si las direcciones de memoria son de 16 bits, la caché tiene 4 conjuntos de un bloque cada uno y en cada bloque se pueden almacenar 2 datos de 16 bits, la dirección se descompondrá de la siguiente manera (ver Figura 4):

- Etiqueta: bits 15 al 3.
- Índice: bits 2 y 1. Necesitamos 2 bits pues tenemos 4 conjuntos.
- Posición dentro del bloque: bit 0. Necesitamos 1 bits pues cada bloque almacena dos datos.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	0	1	1	0

Figura 4: Descomposición de una dirección de 16 bits cuando la caché tiene 4 conjuntos y cada bloque puede almacenar 2 palabras.

Si las direcciones de memoria son de 16 bits, si la caché tiene 8 conjuntos de un bloque cada uno y en cada bloque se pueden almacenar 4 datos de 16 bits, la dirección se descompondrá de la siguiente manera (ver Figura 5):

- Etiqueta: bits 15 al 5.
- Índice: bits 4, 3 y 2. Necesitamos 3 bits pues tenemos 8 conjuntos.
- Posición dentro del bloque: bits 1 y 0. Necesitamos 2 bits pues cada bloque almacena cuatro datos.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	0	1	1	0

Figura 5: Descomposición de una dirección de 16 bits cuando la caché tiene 8 conjuntos y cada bloque puede almacenar 4 palabras.

2. Acceso al conjunto:

- Se utiliza el índice para acceder al conjunto correspondiente.

3. Comparación de la Etiqueta:

- La caché verifica el bit de validez para asegurarse de que la línea contenga un bloque válido. Si no es válido se produce un fallo (miss).
- Si es válido, se compara la etiqueta almacenada en la línea con la etiqueta extraída de la dirección de memoria.
- Si las etiquetas coinciden, se produce un acierto (hit), y el procesador puede acceder directamente al bloque de datos en la caché. Devolverá la palabra indicada por los bits que especifican la palabra dentro del bloque (los más a la derecha de la dirección).
- Si las etiquetas no coinciden, se produce un fallo (miss).

4. Manejo de Fallos:

- En caso de un fallo, el procesador debe buscar el bloque de datos en la memoria principal.
- Una vez que se encuentra el bloque, se carga en el conjunto correspondiente, se actualiza la etiqueta y se establece el bit de validez (si estaba a cero). Entonces se envía a la CPU el valor solicitado. Si el bloque puede almacenar más de un dato, se devolverá el indica por los bits “posición dentro del bloque”.
- Si la línea de caché estaba ocupada (el bit de validez está a 1), se reemplaza el bloque existente.

Ventajas de la Caché de Correspondencia Directa

- Simplicidad: La implementación es simple y fácil de entender. Requiere un hardware menos complejo en comparación con otras organizaciones de caché, como las cachés totalmente asociativas.
- Velocidad: Debido a que solo se necesita acceder a un conjunto de caché, el tiempo de acceso es relativamente rápido.

Desventajas de la Caché de Correspondencia Directa

La principal desventaja es que diferentes bloques de memoria que mapean al mismo conjunto pueden causar conflictos. Esto se conoce como conflicto de caché. Por ejemplo, si dos bloques diferentes necesitan ser almacenados en el mismo conjunto, uno de ellos reemplazará al otro, lo que puede resultar en una tasa de acierto menor.

7. Algoritmo de las cachés de correspondencia directa

A continuación se presenta el funcionamiento de una memoria caché de correspondencia directa en lenguaje algorítmico.

```
INICIO
    [etiqueta, índice, posición dentro bloque] = SegmentarDirección(DIR)
    Si (CACHE.bit_validez[índice] == 0)
        CACHE.bit_validez[índice] = 1
        FALLO
    sino si (CACHE.etiqueta[índice] != etiqueta)
        FALLO
    sino
        ACIERTO
FIN

ACIERTO
    Devolver CACHE.datos[posición dentro bloque]

FALLO
    Ir a la memoria RAM y traer el bloque completo
    Copiar el bloque en CACHE.datos
    Devolver CACHE.datos[posición dentro bloque]
```

8. Simulador memoria caché

En el repositorio <https://github.com/montoliu/SimularMemoriaCache> se encuentra un simulador implementado en Python que permite simular el funcionamiento de una memoria caché.

9. Ejemplos de funcionamiento de la memoria caché por correspondencia directa

Ejemplo 1

Se asume una caché de 4 líneas, donde cada línea puede almacenar un bloque que contiene un único dato de 16 bits.

Cada dirección de memoria es de 16 bits.

En este caso, el formato de la dirección se divide de la siguiente manera (ver Figura 3):

- **Tag:** los 14 bits más significativos (bits 15 a 2 de la dirección).
- **Índice:** los 2 bits siguientes (bits 1 y 0 de la dirección).

