Parte IV

Jerarquía de memoria

Competencias genéricas

- Que los estudiantes sepan aplicar sus conocimientos a su trabajo o vocación de una forma profesional y posean las competencias que suelen demostrarse por medio de la elaboración y defensa de argumentos y la resolución de problemas dentro de su área de estudio.
- Que los estudiantes tengan la capacidad de reunir e interpretar datos relevantes (normalmente dentro de su área de estudio) para emitir juicios que incluyan una reflexión sobre temas relevantes de índole social, científica o ética.

Competencias específicas

- Conocer la estructura, organización, funcionamiento e interconexión de los sistemas informáticos.
- Conocer la aplicación de los sistemas informáticos para la resolución de problemas propios de la ingeniería.
- Adquirir la capacidad de análisis y resolución de problemas.
- Saber interpretar los resultados.
- Desarrollar la capacidad para trabajar en grupo.
- Desarrollar de forma efectiva la comunicación oral y escrita.
- Desarrollar habilidades de aprendizaje necesarias para emprender estudios posteriores con un alto grado de autonomía.

Capítulo 7

Memoria caché

7.1. Introducción. Principio de localidad

Como vimos en el tema 1, la velocidad del procesador ha ido aumentando más rápidamente que la velocidad de las memorias. Para compensar esto se toman soluciones arquitectónicas: un computador va a tener distintos tipos de memorias que van desde la memoria cara y rápida de los registros internos hasta las memorias baratas y lentas de los discos removibles. Su funcionamiento va a ser equivalente al que tendría si tuviera una memoria única, grande y rápida. La forma en que se organizan estos distintos tipos de memorias se conoce como *jerarquía de memorias [1].*

Estos subsistemas de memorias de distintas tecnologías, costes y rendimientos serán internos al propio computador (directamente accesibles desde la CPU) o externos (accesibles desde la CPU a través de un módulo de E/S).

De esta forma optimizamos el acceso a la información que almacena la unidad de memoria (instrucciones + datos). La jerarquía de memorias se basa en:

- El *Principio de Localidad de las Referencias*: los computadores tienden a reutilizar los datos y las instrucciones que utilizaron recientemente. Tiene dos aspectos:
  - temporal: en un tiempo cercano se accede a datos o instrucciones a los que se ha accedido recientemente.

```
Ejemplo: a = 5;
         b=a+10;
         if (b<a)
           ...
          while(i<0)
          {
           ...// las instrucciones de dentro del bucle se repetirán (se accede de nuevo)
```

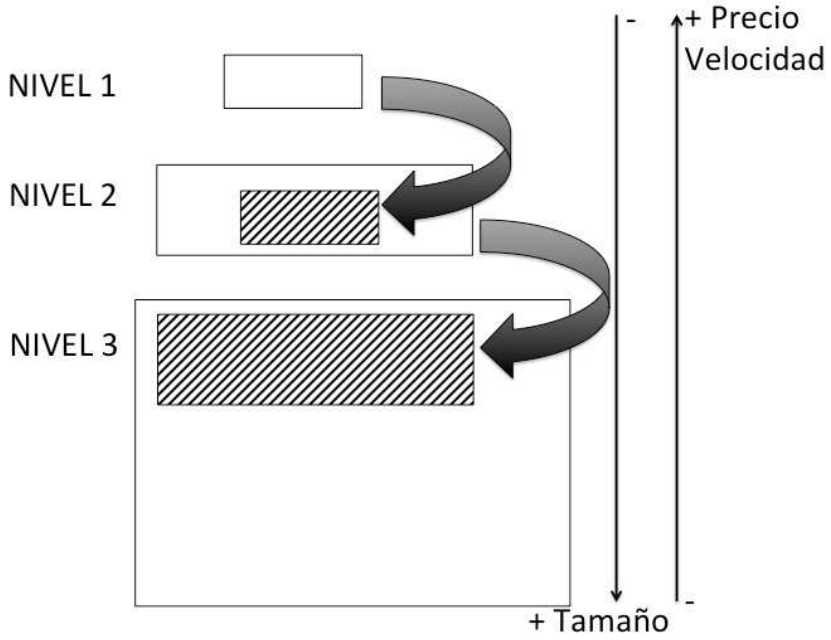

Figura 7.1: *Esquema de los distintos niveles de memoria los cuales a medida que aumenta el tamaño disminuye el precio y la velocidad. Cada nivel está contenido en el siguiente.*

```
}
```

- espacial: la probabilidad de que el próximo dato o instrucción que se utilice esté en una posición de memoria cercana a la anterior es elevada. Por ejemplo, la ejecución es secuencial por defecto (las instrucciones ocupan posiciones consecutivas en memoria), el acceso a los elementos de un array (sus elementos ocupan posiciones consecutivas en memoria), etc.

