Universidad
Europea de Madrid

LAUREATE INTERNATIONAL UNIVERSITIES

MEMORIA

MEMORIA CACHÉ I

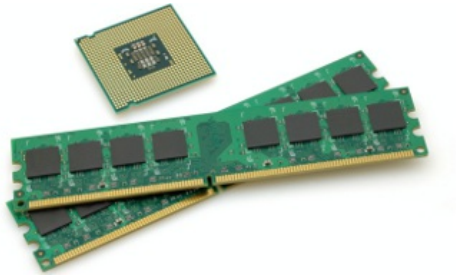
Índice

Presentación

En este tema, vamos a extender nuestros conocimientos con uno de los grandes avances en la memoria de los computadores: la memoria caché.

Nos centraremos en ver:

- La necesidad de la incorporación de la caché a los sistemas actuales.
- La definición el concepto de conjunto de vecindad y su principio de funcionamiento.
- El análisis de la estructura interna de la caché
- Un estudio de uno de sus métodos de funcionamiento, el conocido como función de correspondencia.
- Un ejemplo de su funcionamiento.



Aunque este es un tema fácil, la experiencia nos dicta que resulta muy abstracto de entender. Por lo tanto, te recomendamos que dediques el tiempo necesario y hagas de tu puño y letra el ejemplo que aparece en la última pantalla de contenido, ya que, después de leer la teoría, es el único método de afianzar los conocimientos adquiridos.

Una vez entendido este tema, el siguiente será mucho mas fácil de entender, ya que es una prolongación de este y se basa en buscar optimaciones a este método que, aunque parezca perfecto, tiene algunos puntos poco eficientes.

La necesidad I

En la siguiente tabla, se presentan las características de tres tipos de memoria (en 2010).

	Tiempo medio de acceso	Precio por un gigabyte
Memoria SRAM	Entre 0,1 y 1,5 nanosegundos	Entre 3000 y 5000 dólares
Memoria DRAM	Entre 50 y 70 nanosegundos	Entre 50 y 100 dólares
Disco Duro	Entre 5 y 20 milisegundos	Entre 0,1 y 1,0 dólares

Los datos de la misma muestran claramente a lo que nos hemos referido en temas anteriores respecto a precio, rapidez y tamaño. Como ya sabemos, este problema se soluciona con la jerarquía de memoria, por lo que pasaremos a ver por qué la caché.

La caché es una memoria de tecnología static RAM, lo que hace que sea una memoria extremadamente rápida, pero el inconveniente es que es una memoria cara, que consume mucha energía y que físicamente es grande. Por lo tanto, ha tenido que pasar mucho tiempo hasta que este se integrase on-chip en los microprocesadores. Intel lo introdujo en su modelo 80486 allá por el año 1990 (hace poco más de 20 años) con 16Kb de caché, lo que permitía un aumento del doble en la ejecución de instrucciones.

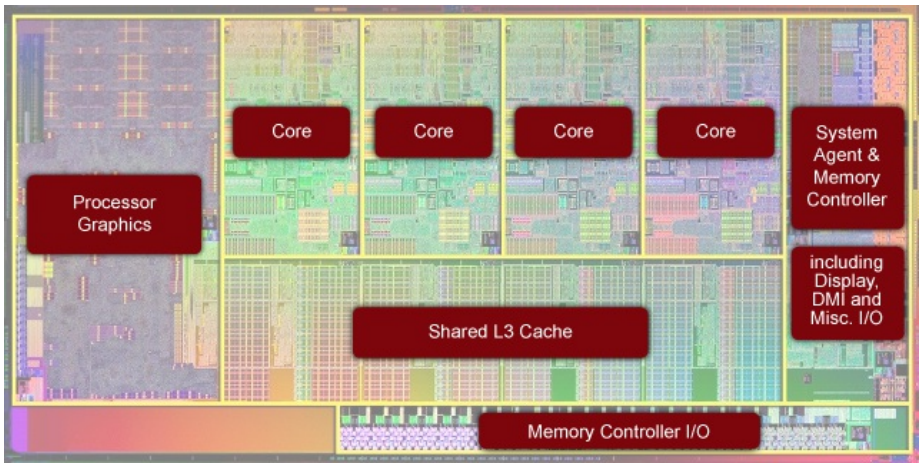
El propósito de la memoria caché es reducir los tiempos de acceso de datos e instrucciones a la velocidad del microprocesador, ya que este es mucho más rápido que el tiempo de acceso a memoria, por lo que el microprocesador tiene que esperar a que los datos estén disponibles.