La caché comienza con todos los bloques con el bit de validez a 0.

Direcciones de memoria solicitadas por la CPU:

1. 0000 0000 0000 0011
2. 0000 0000 0000 0110
3. 0000 0000 0000 0001
4. 0000 0000 0000 0011

Asumimos que la memoria tiene las siguientes direcciones y datos almacenados:

Dirección	Contenido
0000 0000 0000 0001	1001 0110 1110 1011
0000 0000 0000 0011	1101 0011 1010 0110
0000 0000 0000 0110	0011 1100 1010 1011
0000 0000 0000 1111	1001 0110 0000 0000

Estado inicial de la caché:

Conjunto	Bit de validez	Tag	Datos
00	0	-	-
01	0	-	-
10	0	-	-
11	0	-	-

Dirección: 0000 0000 0000 0011

- **Tag:** 0000 0000 0000 00
- **Índice:** 11

Se produce un fallo pues en el único bloque del conjunto 11 el bit de validez está a cero. Por lo tanto, hay que ir a la memoria RAM y traer el dato almacenado en la dirección 0000 0000 0000 0011. Ese dato (1101 0011 1010 0110) se almacenará en el campo datos del bloque del conjunto 11 de la caché y se devolverá a la CPU.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
00	0	-	

01	0	-	-
10	0	-	-
11	1	0000 0000 0000 00	1101 0011 1010 0110

Dirección: 0000 0000 0000 0110

- **Tag:** 0000 0000 0000 01
- **Índice:** 10

Se produce un fallo pues en el bloque del conjunto 10 el bit de validez está a cero. Por lo tanto, hay que ir a la memoria RAM y traer el dato almacenado en la dirección 0000 0000 0000 0110. Ese dato (0011 1100 1010 1011) se almacenará en el campo datos del bloque del conjunto 10 de la caché y se devolverá a la CPU.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
00	0	-	
01	0	-	-
10	1	0000 0000 0000 01	0011 1100 1010 1011
11	1	0000 0000 0000 00	1101 0011 1010 0110

Dirección: 0000 0000 0000 0001

- **Tag:** 0000 0000 0000 00
- **Índice:** 01

Se produce un fallo pues en el bloque del conjunto 01 el bit de validez está a cero. Por lo tanto, hay que ir a la memoria RAM y traer el dato almacenado en la dirección 0000 0000 0000 0001. Ese dato (1001 0110 1110 1011) se almacenará en el campo datos del bloque del conjunto 01 de la caché y se devolverá a la CPU.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
00	0	-	
01	1	0000 0000 0000 00	1001 0110 1110 1011
10	1	0000 0000 0000 01	0011 1100 1010 1011

11	1	0000 0000 0000 00	1101 0011 1010 0110
----	---	-------------------	---------------------

Dirección: 0000 0000 0000 0011

- **Tag:** 0000 0000 0000 00
- **Índice:** 11

En este caso se produce un acierto, pues en el bloque del conjunto 11 el bit de validez está a uno y el tag de la dirección solicitada (0000 0000 0000 00) coincide con el tag de la línea 11 de la caché. Se devolverá a la RAM el dato almacenado en el campo datos del bloque del conjunto 11.

La caché no cambia.

Dirección: 0000 0000 0000 1111

- **Tag:** 0000 0000 0000 11
- **Índice:** 11

En este caso se produce un fallo pues, aunque el bit de validez del bloque del conjunto 11 de la caché está a 1, la etiqueta de la dirección solicitada (0000 0000 0000 11) no coincide con la almacenada en el bloque del conjunto 11. Por lo tanto, hay que ir a la memoria RAM y traer el dato almacenado en la dirección 0000 0000 0000 1111. Ese dato (1001 0110 0000 0000) se almacenará en el campo datos del bloque del conjunto 11 de la caché y se devolverá a la CPU.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
00	0	-	
01	1	0000 0000 0000 00	1001 0110 1110 1011
10	1	0000 0000 0000 01	0011 1100 1010 1011
11	1	0000 0000 0000 11	1001 0110 0000 0000

Ejemplo 2

Se asume una caché de 4 líneas, donde cada conjunto puede almacenar un bloque que contiene dos datos de 16 bits.

Cada dirección de memoria es de 16 bits.

En este caso, el formato de la dirección se divide de la siguiente manera (ver Figura 43):

- **Tag:** los 13 bits más significativos (bits 15 a 3 de la dirección).
- **Índice:** los 2 bits siguientes (bits 2 y 1 de la dirección).

- **Posición dentro del bloque:** el bit menos significativo (bit 0).