- El hardware más pequeño es más rápido.

- El hardware más rápido proporciona mayor velocidad.

Por tanto, en el computador se tiene un sistema de niveles, una jerarquía de memoria en el que cada nivel está incluido en el siguiente, como se muestra en el esquema de la figura 7.1.

Un esquema típico de jerarquía de memoria en un computador sería el que se muestra en la figura 7.2. Para acceder a la memoria principal se sale del chip y se usa un bus. Para acceder a la memoria virtual, dentro del disco duro hay que hacer una operación de E/S.

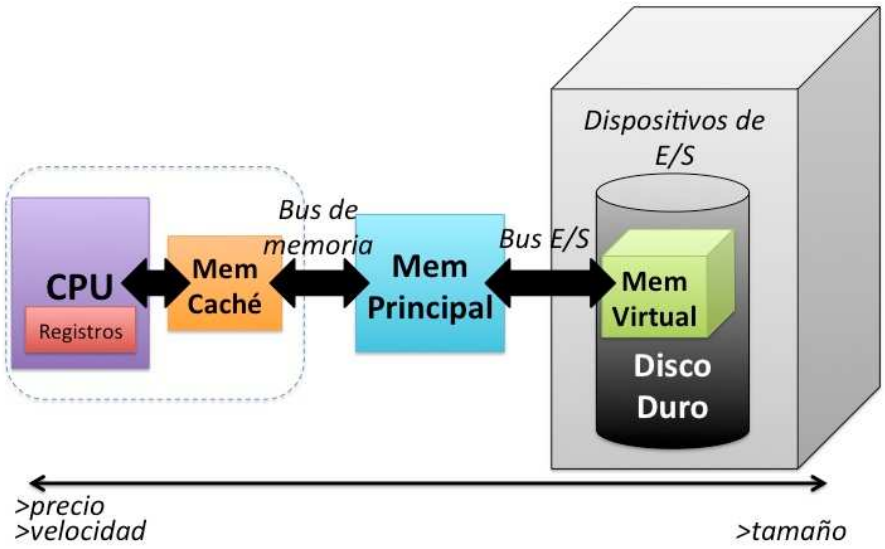

Figura 7.2: *Esquema de los distintos tipos de memoria. A medida que almacenan más información son más lentos e incluyen más operaciones para acceder a su contenido. Cada nivel está contenido en el siguiente.*

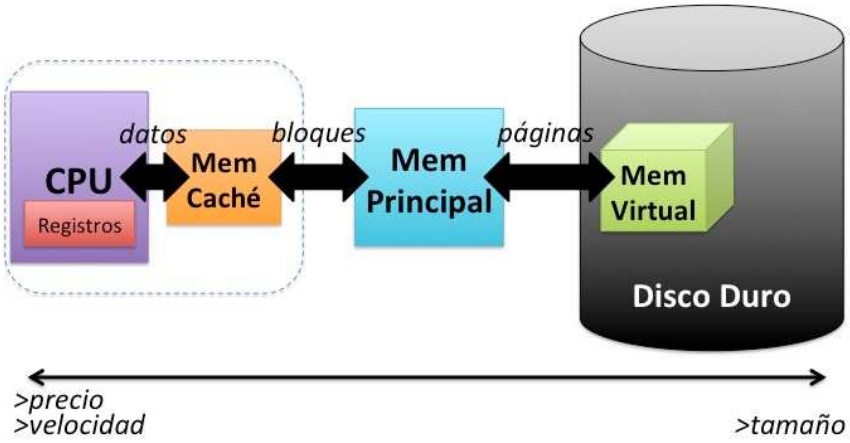

Figura 7.3: *Localización de la memoria caché en la jerarquía de memoria.*

7.2. La memoria caché

La CPU accede a memoria para buscar instrucciones, leer operandos o almacenar resultados. Por tanto, la velocidad con que la CPU puede ejecutar instrucciones está limitada por el *tiempo de ciclo de memoria* t_c (intervalo de tiempo mínimo entre dos accesos a memoria consecutivos *[1]*).