La caché reduce este tiempo aproximadamente a 1, es decir, hace que la velocidad de microprocesador y de la caché sean prácticamente las mismas.

> Para que esto ocurra, la caché tiene que estar a muy corta distancia del microprocesador (milímetros), con un bus de comunicación de muy alta velocidad y protegido de señales electromagnéticas para que la comunicación sea estable sin necesidad de códigos correctores de errores. En definitiva, tiene que estar fabricada dentro del propio micro controlador.

La necesidad II

En la imagen se puede ver un microprocesador de la familia Intel en el que se puede identificar la caché L3. Las cachés L2 y L1 no son compartidas al estar dentro de los propio cores.

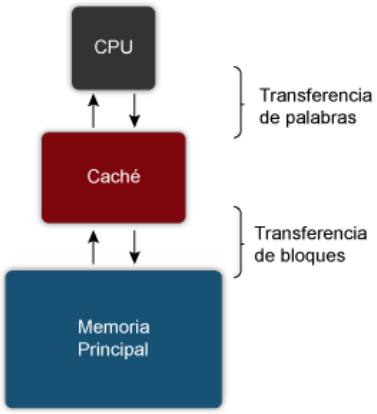


Para ser exactos, las cachés son los cuadrados amarillos que se diferencia en la esquina inferíos derecha de cada Core. En la imagen se puede apreciar la relación de espacio ocupado por los cores (unidades de trabajo) frente al espacio asignado por la cache, que es prácticamente la misma.

Por tanto, el propio espacio disponible en el microprocesador es una limitación para el tamaño de la caché.

Funcionamiento

Básicamente, la caché sirve de almacén temporal de datos que estarán disponibles para el microprocesador a una velocidad mucho más rápida que la de la memoria. Por otro lado, la memoria tiene que ser un elemento autónomo, ya que el microprocesador no tiene control sobre ella: es 100% autónoma. Lo único que detecta el microprocesador es que los tiempos de espera por los datos, se reducen a prácticamente 1.



Para ello, la memoria se coloca en medio de bus de comunicación entre el microprocesador y la memoria. En realidad, como ya sabemos, eso no es del todo cierto pues la caché es en realidad una interface de comunicaciones entre el bus local y el bus de sistema, que es donde se sitúa la RAM. Sin embargo, a efectos de funcionamiento, podemos verla como antes se ha explicado.

Por tanto, cuando el microprocesador solicita el dato según una dirección, esta petición es interceptada por la caché, que momentáneamente retiene la petición hasta comprobar si el dato existe en caché o no, existiendo así dos posibilidades:

- **El dato esta en caché**, lo que se conoce como acierto de caché, respondiendo la caché con el dato y no enterándose nunca la memoria de la petición de información. El microprocesador solo detecta que el dato pasa a estar disponible en un tiempo record.
- **El dato no está en caché** por lo que esta deja que la petición continúe propagándose por los buses hasta llegar a la memoria RAM, quien atiende la petición y responde con el valor almacenado en esa dirección. Este caso se conoce como fallo de caché. La caché se quedará con una copia de este dato, para futuras referencias.

Aspectos a aclarar

Aspectos a aclarar

Es importante aclarar los siguientes aspectos:

- La caché no tiene datos propios, solo copia de datos almacenados en la RAM, siendo esta la que tiene todos los datos que el microprocesador puede direccionar.
- La caché es mucho más rápida y pequeña que la RAM, por lo que solo un pequeño conjunto de datos de la RAM puede estar simultáneamente en caché.

Organización

La memoria caché tiene un método de acceso asociativo, lo que significa que los datos almacenados en ella tienen asociado un valor al que llamaremos etiqueta y que se puede utilizar para identificar el dato almacenado.

Además, la caché no guarda los datos igual que los guarda la RAM, en bytes, sino que los guarda en bloques relacionados por cercanía. Los datos que la caché considera cercanos, conocidos como **conjunto de vecindad,** varían con la implementación de la misma. Todos los valores de un conjunto de vecindad tienen asociada una única etiqueta que se liga con el conjunto de vecindad, no con los valores almacenados en la caché.