Direcciones de memoria solicitadas por la CPU:

1. 0000 0000 0000 0000
2. 0000 0000 0000 0001
3. 0000 0000 0000 0111
4. 0000 0000 0000 0110

Asumimos que la memoria tiene las siguientes direcciones y datos almacenados:

Dirección	Contenido
0000 0000 0000 0000	1101 0011 1010 0110
0000 0000 0000 0001	0011 1100 1010 1011
0000 0000 0000 0110	1001 0110 1110 1011
0000 0000 0000 0111	1001 0110 0000 0000
0000 0000 0000 1100	1001 0110 0000 0000
0000 0000 0000 1101	1001 0110 0000 0011

Estado inicial de la caché:

Conjunto	Bit de validez	Tag	Datos
00	0	-	-
01	0	-	-
10	0	-	-
11	0	-	-

Dirección: 0000 0000 0000 0000

- **Tag:** 0000 0000 0000 0
- **Índice:** 00
- **Posición dentro del bloque:** 0

En este caso se produce un fallo pues el bit de validez del bloque del conjunto 00 está a cero. Por lo tanto, hay que ir a la memoria RAM y traer los datos almacenados en las direcciones 0000 0000 0000 0000 y su consecutiva 0000 0000 0000 0001. Este bloque de dos datos (1101 0011 1010 0110 y 0011 1100 1010 1011) se almacenarán en el campo datos del bloque del conjunto 00 de la caché y se devolverá a la CPU el primero de ellos (1101 0011 1010 0110) pues la posición dentro del bloque está a cero.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
00	1	0000 0000 0000 0	0 -> 1101 0011 1010 0110 1 -> 0011 1100 1010 1011
01	0	-	-
10	0	-	-
11	0	-	-

Dirección: 0000 0000 0000 0111

- **Tag:** 0000 0000 0000 0
- **Índice:** 11
- **Posición dentro del bloque:** 1

En este caso se produce un acierto pues el bit de validez del bloque del conjunto 00 está a uno y el tag de la dirección solicitada (0000 0000 0000 0) coincide con el tag del bloque del conjunto 00. Se devolverá el segundo dato, pues la posición dentro del bloque es 1, es decir, se devolverá el dato 0011 1100 1010 1011.

La caché no cambiará.

Dirección: 0000 0000 0000 0001

- **Tag:** 0000 0000 0000 0
- **Índice:** 00
- **Posición dentro del bloque:** 1

En este caso se produce un fallo pues el bit de validez del bloque del conjunto 11 está a cero. Por lo tanto, hay que ir a la memoria RAM y traer los datos almacenados en las direcciones 0000 0000 0000 0110 y su consecutiva 0000 0000 0000 0111. Este bloque de dos datos (1001 0110 1110 1011 y 1001 0110 0000 0000) se almacenarán en el campo datos del bloque del conjunto 11 de la caché y se devolverá a la CPU el segundo de ellos (1001 0110 0000 0000) pues posición dentro del bloque está a uno.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
00	1	0000 0000 0000 0	0 -> 1101 0011 1010 0110 1 -> 0011 1100 1010 1011

01	0	-	-
10	0	-	-
11	1	0000 0000 0000 0	0 -> 1001 0110 1110 1011 1 -> 1001 0110 0000 0000

Dirección: 0000 0000 0000 0110

- **Tag:** 0000 0000 0000 0
- **Índice:** 11
- **Posición dentro del bloque:** 0

En este caso se produce un acierto pues el bit de validez del bloque del conjunto 11 está a uno y el tag de la dirección solicitada (0000 0000 0000 0) coincide con el tag del bloque del conjunto 11. Se devolverá el primer dato, pues la posición dentro del bloque es 0, es decir, se devolverá el dato 1001 0110 1110 1011.

La caché no cambiará.

Ejemplo 3

Se asume una caché de 4 líneas, donde cada línea puede almacenar un bloque que contiene cuatro datos de 16 bits.

Cada dirección de memoria es de 16 bits.

En este caso, el formato de la dirección se divide de la siguiente manera:

- **Tag:** los 12 bits más significativos (bits 15 a 4 de la dirección).
- **Índice:** los 2 bits siguientes (bits 3 y 2 de la dirección).
- **Posición dentro del bloque:** los dos bits menos significativos (bits 1 y 0).