Se podría construir la memoria principal con la misma tecnología que los registros de la CPU para que fuera rápida, pero es my costoso. Lo que se hace es colocar una memoria pequeña, muy rápida, físicamente próxima a la CPU, entre la CPU y la memoria principal *[1]*. Es la *memoria caché*. La memoria caché almacena una copia de ciertas partes de la memoria principal, basándose en el Principio de Localidad de las Referencias. En la figura 7.3 se muestra qué lugar en la jerarquía de memoria ocupa la memoria caché.

Se pueden producir dos situaciones:

- Acierto de caché (*cache hit*): la CPU solicita un dato y está en la memoria caché. Como la memoria caché tiene copia de partes de la memoria principal si el dato que solicita la CPU coincide que está almacenado en la memoria caché, ésta le devuelve el dato y se produce un acierto de caché.

- Fallo de caché (*cache miss*): la CPU solicita un dato y no está en la memoria caché. Como la memoria caché es más pequeña que la memoria principal, no puede almacenar toda su

información. En ocasiones, el dato que solicita la CPU no se encuentra en la memoria caché con lo que se produce un fallo de caché. En este caso, se transfiere un bloque de la memoria principal a la caché el cual incluye el dato solicitado. Una vez que la caché ya tiene el bloque en cuestión, le devuelve el dato solicitado por la CPU.

Un bloque es la unidad de transferencia (intercambio de información) entre la memoria caché y la memoria principal. Tiene un tamaño mayor que la palabra (dato solicitado por la CPU).

La *memoria virtual* permite que los programas direccionen la memoria sin considerar la cantidad de memoria principal real. Cuando un programa se ejecuta, sólo se carga en memoria principal la parte de código y datos que se van a utilizar. Como un proceso puede tener asignada más memoria que el tamaño de la memoria principal, lo que ocurre es que partes de dicho proceso estarán en memoria principal, pero habrá partes que no. Si el bloque que le pide la memoria caché está almacenado en memoria principal, se lo devuelve a la memoria caché (acierto de página). Si no está (fallo de página), se va a la memoria virtual y se trae a la memoria principal la página o segmento que lo contiene (una página o segmento tiene muchos bloques). Los aciertos/fallos de página aparecen al acceder a memoria principal. Los fallos de página se resuelven por software, mientras que los de caché se resuelven por hardware.

Cuando la memoria caché se integra en el propio procesador, puede resultar imposible aumentar su tamaño si el rendimiento del sistema no es el adecuado. Al aumentar su tamaño puede que la caché sea más lenta. Una alternativa es introducir una segunda caché de mayor capacidad entre la primera caché y la memoria principal. Es la caché de segundo nivel o caché secundaria (L2). En una referencia a memoria, el procesador accederá a la caché de primer nivel. Si se produce un fallo en la caché de primer nivel, se accede a la caché de segundo nivel. Si tampoco está allí, se va a la memoria principal. Por último, en algunas organizaciones existen memorias caché de tercer nivel.

7.3. Ubicación de bloque

Hay varias formas de organizar una caché para almacenar información. Como la CPU referencia a la caché con la dirección de memoria principal del dato que necesita, la caché debe usar esta dirección para encontrar el dato [1].

La correspondencia entre la información de memoria principal y la de la caché debe implementarse en hardware para conseguir un rendimiento óptimo [1].

La organización de la memoria caché consiste en establecer la función de correspondencia, que asigna a los bloques de la memoria principal posiciones definidas en la memoria caché [1]. Para el cálculo de dicha función de correspondencia se emplean 3 técnicas básicas [1]:

a) Mapeo directo.

