Tema 17: Memorias de semiconductor (2)

Objetivos:

- Introducción a las memorias Caché.
- Rendimiento de una memoria Caché.
- Capacidad de la memoria Caché.
- Principio de Funcionamiento.
- Tipos de organización.
- Algoritmos de Reemplazamiento.
- Escritura de Caché a memoria principal.
- Tipos de conexionado de la memoria Caché.
- Niveles de jerarquía en la memoria Caché.
- Conexionado de Cachés de diferentes niveles.
- Conexionado de Cachés en sistemas multiprocesador
- Protocolos de coherencia de Caché

INTRODUCCIÓN A LA MEMORIA CACHÉ

• En todo avance en la arquitectura de los computadores, se pretende mejorar el rendimiento de los mismos de manera que se ejecuten más instrucciones por unidad de tiempo.

• Para la consecución de este objetivo, uno de los aspectos a tener en cuenta será el de la memoria caché, para con ello aumentar la velocidad de la memoria.

• Mientras un procesador puede tener una frecuencia de reloj muy elevada, que le permite ejecutar un gran número de instrucciones por unidad de tiempo, la velocidad de la memoria, además de la de otros dispositivos, hace esperar al procesador para tener sus instrucciones y datos disponibles.

•En la ejecución de cualquier instrucción, el procesador realiza las siguientes funciones:

– Búsqueda de la instrucción (FETCH): La CPU realiza la búsqueda de siguiente instrucción a ejecutar en la Memoria.

– Decodificación de la instrucción : La CPU interpreta lo que debe hacer.

– Búsqueda de operandos : Nuevo acceso a memoria.

– Ejecución de la instrucción : Lo realiza internamente la CPU.

– Escritura del resultado : Nuevo acceso a memoria

INTRODUCCIÓN A LA MEMORIA CACHÉ

• Ejemplo : Para un procesador que trabaja a 1 Ghz, su periodo es de 1 ns, por lo que las etapas en las que sólo opera la CPU (2) se realizan en 2 ns, y si la memoria principal tiene un tiempo de acceso de 50 ns, empleará 150 ns en las tres etapas en las que la CPU debe acceder a memoria.

• Idealmente los tiempos deberían de ser iguales para optimizar el sistema, para ello deberíamos usar memorias muy rápidas, pero ello implicaría un elevado coste, por ello se hace necesario el uso de memorias caché que si bien cuentan con menor capacidad que la memoria principal, su velocidad es mayor.

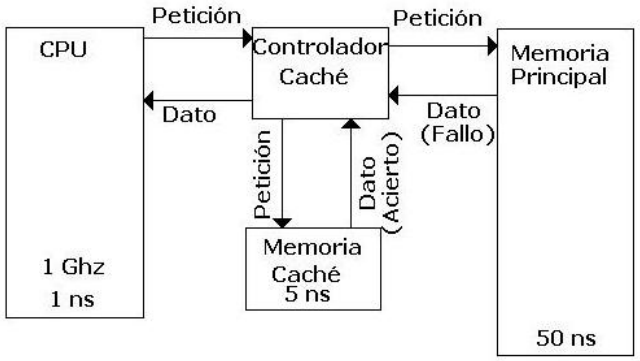
• Atendiendo a la jerarquía de la memoria, intercalaremos las memorias caché entre la CPU y la memoria principal.

• Junto con la memoria caché introduciremos otro elemento "*controlador de memoria caché"* que se encargará de atender las peticiones de código o datos de la CPU.

• Si la petición está en la caché, está será aportada a la CPU desde ella en un tiempo menor del que tomaría la memoria principal, a este hecho le llamaremos ACIERTO

• En caso contrario FALLO, el código o dato deberá ser recogido de la memoria principal.

INTRODUCCIÓN A LA MEMORIA CACHÉ



• En el caso que el dato buscado no se encuentre tampoco en la memoria principal, tendrá que ser buscado en las memorias secundarias, lo que aumentará aún más el tiempo de acceso.

• Dependiendo del tipo de conexionado de la memoria caché, la petición del dato a la memoria principal se puede hacer de dos formas:

  • Pedir a la vez el dato a la caché y a la memoria principal

  • Pedir primero a la caché y sólo en caso de Fallo acudir a la memoria Principal.

•De cualquiera de las dos formas, si la caché tiene el dato, sólo se esperará el tiempo de acceso a la caché, en caso contrario, el tiempo de acceso variará según el tipo de conexionado.

INTRODUCCIÓN A LA MEMORIA CACHÉ

• Cuando es necesario recoger los datos de la memoria principal, éstos no van únicamente a la CPU para ser ejecutados, sino que también se copian en la Memoria Caché junto con un conjunto de instrucciones *cercanas* a la deseada ya que por el principio de localidad es muy probable que las siguientes instrucciones que pida la CPU estén cerca de la actual *(vecindad espacial o localidad espacial).*

•En resumen, la memoria caché mejora la agilidad de los sistemas informáticos al reducir los tiempos de acceso a los datos de memoria y a la vez reduciendo el tráfico de datos por los buses del sistema.

RENDIMIENTO DE LA MEMORIA CACHÉ

• Rendimiento de la memoria caché : El rendimiento de una memoria caché lo mediremos a través de su tasa de aciertos (h)

$$h = \frac{\text{Nº de veces que la palabra solicitada se encuentra en la cache}}{\text{Nº Total de referencias}}$$

• Analizando la ecuación anterior observamos que "h" es la probabilidad de obtener éxito en una petición de la CPU $\frac{\text{Nº de aciertos}}{\text{Nº de aciertos + Nº Fallos}}$

• Éste parámetro se puede medir experimentalmente ejecutando unos ciertos programas representativos y calculando el número de aciertos y fallos en un determinado intervalo de tiempo.

• En sistemas correctamente diseñados la tasa de aciertos ronda el 0.9.

• Lógicamente la tasa de fallos será 1-h.

• Tiempo de acceso medio t_a: El tiempo medio de acceso a memoria teniendo en cuenta los accesos a caché con éxito y a la memoria principal en caso de Fallo serán: $t_a = h \bullet t_{ca} + (1-h) \bullet t_p$

siendo t_{ca} el tiempo de acceso medio a memoria cache

siendo t_p el tiempo de acceso medio a memoria principal

• *Suponiendo que se accede a memoria principal una vez que la caché ha fallado*

RENDIMIENTO DE LA MEMORIA CACHÉ

• Factor de velocidad : Es la relación entre el tiempo de acceso a la memoria principal y el de acceso a la caché.

$$\gamma = \frac{t_p}{t_{ca}}$$

• Se puede introducir ahora un cierto índice de mejora de un sistema informático al añadirle una memoria caché

$$\lambda = \frac{t_p}{t_a} = \frac{t_p}{h \bullet t_{ca} + (1-h) \bullet t_p} = \frac{1}{h \bullet \frac{t_{ca}}{t_p} + (1-h) \bullet 1}$$

$$\lambda = \frac{1}{h \bullet \frac{1}{\gamma} + 1 - h} = \frac{1}{1 - h \bullet (1 - \frac{1}{\gamma})}$$

• Cuando h = 0 todos los accesos se hacen a la memoria principal y λ = 1, a partir de ese valor, según aumenta h (número de aciertos), λ ira incrementando su valor, hasta llegar a su máximo cuando h=1 y entonces

$$\gamma = \lambda$$

• Por lo que lógicamente el índice de mejora depende de una forma muy directa de la tasa de aciertos en los accesos a caché, es decir, de h

CAPACIDAD DE LA MEMORIA CACHÉ

• Como se ha comentado ya, según aumentamos la capacidad de una memoria, ésta se hace más lenta principalmente por la mayor complejidad de la circuitería que lleva asociada.