Direcciones de memoria solicitadas por la CPU:

1. 0000 0000 0000 0000
2. 0000 0000 0000 0001
3. 0000 0000 0000 0011

Asumimos que la memoria tiene las siguientes direcciones y datos almacenados:

Dirección	Contenido
0000 0000 0000 0000	1101 0011 1010 0110
0000 0000 0000 0001	0011 1100 1010 1011
0000 0000 0000 0010	1001 0110 1110 1011

0000 0000 0000 0011	1001 0110 0000 0000
0000 0000 0000 1100	1001 0110 0000 0000
0000 0000 0000 1101	1001 0110 0000 0011

Estado inicial de la caché:

Conjunto	Bit de validez	Tag	Datos
00	0	-	-
01	0	-	-
10	0	-	-
11	0	-	-

Dirección: 0000 0000 0000 0000

- **Tag:** 0000 0000 0000
- **Índice:** 00
- **Posición dentro del bloque:** 00

En este caso se produce un fallo pues el bit de validez del bloque del conjunto 00 está a cero. Por lo tanto, hay que ir a la memoria RAM y traer los datos almacenados en las direcciones 0000 0000 0000 0000 y sus consecutivas 0000 0000 0000 0001, 0000 0000 0000 0010 y 0000 0000 0000 0011. Este bloque de cuatro datos se almacenarán en el campo datos del bloque del conjunto 00 de la caché y se devolverá a la CPU el primero de ellos (1101 0011 1010 0110) pues posición dentro del bloque está a 00.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
00	1	0000 0000 0000 0	00 -> 1101 0011 1010 0110 01 -> 0011 1100 1010 1011 10 -> 1001 0110 1110 1011 11 -> 1001 0110 0000 0000
01	0	-	-
10	0	-	-
11	0	-	-

Dirección: 0000 0000 0000 0001

- **Tag:** 0000 0000 0000
- **Índice:** 00
- **Posición dentro del bloque:** 01

En este caso se produce un acierto pues el bit de validez del bloque del conjunto 00 está a uno y el tag de la dirección solicitada (0000 0000 0000) coincide con el tag del bloque del conjunto 00. Se devolverá el segundo dato, pues la posición dentro del bloque es 01, es decir, se devolverá el dato 0011 1100 1010 1011.

La caché no cambiará.

Dirección: 0000 0000 0000 0011

- **Tag:** 0000 0000 0000
- **Índice:** 00
- **Posición dentro del bloque:** 11

En este caso se produce un acierto pues el bit de validez del bloque del conjunto 00 está a uno y el tag de la dirección solicitada (0000 0000 0000) coincide con el tag del bloque del conjunto 00. Se devolverá el cuarto dato, pues la posición dentro del bloque es 11, es decir, se devolverá el dato 1001 0110 0000 0000.

La caché no cambiará.

10. Escrituras

En una caché de correspondencia directa, la gestión de las escrituras puede implementarse usando dos políticas principales: escritura directa (write-through) o escritura diferida (write-back).

Escritura directa (write-through):

En esta política, cada vez que se realiza una escritura en la caché, el dato se escribe simultáneamente en la caché y en la memoria principal (RAM). Esto garantiza que la memoria principal esté siempre sincronizada con la caché, pero puede tener un impacto en el rendimiento debido a la frecuencia con la que se accede a la RAM.

- Ventajas:
 1. La memoria principal y la caché están siempre en coherencia, es decir, los datos son consistentes en ambas.
 2. Es más sencilla de implementar y no requiere un mecanismo complejo para manejar los datos sucios (dirty blocks).
- Desventajas:

1. Cada escritura implica acceso a la RAM, lo que puede ser costoso en términos de tiempo, ya que las escrituras a la memoria principal son más lentas que las lecturas/escrituras a la caché.

Escritura diferida (write-back):

Con esta política, las escrituras se realizan solo en la caché cuando hay un acceso de escritura. El dato en la caché se marca como "sucio" (dirty bit), lo que significa que es diferente al que está almacenado en la memoria principal. Solo cuando el bloque es reemplazado, el dato sucio se escribe de vuelta en la memoria principal.

- Ventajas:
 1. Se reduce el número de accesos a la memoria principal, lo que mejora el rendimiento, ya que las escrituras a RAM solo ocurren cuando se reemplaza un bloque sucio.
- Desventajas:
 1. La memoria principal puede no estar actualizada y estar en un estado inconsistente con respecto a la caché hasta que se escriba el bloque.
 2. Requiere un bit de validez adicional, llamado "bit sucio" (dirty bit), para marcar los bloques que han sido modificados.