¿Cómo se organiza la caché?

Consideremos un ordenador con una capacidad de direccionamiento de memoria principal de hasta 2^n bytes direccionables, teniendo cada byte una única dirección de **n** bits. Consideremos además una caché de tamaño mucho menor que 2^n y organizada según C líneas y K bytes cada línea (una línea es un conjunto de vecindad).

Imaginemos que la memoria está dividida en un número de bloques y que, cada bloque, es curiosamente del mismo tamaño que el conjunto de vecindad (de longitud fija, de K bytes) provocando que la memoria tenga $M = 2^n/K$ bloques. El número de líneas es considerablemente menor que el número de bloques de memoria principal ($C << M$).

Organización de la caché

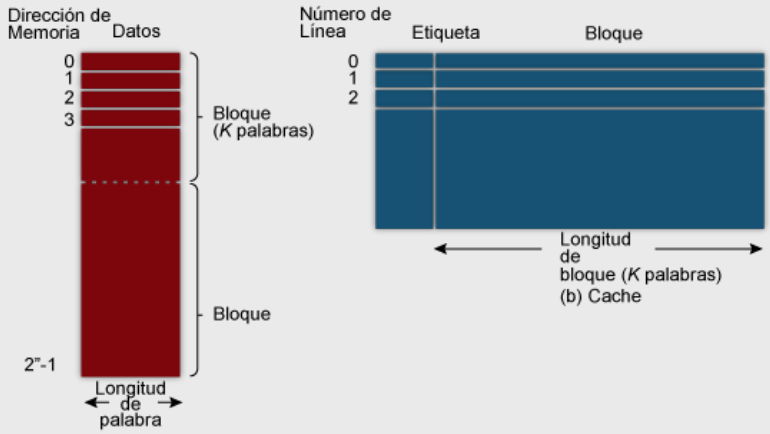
Al definir la misma división de bloques en caché (la línea) y en memoria (cada bloque imaginario), el movimiento de datos entre una y otra memoria es perfecto, pues un bloque cabe perfectamente en la caché.

Por tanto, en todo momento, un subconjunto de los bloques de memoria residirá en las líneas de la caché. Ya que hay más bloques que líneas, una línea dada no puede dedicarse unívoca y permanentemente a un bloque por lo que es aquí donde la etiqueta toma su importancia: cada línea incluye una etiqueta que identifica qué bloque de memoria en particular está siendo almacenado en una línea de caché. La etiqueta es usualmente una porción de la dirección de memoria principal, como describiremos más adelante en este tema.

Identificar el dato almacenado

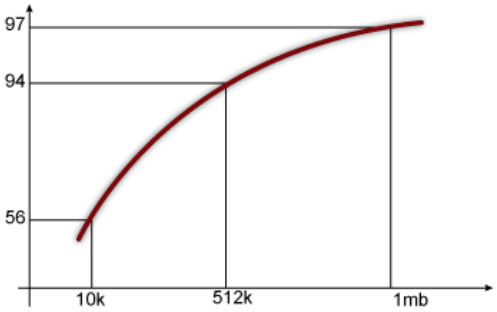
Es como si dijésemos que tenemos una memoria que puede almacenar datos que, en lugar de almacenarse en una dirección de memoria, se almacenan con un nombre (guarda *peso=40*, lee *edad*, guarda *altura=1,45*).

Organización de la caché



Tamaño

La elección del tamaño de la caché es una **cuestión crítica en el diseño de un microprocesador actual**. Ya hemos especificado algunas causas: precio, espacio, consumo eléctrico, etc., pero el principal motivo de elección es la eficiencia, siendo en este punto donde se difiere de la RAM. La caché, al aumentar de tamaño, tiende a ser ligeramente más lenta, ya que implica más electrónica y, por consiguiente, más tiempo en buscar un hueco libre o localizar un dato.



No solo está el tamaño de la caché, sino la relación de número de líneas por tamaño de la línea. A más líneas de caché, más conjuntos de vecindad diferentes simultáneamente en caché, lo que es bueno para sistemas multiproceso: cuanto más grande sea la línea, más probabilidad de éxito (acierto de caché) al existir un mayor conjunto de datos relacionados por entrada de caché, lo que suele ser mejor para sistemas con procesos grandes.