• Por una parte, el principal objetivo de introducir memorias caché es el de dar mayor agilidad al sistema.

•Por otra parte, si tengo una caché más grande, podré almacenar mas información dentro de ella, con lo que la tasa de acierto h aumentará, reduciendo de esa forma el tiempo de acceso medio a memoria.

•También existe limitaciones de espacio físico para implementar este tipo de memorias ya que habitualmente vienen integradas en el mismo I.C. del procesador al que sirven los datos.

•Es muy complicado el dar un valor óptimo de la memoria caché que debe llevar un procesador de propósito general.

PRINCIPIO DE FUNCIONAMIENTO

• ***REGLA 80/20 :*** Aproximadamente el 20% de todos los programas y datos presentes en un ordenador son requeridos por el procesador el 80% del tiempo.

• El sentido del uso de la memoria caché se basa en la regla 80/20 ya que siempre existe una alta posibilidad de que la CPU necesite aquellos datos que ha usado hace poco.

• Las peticiones de instrucciones y datos por parte de la CPU no es diferente si está presente una memoria caché o no lo está, las peticiones siempre las hará a través del bus de control y del Bus de direcciones, y la única diferencia será la velocidad a la que le son servidos las instrucciones o los datos.

• Los componentes más importantes de la caché son:

- ***El controlador de caché***
- ***El Bloque de etiquetas***
- ***Memoria de datos caché***
- ***Lógica de Control***

PRINCIPIO DE FUNCIONAMIENTO

• ***El controlador de caché :*** Es el elemento encargado de controlar y regular el funcionamiento de toda la memoria caché, controlando el flujo de los datos, los algoritmos que emplea para predecir los datos que debe intercambiar con la memoria principal y los que debe servir a la CPU.

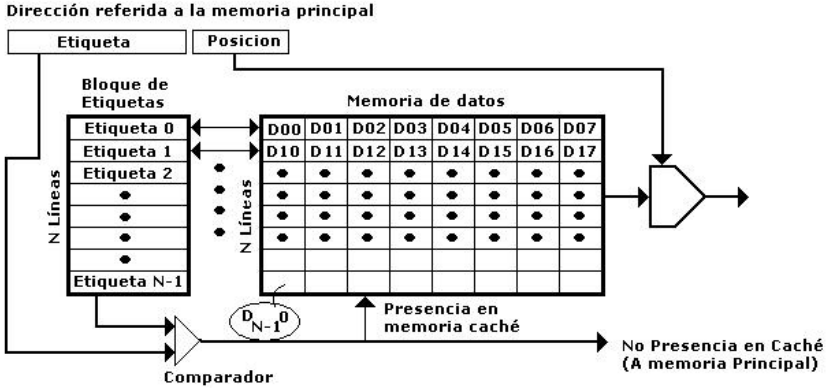
• ***El Bloque de etiquetas :*** Se trata de una memoria RAM de acceso por contenido (CAM), no por dirección. Contiene una lista de etiquetas que hacen referencia a las direcciones de la memoria principal cuyos datos están presentes en la memoria caché. Su misión dentro de la memoria caché será la de decir si la petición de la CPU se encuentra en la caché o no lo está.

• ***Memoria de datos caché :*** Se trata de una memoria SRAM de reducido tamaño pero muy rápida en la que se almacena la información presente en la memoria principal que se prevé que va a ser solicitada por la CPU en breve.

• ***Lógica de Control :*** Es la lógica encargada de realizar la conexión entre los diferentes elementos antes comentados, así como entre dichos elementos y la CPU o resto de elementos.

PRINCIPIO DE FUNCIONAMIENTO

• Una posible organización de una memoria caché puede ser la siguiente:



• Vamos a considerar la memoria principal dividida en bloques de longitud fija k, es decir, si nuestro bus de direcciones está compuesto por n bits, podremos llegar a direccionar hasta 2^n datos, pero además estarán divididos en $2^n/k$ bloques.

• Si los bloques son contiguos, pueden ser diferenciados por la parte alta de su dirección, por ejemplo con 8 bits de direcciones la memoria principal iría desde 00h hasta FFh, podré considerar por ejemplo 16 bloques con 16 direcciones físicas cada uno de esos bloques, es decir, tendremos un bloque que abarque desde la dirección 00h a la 0Fh, el siguiente desde 10h hasta 1Fh …etc, y a cada uno de ellos le asigno la etiqueta correspondiente a la cifra hexadecimal de más peso de su dirección.

PRINCIPIO DE FUNCIONAMIENTO

• De esta forma tenemos el bloque 0 que abarca desde la dirección 00h hasta 0Fh, y así sucesivamente.

•Si la CPU solicita la información presente en la dirección 3Bh de la memoria principal, ésta petición llegará a la memoria caché que investigará si tiene presente el bloque 03.

•En caso afirmativo, el resto de la dirección solicitada nos permitirá seleccionar el Dato que debe ser entregado a la CPU de entre todos los del bloque al que pertenece, lógicamente en un tiempo bastante inferior del que hubiera necesitado accediendo a memoria principal.

•En caso negativo, se acudirá a memoria principal para poder servir el dato a la CPU, y además se traerá el bloque al que pertenece a la memoria caché ya que por el principio de localidad, es probable que la CPU vuelva a solicitarlo o lo haga con alguno cercano a él.

•El dato puede ser de diferentes tamaños (1 bit, 1 byte …etc).

Tipos de Organización

• El tipo de organización permite definir como se van a almacenar en la caché los datos provenientes de la memoria principal, y de ellos dependerá la rapidez de acceso del microprocesador a los datos, la cantidad de RAM que podrá gestionar la caché …etc…

•En todos los casos, la CPU referencia a la caché con la dirección de la memoria principal.

•Básicamente la organización de la memoria caché consiste en establecer la correspondencia entre la memoria principal y la memoria caché

• Existen principalmente tres tipos de organización:

  • Directa.

  • Totalmente asociativa.

  • Asociativa por conjuntos.

Tipos de Organización

- ***ORGANIZACIÓN DIRECTA u ORGANIZACIÓN DE UNA VIA :***

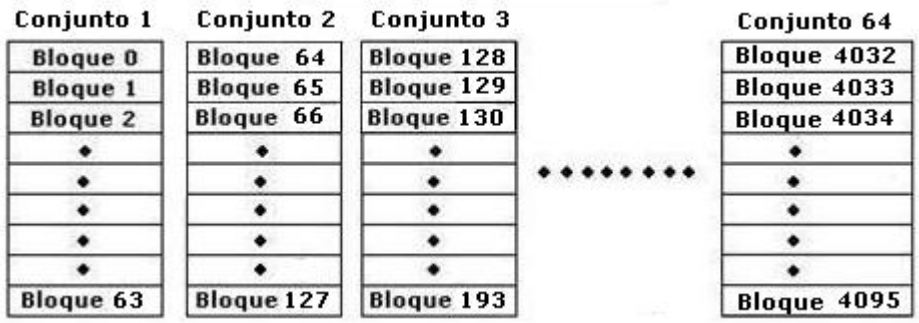
  - Es la técnica más simple de todas y consiste en dividir la memoria principal en conjuntos de bloques consecutivos del tamaño de la memoria caché disponible.