11. Caché totalmente asociativa

Las memorias caché de correspondencia directa son simples y rápidas, pero tienen algunos problemas que pueden afectar su rendimiento. Estos son algunos de los principales:

- Conflictos de mapeo (colisiones): En una caché de correspondencia directa, cada dirección de memoria principal tiene un único conjunto asignado. Si varias direcciones de memoria se mapean al mismo conjunto, ocurre una colisión. Cuando eso sucede, la caché debe reemplazar el dato anterior, incluso si ese dato aún es útil, lo que puede causar una alta tasa de fallos.
- Baja flexibilidad en la reutilización de datos: Debido a la rigidez del mapeo, los datos que podrían seguir siendo útiles pueden ser reemplazados con demasiada frecuencia. Esto ocurre especialmente en aplicaciones con patrones de acceso a la memoria que frecuentemente usan diferentes direcciones que se mapean al mismo conjunto, pero que podrían beneficiarse de estar en diferentes ubicaciones de caché.
- Bajo aprovechamiento del espacio de la caché: A veces, solo una pequeña parte de la caché se utiliza eficientemente, ya que las colisiones pueden provocar que algunos conjuntos se sobreescrbían constantemente, mientras otros conjuntos de la caché permanecen sin usar.

Las memorias caché totalmente asociativas surgen como una solución para los problemas de la correspondencia directa, especialmente los conflictos de mapeo. En una caché totalmente asociativa, cualquier bloque de memoria principal puede almacenarse en cualquier bloque de la caché,

proporcionando mucha más flexibilidad en la colocación de datos. Podemos decir que si las caches de correspondencia directa tienen n conjuntos, con n mayor que 1, y en cada conjunto hay un único bloque, entonces en las cachés totalmente asociativas tenemos un único conjunto y en él se almacenan n bloques (n mayor que 1).

En una caché totalmente asociativa:

- No hay un mapeo fijo entre una dirección de memoria y un bloque de caché específico. Cualquier bloque de la memoria principal puede almacenarse en cualquier bloque.
- Para buscar un dato, la dirección de memoria se compara con los tags (etiquetas) de todos los bloques al mismo tiempo (paralelamente). Si hay una coincidencia en algún tag, se realiza un acierto de caché y el dato es extraído de ese bloque.

Las cachés totalmente asociativas, aunque ofrecen flexibilidad al permitir que cualquier bloque de memoria pueda almacenarse en cualquier bloque de la caché, presentan varias desventajas:

- Mayor complejidad en el hardware: Dado que cualquier bloque de memoria puede estar en cualquier bloque de la caché, el comparador debe verificar simultáneamente todas las líneas de la caché para encontrar un bloque (búsqueda paralela). Esto requiere una mayor cantidad de circuitos comparadores, lo que aumenta la complejidad del hardware.
- Tiempo de acceso más lento: Debido a que se necesita comparar el tag de todos los bloques de la caché simultáneamente, el tiempo de acceso puede ser más largo en comparación con otras configuraciones.
- Política de reemplazo más compleja: En una caché totalmente asociativa, se debe elegir qué línea reemplazar cuando la caché está llena. Esto suele requerir políticas de reemplazo más sofisticadas, como LRU (Least Recently Used), que a su vez implican almacenamiento y lógica adicionales para rastrear el uso de cada bloque.

12. Algoritmo caché totalmente asociativa

A continuación se presenta el funcionamiento de una caché totalmente asociativa en forma de algoritmo. Aunque en realidad la comparación con todos los tag se realiza al mismo tiempo, por simplicidad, vamos a suponer que se hace de forma secuencial. Es importante darse cuenta de que no hace falta el campo índice pues todas las direcciones van al único bloque.

INICIO

```
[etiqueta, posición dentro bloque] = SegmentarDirección (DIR)
PARA (i=0; i<numero de bloques; i++)
    SI (CACHE.bloque[i].bit_validez == 0)
        CACHE.bloque[i].bit_validez = 1
    FALLO
    break
SINO SI (CACHE.bloque[i].etiqueta == etiqueta)
```

```

ACIERTO
break
FINSI
FIN PARA
SI (i == numero de bloques)
    FALLO CON REEMPLAZO
FINSI
FIN

ACIERTO
    Devolver CACHE.bloque[i].datos[posición dentro bloque]

FALLO
    Ir a la memoria RAM y traer el bloque completo
    Copiar el bloque en CACHE.bloque[i].datos
    Devolver CACHE.bloque[i].datos[posición dentro bloque]

FALLO CON REEMPLAZO
- Política de reemplazo LRU.
- j es el bloque que hace más tiempo que ha sido usado.

    Ir a la memoria RAM y traer el bloque completo
    Copiar el bloque en CACHE.bloque[j].datos
    Devolver CACHE.bloque[j].datos[posición dentro bloque]

```

13. Ejemplos de funcionamiento de la memoria caché totalmente asociativa

Ejemplo 1

Se asume una caché de un único conjunto con 4 bloques que contienen, cada uno, un único dato de 16 bits.

Cada dirección de memoria es de 16 bits.

En este caso, el formato de la dirección se divide de la siguiente manera:

- **Tag:** los 16 bits de la dirección.

La caché comienza con el bit de validez de todos los bloques a cero.