En este tipo de organización cada bloque de memoria principal puede estar en una y sólo una posición de la caché. Suponiendo que la memoria caché tiene espacio para C bloques y que

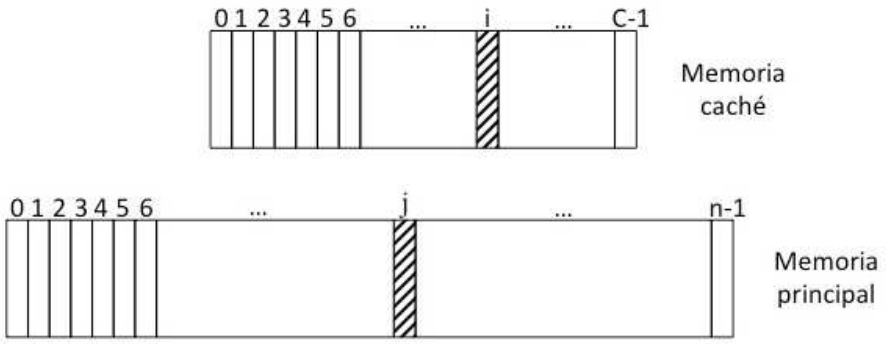

Figura 7.4: *Esquema de la organización de la memoria caché usando mapeo directo.*

la memoria principal tiene n bloques tal y como se muestra en la figura 7.4, el bloque j de la memoria principal ocupará la posición i de memoria caché aplicando la siguiente función:

$$i = j \ mod \ C$$

Siendo i el número de bloque asignado en la memoria caché al bloque de la memoria principal, j el número de bloque de la memoria principal y C el número de bloques que tiene la memoria caché.

Por ejemplo, si la memoria caché tiene 10 bloques, el bloque 18 de la memoria principal estará en la posición 8 puesto que el resto de dividir 18 entre 10 da 8.

b) Totalmente asociativa.

Cada bloque de la memoria principal puede estar en cualquier sitio de la caché.

c) Asociativa por conjuntos.

La memoria caché se divide en q conjuntos, cada uno de los cuales consta de r bloques como se muestra en la figura 7.5. Se dice que la caché es *asociativa por conjuntos de r vías*. Para un bloque de memoria principal, el conjunto donde se almacena en la memoria caché se elige por mapeo directo, es decir, se divide la dirección del bloque entre el número de conjuntos y se toma el resto. Dentro de un conjunto la dirección es totalmente asociativa (se puede almacenar en cualquier bloque de dicho conjunto). Por tanto, se tiene que:

$$C = q * r$$

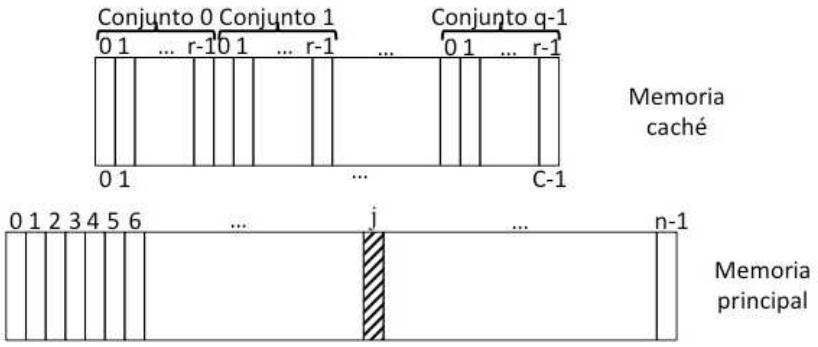

Figura 7.5: *Esquema de la organización de la memoria caché asociativa por conjuntos de r vías.*

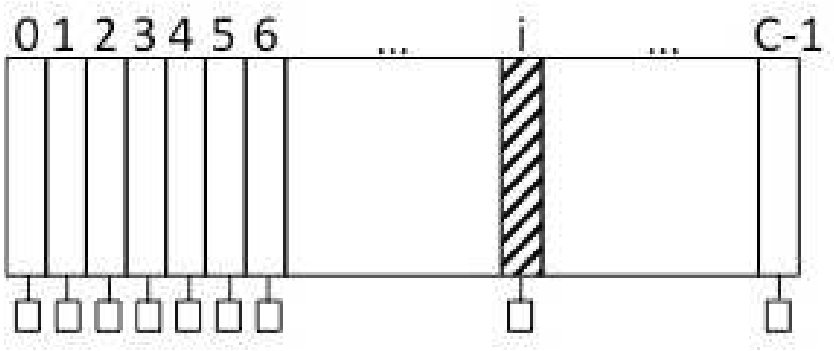

Figura 7.6: *Esquema de la memoria caché. Cada bloque tiene una etiqueta asociada.*

$$k = j \ mod \ q$$

Siendo q el número de conjuntos de la memoria caché, r el número de bloques de cada conjunto y C el número de bloques de la memoria caché. El bloque j de la memoria principal se almacena en el conjunto k de la memoria caché.