  - Se hace una correspondencia directa entre la posición de cada bloque dentro de cada conjunto de la memoria principal y la posición que dicho bloque puede tomar en la caché, es decir, un bloque que ocupe la segunda posición en uno de los conjuntos de la memoria principal, sólo podrá ir al segundo puesto de la caché, y dicho puesto de la caché sólo podrá ser ocupado por los bloques que ocupen la misma posición relativa dentro de cada conjunto en que se divide la memoria principal.

Tipos de Organización

•Como ejemplo para ilustrar esta organización, dispondremos de una memoria principal de 32KB, por lo que tendremos un bus de direcciones de n = 15 líneas.

- •La memoria caché podrá almacenar hasta 512 bytes = 29 bytes.
- •La memoria caché está organizada en bloques de k=8 bytes/bloque, por lo que la cantidad de bloques de la memoria caché son 512/8 = 64 bloques.
- •En nuestro ejemplo, dividimos nuestra memoria principal de 32KB en 32KB/512B 64 conjuntos de 64 bloques cada uno y con 8 bytes por bloque.



• El contenido de la memoria caché puede ser cualquiera de los mostrados en la figura, pudiendo sustituir cada bloque de cada conjunto por otro bloque que ocupe la misma posición relativa en otro conjunto.

Tipos de Organización

- Ejemplo 2:
  - Datos:
    - Tamaño de la memoria principal : 4 GB = 4096 MB = 4194304 KB = 4294967296 Bytes. (bus de direcciones de 32 bits).
    - Tamaño de la memoria caché : 4KB = 4096 Bytes.
      - Dividiremos la memoria principal en 4294967296/4096= 1048576 (1 Mega de grupos) conjuntos de 4KB consecutivos cada uno de esos conjuntos
      - Por lo que necesitaremos los 20 bits de mayor peso para diferenciar el conjunto al que pertenece cada uno de los bloques presentes en la caché en cada momento.
      - Si los bloques de la memoria caché son de k=1 byte por bloque, necesitaremos los 12 bits restantes del bus de direcciones para localizar el bloque dentro del conjunto que a su vez coincidirá con el byte buscado.
      - Si los bloques de la memoria caché son de k=2 bytes por bloque, necesitaremos los 11 bits restantes del bus de direcciones para localizar el bloque dentro del conjunto y el último bit para seleccionar entre los 2 bytes del bloque….

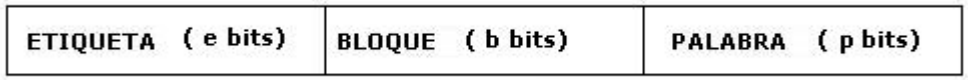
Tipos de Organización

•Si *C* es el tamaño de la memoria Caché y *P* el de la memoria principal, el número de conjuntos en los que dividiremos a la memoria principal será *P / M*, y por lo tanto, necesitaremos e = Log_2 (P/M) bits de mayor peso del bus de direcciones para saber que bloques de los presentes en caché pertenecen al conjunto dosnde se encuentra el bloque buscado.

•Si K es el número de bytes presentes en cada bloque, necesitaremos p=Log_2 (K) bits de menor peso para poder elegir el byte correcto dentro del bloque seleccionado.

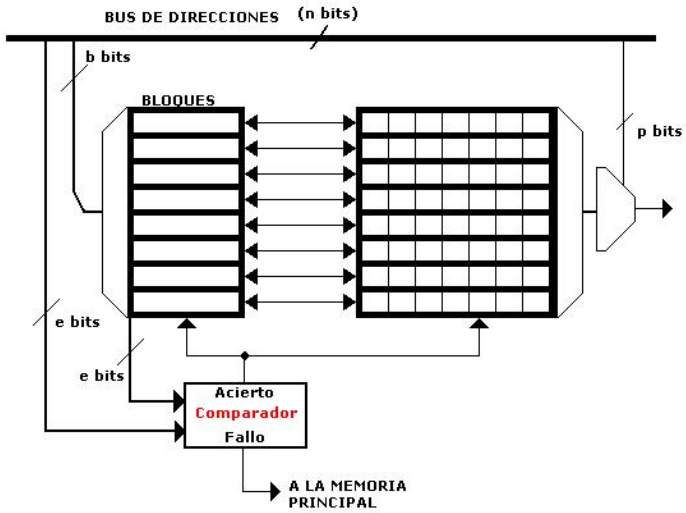
•Para la selección del bloque correspondiente, usaremos los b = n – b – e bits restantes siendo n el número total de bits del bus de direcciones.

• De esta forma, la caché dividirá el bus de direcciones en tres campos:



Tipos de Organización

• Cuando la CPU solicita un dato, dicha petición llega a la memoria caché, donde la dirección es decompuesta en los tres campos, la parte alta (Etiqueta) se compara con las etiquetas de conjunto de cada uno de los bloques presentes en la caché en ese momento



• En caso de producirse un acierto, el dato será buscado, localizando en primer lugar el bloque que lo contiene y posteriormente se separará dicho dato del resto de componentes del bloque.

• Si ocurre un fallo, el dato se buscará en la memoria principal, sustituyendo el bloque que ocupe su misma dirección relativa dentro del grupo dentro de la caché.

Tipos de Organización

• Este tipo de organización de la memoria, tiene sus ventajas y sus inconvenientes, por una parte, no necesitaremos ningún algoritmo de reemplazo del contenido de la caché, ya que en el caso de producirse fallo, la posición donde debería hallarse el bloque buscado será sustituida por él, ya que sólo puede ir a esa localización.

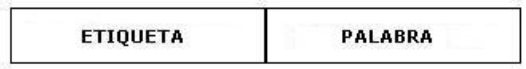
• Pero el mismo aspecto también es la principal desventaja de éste tipo de memorias, ya que si el programa está constantemente accediendo a contenidos lejanos en la memoria principal, se estará constantemente modificando todo el contenido de la memoria caché disminuyendo así el rendimiento del sistema al aumentar los tiempos de acceso.

Tipos de Organización

ORGANIZACIÓN TOTALMENTE ASOCIATIVA :

• En la correspondencia totalmente asociativa, cada bloque de memoria principal puede ubicarse en cualquier bloque o línea de la memoria caché, independientemente del resto de bloques, es decir, lo que se intenta es subsanar la desventaja vista en el caso anterior.