Dado que determinar un tamaño adecuado es complicado, se recurre habitualmente a la estadística:

> Por estadística simple, la probabilidad de que un dato solicitado por el microprocesador se encuentre en caché, es de K/M*100, lo que suele ser menos del 1‰, una mejora demasiado pequeña para los costes asociados. Actualmente, el porcentaje de éxito de la caché ronda el 98%, que es el número que determinará el tamaño de la caché.

Es decir, según se va aumentando el tamaño de la caché, mejoramos el porcentaje de aciertos de caché según una grafica que tiene una curva de crecimiento como la que se muestra en la imagen. Cada vez que se duplica el tamaño (se duplican costes), la grafica se acerca el tope teórico del 100% de aciertos (imposible de alcanzar). Una vez que esta grafica llega a unos baremos de éxito cercanos al 98%, lo que se considera un tamaño óptimo, pues duplicar costes implicaría una mejora inferior al 2%, significando por tanto demasiado gasto para el beneficio obtenido.

Más tiempo

Esta reducción de eficacia no viene determinada por diferencias en la tecnología de fabricación, sino por la complejidad inherente de su fabricación.

Se recurre habitualmente a la estadística

Por este motivo, cada microprocesador determina un tamaño y configuración determinados.

Función de correspondencia

Como ya hemos visto, la caché es mucho más pequeña que la memoria RAM y, a pesar de ello, la caché consigue una tasa de acierto del 98% aproximadamente. ¿Y cómo es esto? La respuesta nos la dan las siguientes circunstancias.

El principio de vecindad	A cortos periodos de tiempo (*milisegundos*) el conjunto de datos que maneja el microprocesador es reducido y cercano. Este principio concuerda exactamente con el funcionamiento de la caché y la idea de organizar los datos en conjuntos de vecindad.
Las funciones de correspondencia	Una función de correspondencia es un algoritmo que se ejecuta en la caché y que determina qué datos (de todos los que están en la memoria) se sitúan en la caché y dónde lo hacen, con el fin de maximizar el numero de aciertos.

Documentos Algoritmo ejecutado por la cache al pedir el procesador una dirección

¿Por qué la necesidad de una función de correspondencia?

La explicación es simple: cuando después de un fallo de caché hay que elegir un hueco en ella para meter un elemento procedente de memoria, hay que tener en cuenta que la posición que se le reserve tiene que ser fácil de encontrar entre los datos ya existentes en caché, dado que después, con toda seguridad, habrá que volver a buscar ese dato y la búsqueda tiene que ser muy rápida.

Podríamos pensar en utilizar un método como el de la memoria, utilizando como elemento de búsqueda la dirección del dato.

El problema es que la caché tiene muchísimas menos direcciones que la memoria, haciendo imposible establecer una relación directa. Esto origina que habría que tener una lista de los elementos guardados en caché y de dónde están guardados, de manera que para encontrar un dato, hay que revisar esa lista para saber dónde se ha guardado (o si no está).

Si tenemos en cuenta que la caché es muy variable, pues los datos guardados en ella varían a cortos periodos de tiempo, obtendríamos una lista sumamente desordenada lo que complicaría mucho encontrar de nuevo el dato.

Complicaría mucho encontrar de nuevo el dato

Para hacer un símil, piensa en el tiempo que tardarías en encontrar un número de teléfono en una agenda en la que los contactos estuvieran desordenados.

Correspondencia directa I

Una de las funciones de correspondencia más utilizada en la caché, es la conocida como **correspondencia directa**.

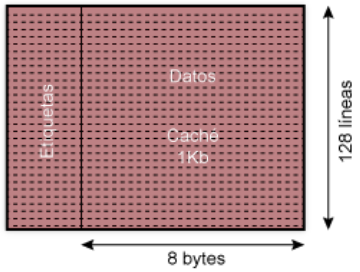
Para entenderla, vamos a pensar en el mismo ejemplo de antes: la agenda.

Cuando introducimos un nuevo nombre en vuestra agenda, no lo escribimos en cualquier posición, sino en una hoja en concreto, normalmente en la hoja con la letra por la que empieza el nombre de la persona en cuestión.