7.4. Identificación de bloque

Los bloques de memoria caché tienen asociado una etiqueta como se muestra en la figura 7.6. Cuando la CPU genera una dirección para acceder a una palabra de la memoria principal su formato desde el punto de vista de la memoria caché depende del tipo de organización de la memoria caché[1].

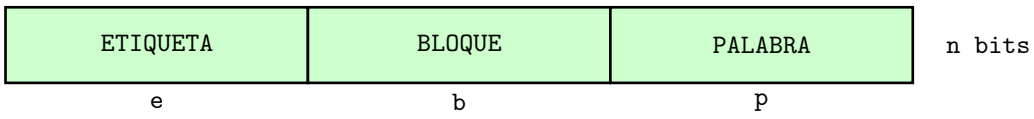

Figura 7.7: *Interpretación de la dirección de memoria principal por parte de una memoria caché por mapeo directo.*

a) Mapeo directo.

En este tipo de organización se divide la dirección de la memoria principal en 3 campos (Fig. 7.7): etiqueta, bloque y palabra. El número de bits de cada campo es [1]:

- Campo palabra: $p = log_2(K)$ bits siendo K el número de unidades mínimas direccionables por bloque.
- Campo bloque: $b = log_2(C)$ bits siendo C el número de bloques de la memoria caché.
- Campo etiqueta: $e = n - p - b$ bits siendo n el número de bits de la dirección de memoria principal.

El conjunto de campos etiqueta y bloque especifican la dirección de un bloque. El campo palabra indica el desplazamiento dentro de un bloque.

El funcionamiento es el siguiente [1]:

1. Se identifica un bloque de la memoria caché con el campo bloque.
2. Se compara la etiqueta de ese bloque de la memoria caché con el campo etiqueta de la dirección solicitada por el procesador.

   a) Si coinciden, con el campo palabra se selecciona la palabra pedida dentro del bloque y se le entrega a la CPU.

   b) Si al comparar las etiquetas no coinciden, quiere decir que no se encuentra en la memoria caché, se produce un fallo y habrá que ir a buscar el bloque que contiene la palabra a memoria principal. La dirección del bloque que contiene la palabra se consigue poniendo el campo palabra todo a 0s.

Las ventajas de este método es que la circuitería es sencilla. Como un bloque sólo puede ocupar una posición, la desventaja se produce si accedemos a dos bloques consecutivamente que han de almacenarse en la misma posición.

b) Totalmente asociativa.

Cuando la CPU genera una dirección para acceder a una palabra de memoria principal, su formato desde el punto de vista de la memoria caché se divide en 2 campos (Fig. 7.8): etiqueta y palabra. El número de bits de cada campo es [1]:

Figura 7.8: *Interpretación de la dirección de memoria principal por parte de una memoria caché totalmente asociativa.*

- Campo palabra: $p = log_2(K)$ bits siendo K el número de unidades mínimas direccionables por bloque.
- Campo etiqueta: $e = n - p$ bits siendo n el número de bits de la dirección de memoria principal.

El funcionamiento es el siguiente [1]:

1. Se busca la etiqueta de la dirección entre todas las etiquetas de la memoria caché. Esta búsqueda se realiza con circuitería.

   a) Si coinciden, con el campo palabra se selecciona la palabra pedida dentro del bloque y se le entrega a la CPU.

   b) Si no se encuentra la etiqueta en la memoria caché, quiere decir que el bloque buscado no se encuentra en la memoria caché, se produce un fallo y habrá que ir a buscar el bloque que contiene la palabra a memoria principal. La dirección del bloque que contiene la palabra se consigue poniendo el campo palabra todo a 0s.

Este método soluciona la desventaja de las memorias por mapeo directo. Sin embargo, tiene como desventaja que la circuitería para comparar simultáneamente etiquetas por *hardware* es compleja.

c) Asociativa por conjuntos.