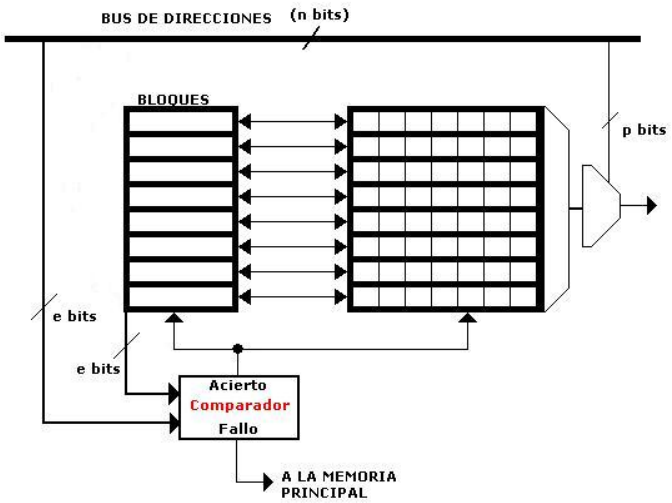
• Cuando la CPU genera la dirección de memoria a la que quiere acceder, dicha dirección se divide en 2 campos:



• El número de bits de cada uno de estos campos viene dado por las siguientes relaciones (para el ejemplo anterior):

$$\text{Campo palabra} = p = \log_2(K)$$

$$\text{Campo etiqueta} = e = n - p$$



• En este tipo de organización la etiqueta se compara con todas las etiquetas de los bloques almacenados en la memoria caché, haciendo cada comparación e un comparador diferente.

Tipos de Organización

•En caso de éxito, mediante el campo palabra se localiza el dato pedido dentro del bloque correspondiente y es entregado a la CPU.

•En caso de que todas las comparaciones fallen, se buscará el dato en la memoria principal.

• También este tipo de organización de la memoria, tiene sus ventajas y sus inconvenientes, por una parte, necesitaremos algoritmos de reemplazo del contenido de la caché para decidir en cada momento que bloques se quedan en la caché o cuales se traen desde la memoria principal, éste aspecto hace que el diseño sea más complejo en este tipo de organización.

• También es más complicada la circuitería al necesitar mayor número de comparaciones.

•Por otra parte, evitamos el problema de la organización Directa ya que en ésta se pueden sustituir los bloques por separado.

Tipos de Organización

ORGANIZACIÓN ASOCIATIVA POR CONJUNTOS ORGANIZACIÓN DE N VIAS :

• Este tipo de organización es una solución intermedia entre la organización directa y la totalmente asociativa.

• Como en los otros tipos de organizaciones, tanto la memoria principal como la memoria caché se divide en bloques de igual tamaño.

• Pero en este caso, los bloques de la memoria caché se agrupan en conjuntos, y al número de bloques por conjunto se le llama número de vías.

•En la memoria principal, los bloques también se agrupan en conjuntos, del mismo tamaño, (número de vías), que los conjuntos de la memoria caché.

• Cada bloque de la memoria principal puede ubicarse en cualquier vía de la memoria caché dentro del conjunto asociado a ese bloque, en este aspecto sería una relación totalmente asociativa.

•La posición de cada conjunto dentro de la caché esta predeterminado, por lo que en este aspecto sería una relación directa

Tipos de Organización

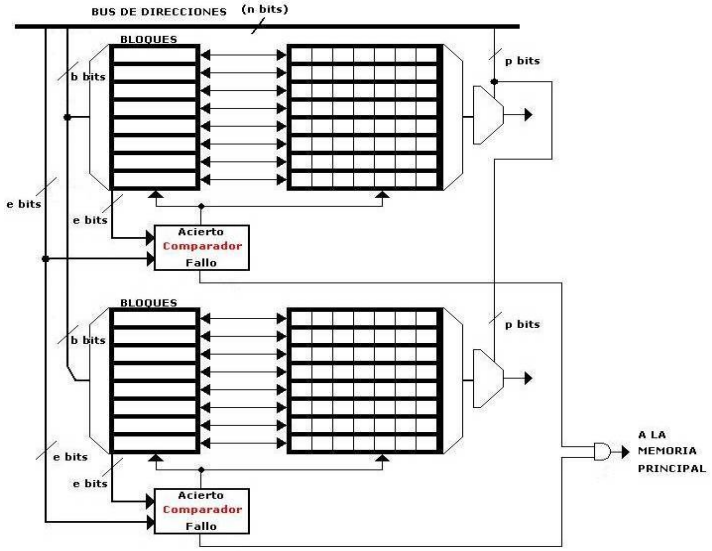
Observar:

- Si el número de conjuntos de la memoria caché es uno, estamos en el caso de la organización totalmente asociativa.

- Si el número de vías, es decir, el número de bloques por conjunto de la memoria caché es uno, estamos frente a la organización Directa, es decir, los casos anteriores son casos particulares de éste tercero.

- Cuando la CPU solicita una dirección, se busca el conjunto que la debe contener dentro de la memoria caché con tantos comparadores como conjuntos hay en caché, en caso de éxito, el datos será buscado dentro del conjunto de la misma forma que en la organización totalmente asociativa.

- En caso de fallo, el dato será buscado en la memoria principal.



Algoritmos de reemplazamiento

• Cuando un nuevo bloque se transfiere a la memoria caché, lo hace en sustitución de uno de los ya presentes en la misma.
• En organización directa, la posición que debe ocupar un bloque importado de la memoria caché está predeterminado.
• Pero en las otras dos organizaciones, necesitaremos implementar un algoritmo que se encargue de gestionar esas sustituciones.
• Dicha implementación debe ser realizada a nivel de Hardware.
• Idealmente, se podría suponer que ese bloque que nos hemos visto obligados a buscar en la memoria principal, no va a ser requerido de nuevo, por lo que nos evitaríamos la necesidad de trasladarlo a la caché, pero este dato no lo podemos conocer a priori, por lo tanto debemos implementar los algoritmos de sustitución.
•Los algoritmos de sustitución los podemos clasificar en dos grandes bloques:

- Los que están basados en los grados de utilización de los bloques.
- Los que NO están basados en los grados de utilización de los bloques.

•En ambos casos debemos tener presente que la complejidad de cada algoritmo, determinará la complejidad del Hardware a implementar.

Algoritmos de reemplazamiento

• Como ya hemos comentado, en una organización directa, la posición que debe ocupar un bloque importado de la memoria caché está predeterminado, por lo que no cabe implementar ningún algoritmo.
• En el caso de las organizaciones totalmente asociativas, el bloque puede llevarse a cualquier posición de la caché, por lo que cualquier algoritmo de reemplazo será válido.
• El caso de una organización de N vías, como es el caso de los procesadores Pentium, existen diferentes posibilidades, siendo los más comunes:

- •RANDOM
- •LRU
- •LFU
- •FIFO

Algoritmos de reemplazamiento

- **ALGORITMO DE SUSTITUCION RANDOM**
  - Este algoritmo toma de forma aleatoria un bloque de la memoria caché y lo sustituye por el bloque de la memoria principal que contiene la información a la que se acaba de acceder.
  - Sus principales ventajas son la facilidad de implementación y su rapidez.
  - Su principal desventaja es que no se tiene ningún cuidado al decidir que bloque es desalojado de la memoria principal, esto conlleva, sobre todo para el caso de memorias caché pequeñas, una alta probabilidad de estar desalojando un bloque que va a ser solicitado en breve.
- El mayor problema a la hora de diseñar los algoritmos de reemplazamiento, es el de *predecir* lo que nos va a solicitar la CPU antes de que ésta lo haga.
- Una posible estrategia es la de pensar que aquel bloque que lleve más tiempo en la caché si ser solicitado por la CPU será el que menos probabilidades tenga de ser solicitado en breve, y por ello será el primero en ser desalojado cuando necesite importar un bloque de la caché, esto nos lleva al segundo algoritmo (LRU)