¿Por qué hacemos esto?	Para que cuando busquemos de nuevo ese nombre, podamos encontrarlo rápidamente. Con ello, reducimos el tiempo de búsqueda con respecto a la lista de personas que empiecen por esa letra. Aunque no es un método perfecto, sí resulta un buen método para mantener una agenda ordenada.
¿Qué defectos puede tener este sistema?	La super utilización de algunas letras frente a la infrautilización de otras. Apellidos como Gómez o Fernández llenan la letra G y F respectivamente, mientras que la W queda prácticamente vacía (es España).

Este mismo sistema es el que utiliza la caché que implementa una función de correspondencia directa, aunque la manera de determinar dónde se guarda la información, es por la dirección física de ese dato en memoria principal.

Veamos un ejemplo: supongamos que tenemos una memoria caché de 1Kb organizada en líneas de 8bytes. Por tanto, nuestra caché tiene $2^{10}/2^3=2^7$ líneas, es decir 128 líneas de datos.



Si la memoria que tiene el ordenador es de 1Mb, las direcciones de memoria están formadas por 20bits. Si pensamos en que la memoria también se pudiese direcciones con un número de dirección, al tener solo 128 líneas, necesitaríamos solo 7 bits.

Correspondencia directa II

Demos unas vueltas a una dirección de memoria relacionándola con el tamaño de caché, pensemos por ejemplo en la dirección que se correspondería con la línea roja de la imagen. De este análisis podemos desarrollar:

El conjunto de vecindad que hemos definido, y que coincide siempre con el número de bytes almacenados en una línea de caché, es de 8. Eso implica que los datos que se mueven entre memoria y caché lo hacen siempre en bloques de 8 (el conjunto de vecindad).

No es necesario tener en cuenta los elementos individuales, solo conjuntos de vecindad, por lo que, de esos 20bits, solo me interesan los 17 bits que forman los 2^{17} bloques (o conjuntos de vecindad). Es decir, la memoria de nuestro ejemplo tiene 2^{17} conjuntos de vecindad y en memoria solo cogen 2^{7} (=128).

Ahora ya no pensamos en el elemento direccionado por el microprocesador, sino en el conjunto de vecindad donde se incluye dicho dato. Por tanto, ya no hay que buscar un hueco, en la caché, al dato (de tamaño byte) solicitado por el microprocesador, sino al conjunto de vecindad.

Si volvemos al símil de la agenda, lo que hacemos es usar parte del nombre para saber donde guardarlo, pues la caché usará parte de la dirección del conjunto de vecindad, para saber donde le corresponde guardarlo y que así sea fácil de encontrar en caso de futura necesidad.

Bloques de 8 (el conjunto de vecindad)

Si se produce un fallo de caché en la lectura de un dato, y hay que traerlo de memoria, no solo se trae ese elemento, sino todo su conjunto de vecindad, para llenar una línea completa de caché.

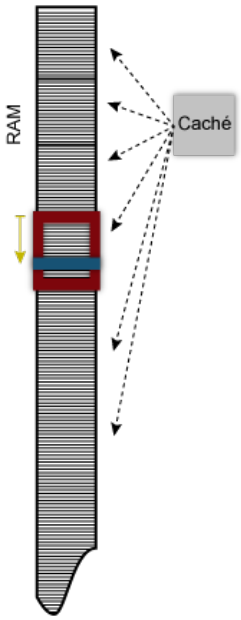
17 bits

1Mgb=2^{20} bytes, y el conjunto de vecindad es de 8=2^{3}bytes. Por tanto la división es 2^{17} conjuntos.

Correspondencia directa III

¿Y qué parte de la dirección utilizar? ¿El principio del nombre como en la agenda?

Pues no, todo lo contrario, el final. Lo que haremos será dividir el número de bloques de memoria por el número de líneas de la caché.



¿Cómo interpretamos esta división?

De esta división salen dos valores, el cociente y el resto. El cociente nos indica cuántas veces es más grande la memoria que la caché o, lo que es lo mismo, en cuántas partes tamaño caché puedo dividir la memoria (véase la imagen).