Direcciones de memoria solicitadas por la CPU:

1. 0000 0000 0000 0011
2. 0000 0000 0000 0110
3. 0000 0000 0000 0001

4. 0000 0000 0000 1111

5. 0000 0000 0000 0111

Asumimos que la memoria tiene las siguientes direcciones y datos almacenados:

Dirección	Contenido
0000 0000 0000 0001	1001 0110 1110 1011
0000 0000 0000 0011	1101 0011 1010 0110
0000 0000 0000 0110	0011 1100 1010 1011
0000 0000 0000 1111	1001 0110 0000 0000
0000 0000 0000 0111	1101 0011 1010 0000

Estado inicial de la caché:

Conjunto	Bit de validez	Tag	Datos
0	0	-	-
	0	-	-
	0	-	-
	0	-	-

Dirección: 0000 0000 0000 0011

- **Tag:** 0000 0000 0000 0011

Como el bit de validez del primer bloque está a cero, se produce un fallo. Por lo tanto, hay que ir a la memoria RAM y traer el dato almacenado en la dirección 0000 0000 0000 0011. Ese dato (1101 0011 1010 0110) se almacenará en el campo datos del primer bloque y se devolverá a la CPU.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
0	1	0000 0000 0000 0011	1101 0011 1010 0110
	0	-	-
	0	-	-
	0	-	-

Dirección: 0000 0000 0000 0110

- **Tag:** 0000 0000 0000 0110

En primer lugar, comprobamos si el tag del primer bloque se corresponde con el de la dirección. Como no es el caso, se pasa al segundo bloque que como tiene el bit de validez a cero, se produce un fallo. Por lo tanto, hay que ir a la memoria RAM y traer el dato almacenado en la dirección 0000 0000 0000 0110. Ese dato (0011 1100 1010 1011) se almacenará en el campo datos del segundo bloque y se devolverá a la CPU.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
	1	0000 0000 0000 0011	1101 0011 1010 0110
	1	0000 0000 0000 0110	0011 1100 1010 1011
	0	-	-
	0	-	-

Dirección: 0000 0000 0000 0001

- **Tag:** 0000 0000 0000 0001

En primer lugar, comprobamos si el tag del primer bloque se corresponde con el de la dirección. Como no es el caso, se pasa al segundo bloque y como tampoco coinciden los tags, se pasa al siguiente. Como tiene el bit de validez a cero, se produce un fallo. Por lo tanto, hay que ir a la memoria RAM y traer el dato almacenado en la dirección 0000 0000 0000 0001. Ese dato (1001 0110 1110 1011) se almacenará en el campo datos del tercer bloque y se devolverá a la CPU.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
	1	0000 0000 0000 0011	1101 0011 1010 0110
	1	0000 0000 0000 0110	0011 1100 1010 1011
	1	0000 0000 0000 0001	1001 0110 1110 1011
	0	-	-

Dirección: 0000 0000 0000 1111

- **Tag:** 0000 0000 0000 1111

Se comprueba de forma secuencial y en los tres primeros bloques no coincide el tag, por lo que se pasa a comprobar el cuarto. Como tiene el bit de validez a cero, se produce un fallo. Por lo tanto, hay

que ir a la memoria RAM y traer el dato almacenado en la dirección 0000 0000 0000 1111. Ese dato (1001 0110 0000 0000) se almacenará en el campo datos del cuarto bloque y se devolverá a la CPU.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
	1	0000 0000 0000 0011	1101 0011 1010 0110
	1	0000 0000 0000 0110	0011 1100 1010 1011
	1	0000 0000 0000 0001	1001 0110 1110 1011
	1	0000 0000 0000 1111	1001 0110 0000 0000

Dirección: 0000 0000 0000 0111

- **Tag:** 0000 0000 0000 0111

Ocurre que en todos los bloques el bit de validez está a uno y no coincide el tag en todos los casos. Hemos de elegir un bloque ha ser reemplazado por el nuevo.

Existen diferentes estrategias (o políticas) para realizar el reemplazo. La más común es seleccionar el bloque que hace más tiempo que fue accedido (LRU). En este caso será el primero.

Por lo tanto, hay que ir a la memoria RAM y traer el dato almacenado en la dirección 0000 0000 0000 0111. Ese dato (1101 0011 1010 0000) se almacenará en el campo datos del primer bloque y se devolverá a la CPU.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos
	1	0000 0000 0000 0111	1101 0011 1010 0000
	1	0000 0000 0000 0110	0011 1100 1010 1011
	1	0000 0000 0000 0001	1001 0110 1110 1011
	1	0000 0000 0000 1111	1001 0110 0000 0000

Esta política de reemplazo añade complejidad a la memoria caché siendo necesario incluir un campo nuevo a cada bloque que indique el orden en que se deben reemplazar.