Algoritmos de reemplazamiento

- **ALGORITMO DE SUSTITUCION LRU *(Least Recently Used)***
  - Este algoritmo también llamado (Algoritmo del bloque utilizado menos recientemente) se basa en la idea de que un bloque que no se ha utilizado en un largo periodo de tiempo tiene una probabilidad menor de ser utilizado en un futuro inmediato según el principio de localidad temporal.
  - Una forma de implementar este algoritmo es adjudicar un contador a cada bloque presente en la memora caché.
  - Debemos tener en cuenta que los contadores tienen un número limitado de bits, por lo que el algoritmo podría fallar en caso de que algún contador llegue a desbordarse, con lo que pasaría a contar 0
  - Desde este punto de vista, interesaría implementar algoritmos con módulos muy altos.
  - Pero por otra parte, al aumentar el módulo del contador, también estamos complicando el HW de control de la caché y por lo tanto haciéndola más lenta

Algoritmos de reemplazamiento

• Para lograr reducir el módulo de los contadores a utilizar, haremos las siguientes modificaciones en el algoritmo:

- •En caso de acierto:
  - • El contador asociado a ese bloque se pone a cero.
  - • Todos los que tenían un valor menor se incrementan en una unidad.
  - • Todos los que tenían un valor mayor se decrementan en un unidad.
  - •*Con esto me aseguro que en caso de acierto, nunca se desbordará ningún contador ya que si tenemos alguno al límite de su módulo, en este momento disminuirá su cuenta en una unidad.*
- • En caso de fallo:
  - • Si la memoria caché no estaba llena, el contador correspondiente al nuevo bloque se inicializa a cero y el resto se incrementa en una unidad.
  - • Si la memoria caché estaba llena, se busca el contador más alto en ese instante, sustituyendo su contenido por el buscado por la CPU y se inicializa a cero, incrementando el resto de contadores.

Algoritmos de reemplazamiento

• Cuando se produce una fallo, podemos diseñar el algoritmo LRU para que funcione de dos formas diferentes:

- •***EL DATO PEDIDO VA EN PRIMER LUGAR :*** El dato requerido pasa directamente de la Mp a la CPU y posteriormente, mientras el procesador ya está ocupado, se sustituye la línea de la caché, esto mejora el rendimiento pero complica la circuitería.
- •***EL DATO PEDIDO VA EN ÚLTIMO LUGAR :*** Se sustituye la línea de la caché y posteriormente la cpu extrae el dato de la caché, la circuitería es más simple, pero tenemos más tiempo al procesador esperando.

ALGORITMO DE SUSTITUCION LFU ***(Least Frecuently Used)***

- • Este algoritmo también llamado (Algoritmo del bloque utilizado menos frecuentemente) se basa en la idea de que un bloque que no se ha utilizado de forma frecuente, no tiene porque estar en caché.
- • Para ello asociaremos a cada bloque de la memoria caché un contador que se incremente cada vez que la CPU accede a ese bloque.
- • A la hora de decidir el bloque a sustituir en la caché se elegirá aquel cuyo contador tenga el valor menor
- •Es un método poco usado por su complejidad de implantación a nivel HW y por penalizar en exceso a bloques recién llegados a la caché.

Algoritmos de reemplazamiento

ALGORITMO DE SUSTITUCION FIFO

•Este algoritmo también llamado (Primero en entrar primero en salir) se basa en la idea de sustituir en cada momento aquel bloque que lleve más tiempo en caché.

• Es sencillo de implementar a nivel de HW, pero no tiene en cuenta las probabilidades de que un bloque sea requerido a continuación.

Escritura de caché a memoria principal.

Escritura de caché a memoria principal.

• Cuando se va a sobreescribir un bloque de la memoria caché, hay que plantearse previamente si el bloque que se va a desalojar está salvado en la memoria principal, para no perder las modificaciones que pueda haber experimentado .

• También es importante tener en cuenta la coherencia entre los datos de la memoria caché y la principal:

  • Supongamos que un byte acaba de ser modificado en la memoria caché, pero aún no se ha refrescado dicho contenido en la principal, y la CPU da orden a un periférico de recoger ese dato de la memoria principal, obviamente, recogerá un dato sin actualizar, lo cual redundará en un mal funcionamiento del programa.

•Ambos aspectos deben ser tenidos en cuenta a la hora de decidir la forma en que se devolverán los datos de la memoria caché a la principal.

•El problema se amplia cuando en el sistema existen más de un elemento con memoria caché, (por el momento no consideraremos esta opción).

Escritura de caché a memoria principal.

• Las formas más habituales de hacer estas actualizaciones son:

ACTUALIZACIÓN POR ESCRITURA INMEDIATA (Write Through)

Consiste en que cada vez que la CPU realice una operación de escritura en la memoria caché, también lo haga en la memoria principal, con lo que aseguramos que esta última está siempre actualizada. La principal desventaja de éste método es el alto tráfico de datos lo que ralentiza el sistema.

ACTUALIZACIÓN POST-ESCRITURA (Write Back)

Las acciones de escritura sólo se realizan en la caché, y en ese momento se pone a uno un bit adicional que añadimos a cada bloque para saber si ha sido modificado.

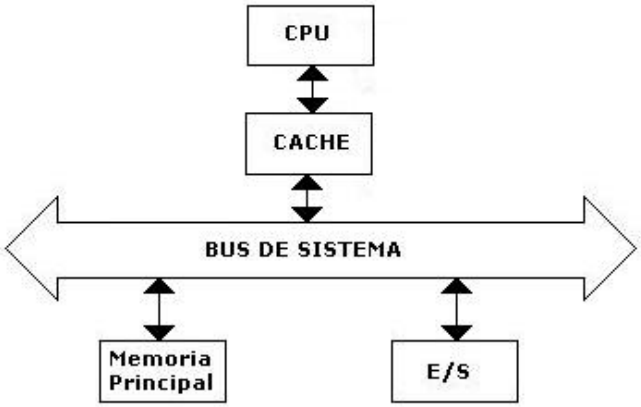
El bloque será actualizado en la Memoria principal, sólo cuando vaya a ser desalojado de la memoria caché y siempre y cuando tenga ese bit a 1.

Esta técnica también implica que todos los accesos de periféricos se tengan que hacer a la caché o bien obligar a la actualización de la memoria principal cuando se detecte esta situación.

Tipos de conexionado de la memoria caché

• Básicamente la Memoria Caché se puede conectar con la CPU en serie o en paralelo.

CONEXIÓN EN SERIE :



• La CPU se conecta directamente a la caché y sólo con ella, por lo que todas las peticiones de la CPU a Otros dispositivos se hacen a través de la memoria caché

• **VENTAJAS :**

• Cuando el dato solicitado está almacenado en la caché el acceso de la CPU a dicha información es muy rápido.