Cuando la CPU genera una dirección para acceder a una palabra de memoria principal, su formato desde el punto de vista de la memoria caché se divide en 3 campos (Fig. 7.9): etiqueta, conjunto y palabra. El número de bits de cada campo es [1]:

- Campo palabra: $p = log_2(K)$ bits siendo K el número de unidades mínimas direccionables por bloque.
- Campo conjunto: $c = log_2(q)$ bits siendo q el número de conjuntos de la memoria caché.
- Campo etiqueta: $e = n - p - b$ bits siendo n el número de bits de la dirección de memoria principal.

El conjunto de campos etiqueta y conjunto especifican la dirección de un bloque de memoria principal. El campo palabra indica el desplazamiento dentro de un bloque. El campo conjunto

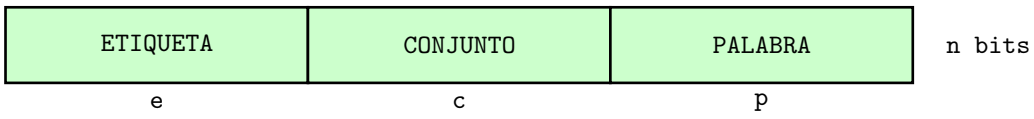

Figura 7.9: *Interpretación de la dirección de memoria principal por parte de una memoria caché asociativa por conjuntos.*

especifica uno de los $q = 2^c$ conjuntos de la memoria caché. Cada bloque en cada conjunto tiene una etiqueta almacenada que junto con el número del conjunto completa la identificación del bloque de la memoria caché.

El funcionamiento es el siguiente [1]:

1. Se utiliza el número de conjunto de la dirección solicitada por el procesador para acceder al conjunto correspondiente de la memoria caché.

2. Se comparan simultáneamente todas las etiquetas del conjunto seleccionado con la etiqueta de la dirección solicitada.

   a) Si coinciden, se accede a la palabra pedida en la memoria caché (usando el campo palabra).

   b) Si al comparar las etiquetas no coinciden, quiere decir que no se encuentra en la memoria caché, se produce un fallo y habrá que ir a buscar el bloque que contiene la palabra a memoria principal. La dirección del bloque que contiene la palabra se consigue poniendo el campo palabra todo a 0s.

Esta técnica contiene las dos técnicas anteriores como casos particulares:

- q=C y r=1. Se trata del caso de mapeo directo. Tenemos tantos conjuntos como bloques y cada conjunto sólo tiene un bloque.

- q=1 y r=C. Se trata del caso de memorias totalmente asociativas. Sólo tenemos un conjunto y en él están los C bloques de la memoria caché.

7.5. Sustitución de bloque

Cuando un nuevo bloque se transfiere a la memoria caché, debe sustituir a uno de los ya existentes. En el caso de que la organización de la memoria caché sea por mapeo directo, el bloque está determinado unívocamente y por lo tanto no se puede realizar ninguna elección.

Sin embargo, si la caché es totalmente asociativa o asociativa por conjuntos se necesita un algoritmo de reemplazo [1]. Hay varias técnicas, las cuales se implementan por *hardware*:

- bloque elegido de forma aleatoria: se toma un bloque cualquiera y se sustituye.

- bloque utilizado menos frecuentemente: emplea el algoritmo LFU (*Least Frequently Used*). Se sustituye el bloque menos utilizado desde que fue transferido a la memoria caché.
- bloque más antiguo en la memoria caché: se utiliza el algorimo FIFO (*First In First Out*). Se sustituye el bloque que ha permanecido durante más tiempo en la caché.
- bloque utilizado menos recientemente: usa el algoritmo LRU (*Least Recently Used*). Se sustituye el bloque que hace más tiempo que no se utiliza.

7.6. Estrategia de escritura

La memoria caché guarda copia de partes de la memoria principal. Por tanto, si se efectúa una lectura y el dato está en memoria caché, se lee y no hay ningún problema. Sin embargo, si se efectúa una escritura, ¿cómo se gestiona la operación? En el caso de que se quiera realizar una operación de escritura en una dirección de memoria y se produzca un acierto de caché, existen dos técnicas [1]:

- Escritura Inmediata (*Write Through*): Se efectúan todas las operaciones de escritura tanto en la memoria caché como en la memoria principal. Proporciona la siguiente ventaja: se mantiene la consistencia en todo momento. Hay los mismos datos en memoria caché que en memoria principal. Como desventaja, indicar que aumenta el tráfico entre memoria principal y memoria caché.
- Post-escritura (*Write Back*): En este caso se realizan las actualizaciones solo en la memoria caché. Para que pueda reemplazarse un bloque residente en la memoria caché es necesario considerar si ha sido modificado en la memoria caché pero no en la memoria principal. Si no lo ha sido, se puede reescribir el bloque antiguo de la memoria caché. Si se ha modificado el bloque, significa que al menos se ha realizado una operación de escritura sobre una palabra de ese bloque de la memoria caché y se debe actualizar la memoria principal. Se basa en la utilización de un bit denominado bit *dirty* o de actualización que vale 1 cuando el bloque se ha modificado. Si se reemplaza un bloque de memoria caché se reescribe en memoria principal si y sólo si el bit de actualización está a 1. Este método disminuye el tráfico entre memoria principal y memoria caché y disminuye el tiempo de acceso para las operaciones de escritura. Como desventaja, indicar que existe una inconsistencia porque en un momento dado los datos de la memoria caché y la memoria principal son distintos.

7.7. Rendimiento de una caché

Una medida de rendimiento de la memoria caché es la tasa de acierto (h) [1]:

$$h = \frac{númeroDeHits}{númeroTotalAccesos}$$

La tasa de acierto es la relación entre el número de veces que la palabra solicitada se encuentra en la memoria caché (se produce un acierto de caché) y el número total de accesos a memoria.

Otra medida de rendimiento es el tiempo de acceso medio a memoria ($\overline{T}_{\text{acc}}$). Es el tiempo que se tarda por término medio en traer un dato de memoria [1]:

$$\bar{T}_{acc} = T_{HIT} + IF * PF$$

Siendo T_{HIT} el tiempo que se tarda en acceder a memoria cuando se produce un acierto de caché. IF es el índice de fallos. Es el número de accesos a memoria por instrucción que originan un fallo de caché. PF es la penalización por fallo, es decir el tiempo que se tarda en resolver un fallo de caché.

7.8. Ejercicios

Tenéis ejercicios de este tema en la colección de exámenes resueltos disponible en el *moodle*.

Bibliografía

[1] Dormido, S., Canto M.A. Mira J. Delgado A.E. 2002. *Estructura y Tecnología de Computadores*. Sanz y Torres. 1.4, 1.4.1, 1.4.3, 1.4.3, 2.4, 2.4.1, 7.1, 7.2, 7.3, 7.4, 7.4, 7.4, 7.4, 7.4, 7.4, 7.5, 7.6, 7.7

[2] Murdocca, M.J., Heuring V.P. 2002. *Principios de Arquitectura de Computadores*. Prentice Hall. 1.3.1, 1.3.2, 1.3.3, 2.2, 2.3.5

[3] Patterson, D.A., Hennessy J.L. 1997. *Computer Organization and Design*. Morgan Kaufmann. (document), 1.1.1, 1.1.2, 1.2.2, 1.2.3, 2.3.1, 2.3.2, 2.3.3, 2.3.3, 2.3.3, 2.3.4, 3.1, 3.2, 3.3, 3.4, 3.5, 3.5.1, 3.6.1, 3.6.2, 3.7, 3.8, 3.8.2, 3.8.3, 3.9.2, 4.2.3, 4.2.4, 4.3.1, 4.3.5, 4.3.6

[4] Patterson, D.A., Hennessy J.L. 2000. *Estructura y Tecnología de Computadores*. Reverté. (document), 1.1.1, 1.1.2, 1.2.2, 1.2.3, 2.3.1, 2.3.2, 2.3.3, 2.3.3, 2.3.3, 2.3.4, 3.1, 3.2, 3.3, 3.4, 3.5, 3.5.1, 3.6.1, 3.6.2, 3.7, 3.8, 3.8.2, 3.8.3, 3.9.2, 4.2.3, 4.2.4, 4.3.1, 4.3.5, 4.3.6

[5] Stallings, W. 2000. *Organización y Arquitectura de Computadores*. Prentice Hall. 1.1.1, 1.1.2, 1.1.2, 1.1.3

[6] Wikipedia. 2009a. *Triodo — Wikipedia, La enciclopedia libre*. [Internet; descargado 15-febrero-2009]. 1.2.2

[7] Wikipedia. 2009b. *Válvula termoiónica — Wikipedia, La enciclopedia libre*. [Internet; descargado 15-febrero-2009]. 1.2.2