Para nuestro ejemplo: $2^{17}/2^{7}=2^{10}$. La memoria es 2^{10}=1024 veces más grande que la caché, si lo vemos desde otra perspectiva, que si ponemos 1024 cachés juntas una tras otra, hacemos una memoria.

¿Y qué es el resto de la división?

Imaginemos esas 1024 cachés una seguida de otra hasta completar la memoria total.

El resto es el desplazamiento del dato solicitado en memoria, dentro del segmento caché que solapa la memoria en la que se encuentra el dato solicitado por el microprocesador. En la imagen, la flecha verde representa el desplazamiento dentro de la hipotética superposición de la caché en la memoria, que a su vez corresponde con la superposición dibujada en azul.

Correspondencia Directa IV

Reorganizando todo lo que hemos visto: una dirección de memoria real que el microprocesador puede solicitar, puede referenciarse por su dirección absoluta de memoria o según la posición que ocupe dentro de:

Un conjunto de vecindad.
El conjunto de vecindad, como un desplazamiento desde el inicio de una hipotética superposición de muchas cachés sobre la memoria.
El desplazamiento referido al número de caché en esa hipotética superposición.

Es decir, si tomamos, por ejemplo, la siguiente dirección real de memoria: $535717_{(10)}$. Esta se podría descomponer diciendo que:

Es el elemento 5 del conjunto de vecindad de 8 elementos.
Esta desplazada desde el inicio de la caché que lo contiene 20 líneas de las 128 que forman la caché.
Se enmarcaría dentro de la copia de caché 523 de las 1024 que se necesitan para completar una memoria.

De este modo, la pregunta ahora es:

¿Como obtenemos estos valores desde una dirección de memoria real del tipo $11010010110010101001_{(base2)}$?

1/4

Para obtener la posición en el conjunto de vecindad, solo hay que saber su desplazamiento dentro de una agrupación de 8 en 8. Es decir, el resto de dividir el número por 8:

$11010010110010101011_{(base2)}/8_{(base10)}$ = 1101001011001010101 1/23.

De aquí se obtiene un cociente **$11010010110010101_{(base2)}$** y un resto de **$011_{(base2)}$**, es decir **$3_{(base10)}$**, por lo que es el cuarto elemento del conjunto de vecindad (el primero es el 0).

Para, desde el cociente anterior, obtener el desplazamiento y el número de caché de esa hipotética superposición que son los conjuntos de vecindad que hay en una memoria de ese tamaño (1Mb), se divide por el número de líneas que tiene la caché.

$11010010110010101_{(base2)}/128_{(base10)}$ = $11010010110010101_{(base2)}/27_{(base10)}$

Obtenemos:
Como cociente: **1101001011**, que coincide con el número de caché en la que se enmarcará **11010010112=84310** (de 1024).
Como resto: **0010101**, que se corresponde con el desplazamiento **$0010101_2=21_{10}$ (de 128).**

Por si no ha quedado claro, nótese dividir un número en binario, por una potencia de dos, es lo mismo que truncar. No hay que dividir, solo truncar el número, es lo mismo que dividir por diez un número en decimal.

2/4

Si analizamos estas divisiones, lo que hemos hecho es dividir una dirección de memoria en tres campos:

Dirección de memoria completa		
Numero de caché	Desplazamiento dentro de la caché	Posición en el conjunto de vecindad

Para nuestro ejemplo:

20 bits 11010010110010101011		
10 bits $1101001011 = 843_{10}$	7 bits $0010101 = 21_{10}$	3 bits $011 = 3_{10}$
Etiqueta	Línea	Palabra

3/4

¿Y a qué viene todo esto?

Pues a que una vez obtenidos estos tres valores ya tenemos el algoritmo que permite determinar donde se guardaría la copia en caché de ese valor solicitado por el microprocesador. Se almacenará en la línea de caché que determinan los bits llamados línea de la tabla anterior.

¿Y para qué sirve la etiqueta?

Pues si analizamos cuantas direcciones de memoria, de las que pueden ser solicitadas por el microprocesador, se escribirían en esa misma línea de caché, vemos que son muchas. De hecho, a cualquier dirección que tenga los mismos bits en el campo línea, pero en la que varíen los bits de etiqueta, le corresponderá la misma posición en caché.

¿Como determina entonces al algoritmo, que es ese el dato guardado y no otro?