•Además, si el dato está en caché, no ocupo el bus del sistema, pudiendo trabajar los dispositivos de E/S con la memoria principal sin interferir ambos trabajos.

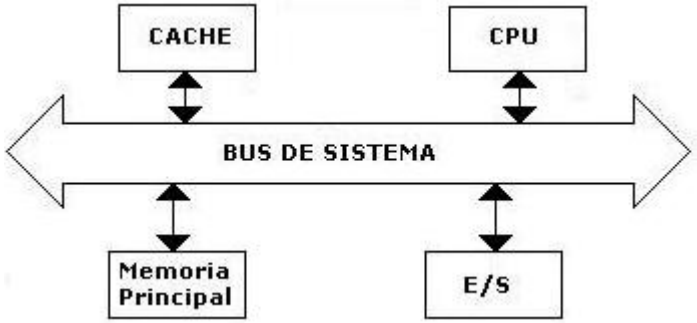
Tipos de conexionado de la memoria caché

- **DESVENTAJAS**

  • Todo tiene que pasar a través de la caché que no puede ser desconectada si no interesa tenerla.

  • Otro inconveniente es la penalización en el tiempo que se sufre cuando la información que necesita la CPU no se encuentra en la caché, teniendo que acceder a la memoria principal a través del bus del sistema, sumándose ambos tiempos, ya que no se puede hacer la petición de forma simultánea a ambas memorias.

CONEXIÓN EN PARALELO:



• La CPU y la caché están conectadas a trvés del bus del sistema así que todo el tráfico de información con la CPU se hace a través del Bus de sistema.

• **DESVENTAJAS**

Esta conexión sobrecarga de tráfico el bus del sistema, por lo que el rendimiento en general disminuye.

Tipos de conexionado de la memoria caché

- **VENTAJAS**

  • Nos permite usar la caché de forma opcional, por lo que es una configuración mucho mas flexible, pudiendo añadirla o eliminarla sin alterar el funcionamiento del sistema.

  •Las peticiones de la CPU son enviadas tanto a la memoria caché como a la principal, por lo que los tiempos de retardos no se suman. Si la información se encuentra en caché es entregada a la CPU en poco tiempo a la vez que detiene la búsqueda en la memoria principal.

Niveles de jerarquía en la memoria Caché

• Entre los mecanismos para mejorar el rendimiento del sistema, podemos disminuir el tiempo de acceso a las caches y/o aumentar la tasa de aciertos.

• Disminuir el tiempo de acceso a una memoria caché es complicado debido a limitaciones tecnológicas, es más sencillo aumentar la tasa de aciertos, y para ello podemos mejorar los algoritmos de carga en caché y/o aumentar el tamaño de la caché.

• Respecto al segundo punto, se emplean diferentes niveles de memoria caché, *se jerarquiza la memoria caché :*

CACHÉ DE NIVEL 1, CACHE L1, CACHE PRIMARIA

Es una memoria caché integrada en la propia CPU, y suele funcionar a la misma velocidad que ésta, se comenzó a utilizar en los 486 con un tamaño de 8KB.

• ***CACHÉ DE NIVEL 2, CACHE L2, CACHE SECUNDARIA***

Lígeramente mas lenta y de mayor tamaño que la de nivel 1, se le puede encontrar tanto integrada con la CPU o exterior en la placa madre, contiene toda la caché de L1 más los datos propios de sus algoritmos de carga.

• ***CACHÉ DE NIVEL 2, CACHE L2***

Como muchos micros actuales integran parte de la caché L2 en la CPU, a la parte que queda en placa se le llama así

Niveles de jerarquía en la memoria Caché

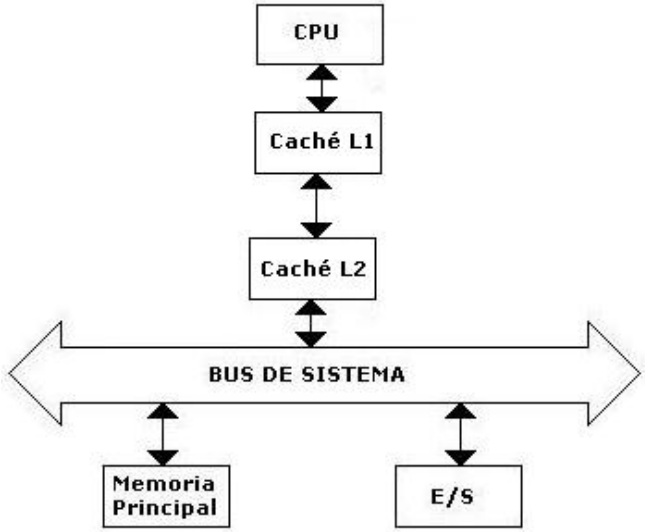
• Los niveles de caché deben cooperar entre ellos con el objetivo común de optimizar el rendimiento del sistema, por ello tenemos dos tipos de funcionamiento:

- **INCLUSIVA:** Ambas niveles tienen la misma información algo más ampliada en el asegundo nivel
- **EXCLUSIVA:** Ambos niveles, L1 y L2 trabajan como si se tratase de una sola memoria caché

Conexionado de Cachés de diferentes niveles.

• Existen dos posibilidades: Conexionado serie y Conexionado paralelo.

•**CONEXIONADO SERIE:**



• **Ventaja:** Disminuye el tráfico en el bus del sistema, por lo que éste se puede encargar de otras tareas.

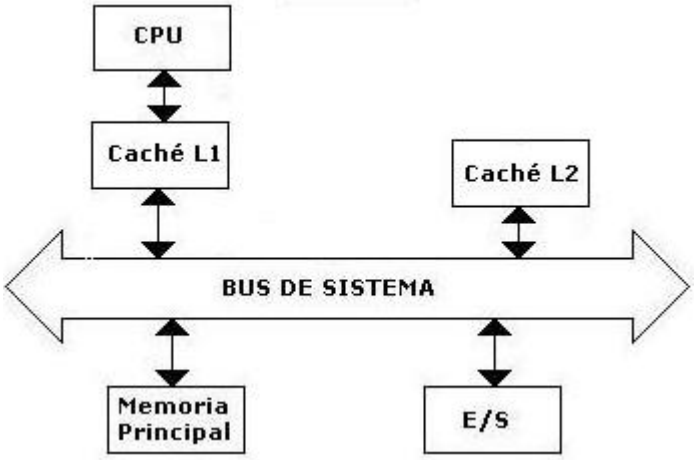
• **Desventaja:** Cualquier interactuación con la CPU debe pasar por caché, siendo ambos niveles de caché obligatorios, incluso si están en placa.

Además los retardos se van acumulando, ya que una petición de la CPU es primeramente buscada en la Caché L1 que tendrá su tiempo de respuesta, Sólo transcurrido éste, comenzará la búsqueda en el L2 con un tiempo de respuesta superior que se suma al anterior.

En el caso de que ambas caché den fallo, aún tendremos que sumar el tiempo de acceso a la memoria principal, que no es consultada hasta que ambas caché dan Fallo.

Conexionado de Cachés de diferentes niveles.

CONEXIONADO PARALELO:



•**Ventaja:** La Caché de L2 es optativa, pudiendo ser eliminada.