La caché, con el nuevo campo, será como sigue:

Conjunto	Bit de validez	Tag	Datos	orden

	1	0000 0000 0000 0111	1101 0011 1010 0000	4
	1	0000 0000 0000 0110	0011 1100 1010 1011	1
	1	0000 0000 0000 0001	1001 0110 1110 1011	2
	1	0000 0000 0000 1111	1001 0110 0000 0000	3

El campo orden indica cual es el bloque que hace más tiempo que ha sido accedido que se corresponde con el valor más pequeño. En el ejemplo, el siguiente bloque a ser reemplazado es el segundo, por ser el que más tiempo hace que ha sido accedido.

Dirección: 0000 0000 0000 0110

- **Tag:** 00000 0000 0000 0110

El tag coincide con el tag del segundo bloque por lo que se produce un acierto y se devuelve el dato 0011 1100 1010 1011. Como ese bloque se acaba de actualizar, pasa a ser el más reciente. Es decir el campo orden se pone a 4 y se resta 1 del resto de bloques.

La caché quedará:

Conjunto	Bit de validez	Tag	Datos	orden
	1	0000 0000 0000 0111	1101 0011 1010 0000	3
	1	0000 0000 0000 0110	0011 1100 1010 1011	4
	1	0000 0000 0000 0001	1001 0110 1110 1011	1
	1	0000 0000 0000 1111	1001 0110 0000 0000	2

Ahora el que se tendrá que reemplazar, si es necesario, será el tercer bloque.

14. Caché asociativa por conjuntos

Las caches de correspondencia directa y las totalmente asociativas tienen sus ventajas y desventajas. Las cachés asociativas por conjunto son una solución intermedia entre los dos extremos. En este tipo de cachés existen varios conjuntos (como en correspondencia directa), pero en cada conjunto tiene varios bloques (como en las totalmente asociativas).

Como en los casos anteriores el primer paso consiste en segmentar la dirección. Entonces, el índice nos dirá en qué conjunto tenemos que mirar (como en correspondencia directa) y dentro del conjunto buscaremos entre los diversos bloques (como en totalmente asociativa).

Este tipo de cachés son las que se usan en la realidad.

15. Algoritmo caché asociativa por conjuntos

A continuación se presenta el funcionamiento de una caché asociativa por conjuntos en forma de algoritmo. Aunque en realidad la comparación con todos los tag del conjunto se realiza al mismo tiempo, por simplicidad, vamos a suponer que se hace de forma secuencial.

INICIO

```
[etiqueta, índice, posición dentro bloque] = SegmentarDirección (DIR)
PARA (i=0; i<número de bloques; i++)
    SI (CACHE.conjunto[índice].bloque[i].bit_validez == 0)
        CACHE.conjunto[índice].bloque[i].bit_validez = 1
    FALLO
    break
SINO SI (CACHE.conjunto[índice].bloque[i].etiqueta == etiqueta)
    ACIERTO
    break
    FINSI
FIN PARA
SI (i == número de bloques)
    FALLO CON REEMPLAZO
    FINSI
ACIERTO
    Devolver CACHE.conjunto[índice].bloque[i].datos[posición dentro bloque]

FALLO
    Ir a la memoria RAM y traer el bloque completo
    Copiar el bloque en CACHE.conjunto[índice].bloque[i].datos
    Devolver CACHE.conjunto[índice].bloque[i].datos[posición dentro bloque]

FALLO CON REEMPLAZO
    - con política de reemplazo LRU.
    - j es el bloque que hace más tiempo que ha sido usado.

    Ir a la memoria RAM y traer el bloque completo
    Copiar el bloque en CACHE.conjunto[índice].bloque[j].datos
    Devolver CACHE.conjunto[índice].bloque[j].datos[posición dentro bloque]
```

16. Ejemplos de caché asociativa por conjuntos

Se asume una caché con cuatro conjuntos, cada uno con 4 bloques que contienen, cada uno, dos datos de 16 bits.

Cada dirección de memoria es de 16 bits.

En este caso, el formato de la dirección se divide de la siguiente manera:

- **Tag:** bits 15 al 3.
- **índice:** bits 2 y 1
- **posición dentro del bloque:** bit 0.

La caché comienza con el bit de validez de todos los bloques a cero.