Debido a que, junto al conjunto de vecindad, se guardan los bits de etiqueta. Con esto, ya tenemos perfectamente definido el elemento guardado en esa línea de caché. De todas las posibilidades, es justamente la que tiene la etiqueta en concreto.

4/4

Ejemplo de función de correspondencia directa

Consideremos un microcontrolador con un bus de direcciones capaz de soportar hasta 2Mb de memoria. A este microcontrolador, se le dota de una caché de 2Kb que define un conjunto de vecindad de 4 bytes.

(a) Definir cuántos bits se le asignan a cada uno de los campos de una dirección de memoria.

Memoria completa	El también conocido como MAR (registro de direcciones a memoria) tiene el número de bits necesarios para poder direccionar toda la memoria del sistema. En nuestro caso seria de 2Mb o lo que es lo mismo 2^{21} bytes. Por tanto, necesita 21 bits.
Palabra	Tiene el tamaño justo y necesario, en bits, para direccionar los vecinos de un conjunto de vecindad. Si se ha decidido un conjunto de vecindad de 8 bytes, entonces 8=23, por lo que el tamaño será 3 bits.
Línea	Si la caché completa tiene 2Kb organizados en L líneas y K vecinos, entonces L x K = 2Kb. Desarrollando la ecuación: L = 2Kb/8 = 211/23 = 28, por lo que tiene 256 líneas. Es decir, para direccionar 256 se necesitan 8 bits.
Etiqueta	Si la memoria es de 2Mb y la caché es de 2Kb, entonces la caché es 2Mb/2Kb = 221/211 = 210 = 1024 veces más pequeña. Es decir, se necesitan 1024 cachés para hacer una memoria y se necesitarían (1024=210) 10 bits para direccionar la copia de caché. Por tanto la etiqueta es de 10 bits.
Comprobación	Etiqueta + línea + palabra = memoria completa 10+8+3=21 Perfecto

1/2

Esquema del MAR:

Dirección completa: 21 bits		
Etiqueta	Línea	Palabra
10 bits	8 bits	3 bits

(b) Determinar dónde se guardan los datos que el microprocesador solicita según estas direcciones:

- $11011010100101010010101_{(2)}$:

Etiqueta	Línea	Palabra
$1101101010_{(2)}=874_{(10)}$	$01010100_{(2)}=84_{(10)}$	$101_{(2)}=5_{(10)}$

- $11011100100101010011010_{(2)}$:

Etiqueta	Línea	Palabra
$1101110010_{(2)}=882_{(10)}$	$01010100_{(2)}=84_{(10)}$	$110_{(2)}=6_{(10)}$

(c) ¿Se guardan en la misma posición de caché las dos direcciones anteriores?

La respuesta es sí, ya que las dos comparten la línea y, como recordaremos, ese campo indica la línea en caché en la que se va a guardar el dato. Simplemente se comparan las etiquetas y como, al corresponder a direcciones de memoria distinta, son distintas, la caché detecta fácilmente que no es el mismo dato y que, por tanto, hay fallo de caché.

2/2

Resumen

En este tema hemos visto por qué la memoria caché ofrece una mejora tan grande a los computadores actuales. Tenemos que recordar que un acierto de caché reduce el tiempo de espera por parte del procesador a una milésima parte.

Los puntos que deben haberte quedado claros al finalizar este tema son:

- La necesidad de la incorporación de la caché a los sistemas actuales.
- La definición del concepto de conjunto de vecindad y su principio de funcionamiento.
- La estructura interna de la caché.
- La función de correspondencia directa.
- Cómo interpretar una dirección de memoria para comprobar donde se guardaría en la caché.

Como habrás comprobado, el tema parece mucho más complicado de lo que es. Una vez comprendido el funcionamiento de la caché, todo lo demás es coherente: la necesidad de una función de correspondencia, el principio de vecindad, el tamaño y organización de la caché, la existencia de la etiqueta, etc.

Antes de dar por terminado el tema, es muy importante que copies, en una hoja aparte, el enunciado del ejemplo que aparece en la última pantalla y que, sin mirar los apuntes, intentes razonar de nuevo el método para resolverlo. Este tema se aprende con ejercicios, pero para realizar estos hay que tener la teoría muy clara.