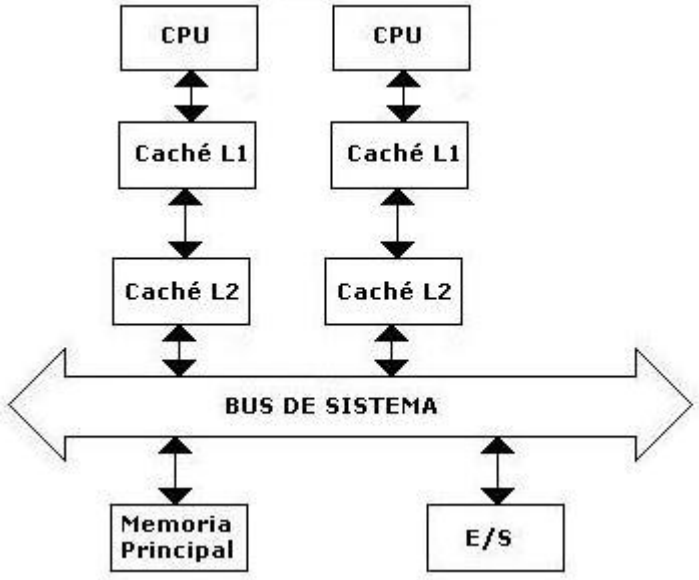
En caso de que la caché L1 de fallo, la petición a L2 se hace al mismo tiempo que la petición a la memoria principal, por lo que en caso de que L2 también de fallo, los retardos no se sumarán

• **Desventaja:** El tráfico por el bus de sistema es mayor, ya que las peticiones a la caché L2 se hace a través de él.

• Conexionado de Cachés en sistemas multiprocesador

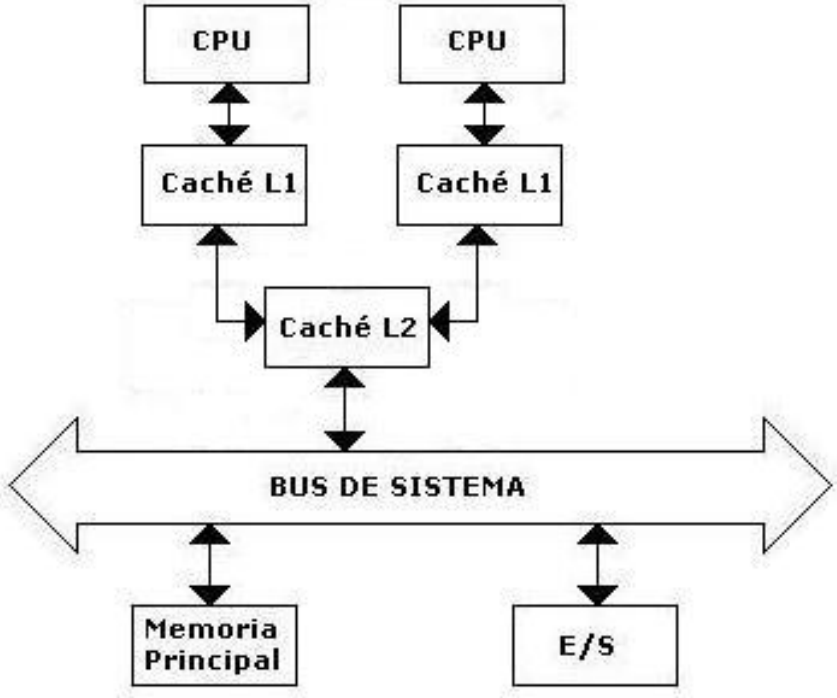
• Existen diferentes formas de realizar el conexionado de las memorias caché cuando se dispone de un sistema con varios procesadores:

•**CONEXIONADO SIMPLE**: Cada procesador tiene su propia jerarquía de cachés y cada jerarquía utiliza el protocolo MESI, el problema es la alta utilización del bus del sistema



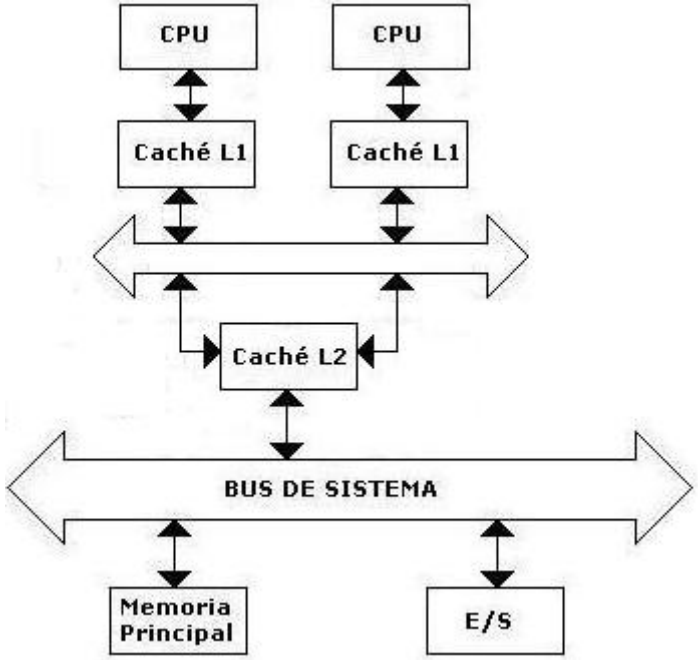
• Conexionado de Cachés en sistemas multiprocesador

CONEXIONADO MULTIPUERTO: Es el sistema más utilizado, donde la caché L2 es multipuerto, por lo que permite que varias cachés L1 accedan a la misma caché L2, con ello liberamos la ocupación que teníamos del bus del sistema con en conexionado simple, pero aumentando el coste de la caché de L2 que debe ser multipuerto.



• Conexionado de Cachés en sistemas multiprocesador

CONEXIONADO POR BUS: También se trata de un sistema altamente utilizado, parecido al multipuerto, pero en vez de usar caches L2 multipuerto, se emplean *sub-buses* para realizar las interconexiones entre L1 y L2, si bien el coste es menor al no requerir caches L2 multipuerto y mantenemos la baja ocupación del bus principal del sistema, se trata de una organización más lenta que la anterior, ya que en cada momento sólo puede acceder una caché L1 a la caché L2



Protocolos de coherencia de Caché

- Un sistema multiprocesador es aquel en el que varios procesadores comparten recursos comunes como pueden ser Buses, Memorias, periféricos ….etc….
- Un problema grave se puede producir por el hecho de que dos procesadores compartan la misma memoria principal.
- Supongamos que un procesador encargado de cálculos recoge en su memoria caché un dato de la memoria principal para actualizarlo, por otra parte tenemos un segundo procesador encargado de gestionar las comunicaciones con otros sistemas y que quiere enviar dicho dato por la red. Si la actualización del primer procesador no ha sido refrescada en la memoria principal, el dato enviado por el segundo será antiguo.
- Para evitar este tipo de problemas es necesario poner en práctica alguna técnica que permita implementar una caché especial para cada uno de los procesadores con el fin de mantener la coherencia.
- Para ello podemos hacer uso de protocolos que se encarguen de mantener dicha coherencia utilizando bits de estado
- Uno de estos protocolos, muy usado sobre todo en conexionado simple es el protocolo MESI

Protocolos de coherencia de Caché

- **Protocolo MESI (Modified Exclusive Shared Invalid)** Desarrollado por INTEL.