Direcciones de memoria solicitadas por la CPU:

6. 0000 0000 0000 0011
7. 0000 0000 0000 0110
8. 0000 0000 0000 0010
9. 0000 0000 0000 1111
10. 0000 0000 0000 0111

Asumimos que la memoria tiene las siguientes direcciones y datos almacenados:

Dirección	Contenido
0000 0000 0000 0000	1001 0110 1110 1000
0000 0000 0000 0001	1001 0110 1110 1011
0000 0000 0000 0010	1101 0011 1010 0000
0000 0000 0000 0011	1101 0011 1010 0110
0000 0000 0000 0110	0011 1100 1010 1011
0000 0000 0000 0111	1101 0011 1010 0000
0000 0000 0000 1110	1001 0110 0000 0000
0000 0000 0000 1111	1001 0110 0000 0011

Estado inicial de la caché:

Conjunto	Bit de validez	Tag	Datos	orden
00	0	-	-	
	0	-	-	
	0	-	-	
	0	-	-	
01	0		-	
	0	-	-	

	0	-	-	
	0	-	-	
10	0	-	-	
	0	-	-	
	0	-	-	
	0	-	-	
11	0	-	-	
	0	-	-	
	0	-	-	
	0	-	-	

Dirección: 0000 0000 0000 0011

- **Tag:** 0000 0000 0000 0
- **índice:** 01
- **posición dentro del bloque:** 1.

Buscamos en el conjunto 01. Como el primer bloque de dicho conjunto tiene el bit de validez a cero, se produce un fallo. Vamos a la RAM y traemos un bloque de dos datos que se corresponde con las direcciones 0000 0000 0000 0010 y 0000 0000 0000 0011. Se copian en la caché en el primer bloque del conjunto 01 y se devuelve el segundo de ellos, es decir el valor 1101 0011 1010 0110.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos	orden
00	0	-	-	
	0	-	-	
	0	-	-	
	0	-	-	
01	1	0000 0000 0000 0	0 -> 1101 0011 1010 0000 1 -> 1101 0011 1010 0110	4
	0	-	-	
	0	-	-	
	0	-	-	

10	0	-	-	
	0	-	-	
	0	-	-	
	0	-	-	
11	0	-	-	
	0	-	-	
	0	-	-	
	0	-	-	

Dirección: 0000 0000 0000 0110

- **Tag:** 0000 0000 0000 0
- **índice:** 11
- **posición dentro del bloque:** 0.

Buscamos en el conjunto 11. Como el primer bloque de dicho conjunto tiene el bit de validez a cero, se produce un fallo. Vamos a la RAM y traemos un bloque de dos datos que se corresponde con las direcciones 0000 0000 0000 0110 y 0000 0000 0000 0111. Se copian en la caché en el primer bloque del conjunto 11 y se devuelve el primero de ellos, es decir el valor 0011 1100 1010 1011.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos	orden
00	0	-	-	
	0	-	-	
	0	-	-	
	0	-	-	
01	1	0000 0000 0000 0	0 -> 1101 0011 1010 0000 1 -> 1101 0011 1010 0110	4
	0	-	-	
	0	-	-	
	0	-	-	
10	0	-	-	
	0	-	-	

	0	-	-	
	0	-	-	
11	1	0000 0000 0000 0	0 -> 0011 1100 1010 1011 1 -> 1101 0011 1010 0000	4
	0	-	-	
	0	-	-	
	0	-	-	

Dirección: 0000 0000 0000 0010

- **Tag:** 0000 0000 0000 0
- **índice:** 01
- **posición dentro del bloque:** 0.

Buscamos en el conjunto 01. Como el primer bloque de dicho conjunto tiene un tag que coincide con el de la dirección, se produce un acierto y se devuelve el dato en la posición cero: 1101 0011 1010 0000.

La caché se queda igual.

Dirección: 0000 0000 0000 1111

- **Tag:** 0000 0000 0000 1
- **índice:** 11
- **posición dentro del bloque:** 1.

Buscamos en el conjunto 11. El primer bloque tiene el bit de validez a uno pero los tags no coinciden. Pasamos al segundo bloque tiene el bit de validez a cero, por lo que se produce un fallo. Vamos a la RAM y traemos un bloque de dos datos que se corresponde con las direcciones 0000 0000 0000 1110 y 0000 0000 0000 1111. Se copian en la caché en el segundo bloque del conjunto 11 y se devuelve el segundo de ellos, es decir el valor 1001 0110 0000 0011.

La caché quedará tras este paso:

Conjunto	Bit de validez	Tag	Datos	orden
00	0	-	-	
	0	-	-	
	0	-	-	
	0	-	-	

01	1	0000 0000 0000 0	0 -> 1101 0011 1010 0000 1 -> 1101 0011 1010 0110	4
	0	-	-	
	0	-	-	
	0	-	-	
10	0	-	-	
	0	-	-	
	0	-	-	
	0	-	-	
11	1	0000 0000 0000 0	0 -> 0011 1100 1010 1011 1 -> 1101 0011 1010 0000	3
	1	0000 0000 0000 1	0-> 1001 0110 0000 0000 1 -> 1001 0110 0000 0011	4
	0	-	-	
	0	-	-	