  El protocolo MESI se basa en cuatro estados (M, E, S, I) en los que puede estar cada una de las líneas de una memoria caché y dependiendo de los distintos eventos se producirán transiciones entre los estados.

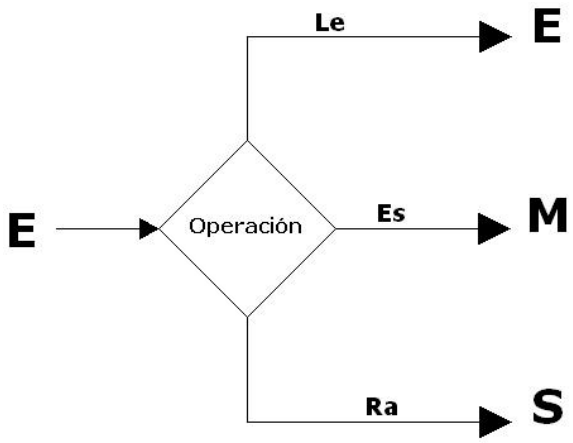
- Los estados son:
  - **M (Modificado) :** Indica que esa línea está modificada por una escritura del procesador y a la espera de ser actualizada en la memoria principal si es preciso, además indica que ninguna otra caché contiene esta línea.
  - **E (Exclusiva) :** Indica al controlador de la caché que la línea de la memoria principal es coherente con la caché y que ninguna otra caché contiene esa linea.
  - **S (Simultáneo o Compartido) :** Indica que dicha línea está repetida (Compartida) en otra/s caché/s. Si se escribe en una de ellas, el resto quedarán invalidadas inmediatamente.
  - **I (Inválido) :** Indica que la línea no es válida y puede ser sustituida y si el procesador la solicita, deberá acudir a la memoria principal, y si el procesador quiere escribirla, se actualizará inmediatamente en la memoria principal

Protocolos de coherencia de Caché

- Cuando una línea se carga por primera vez en una caché, se le marca como (E – Exclusiva) Indicando que esa línea no se encuentra en ninguna otra caché.
  - Si se realizan operaciones de lectura, se mantendrá el estado E.
  - Sólo en caso de realizarse una operación de escritura, variará el estado de E a (M – Modificado).
  - Si estando la línea marcada como E, es solicitada por otra caché, cambiará su estado de E a (S – Compartida) en ambas cachés.
  - Para verlo de forma gráfica, definiremos :

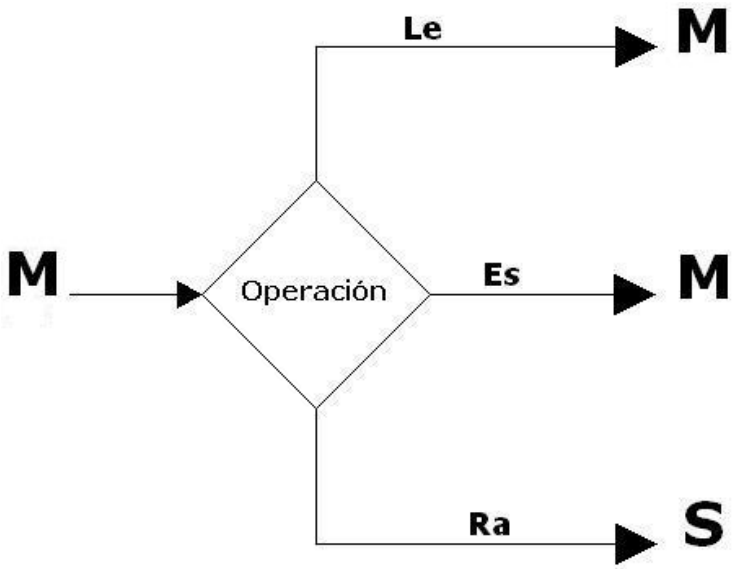
ESTADO		OPERACIÓN	
M	Modificado	Le	Lectura
E	Exclusivo	Es	Escritura
S	Compartido	Ra	Rastreo
I	Invalido		

* Ra : Cuando una caché busca una línea en otras cachés



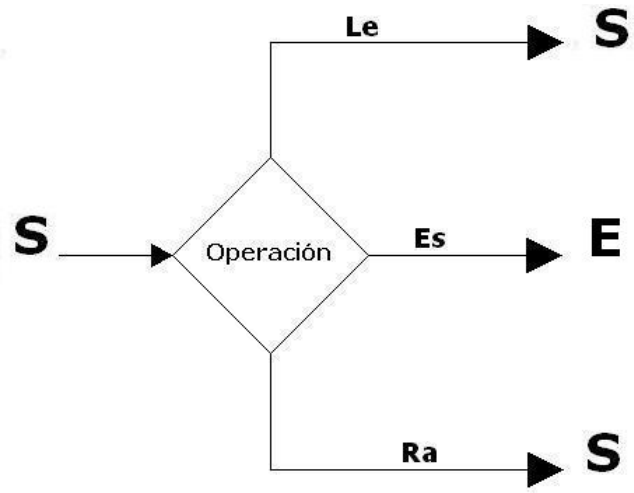
Protocolos de coherencia de Caché

- Si estando marcada una línea con M es solicitada por otra caché, se realizará una escritura obligatoria en la memoria principal y su estado pasará a S en ambas cachés, en caso de ser accedida por el procesador tanto en lectura como en escritura, su estado permanecerá en M.



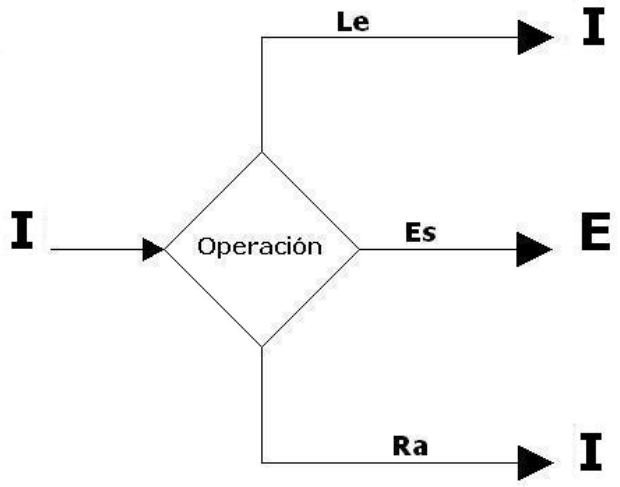
Protocolos de coherencia de Caché

- Si estando marcada una línea con S es solicitada por otra caché (Ra) o leida por alguno de los procesadores (L), se mantendrá su estado S.
- Si estando marcada una línea con S se escribe desde uno de los procesadores, dicha línea debe ser actualizada en la memoria principal, el estado de la caché que ha sido escrita pasará a E y el resto de copias en otras cachés a modo (I – Inválido).



Protocolos de coherencia de Caché

- Si estando marcada una línea con I es solicitada por otra caché (Ra) o leida por alguno de los procesadores (L), se mantendrá su estado I, no produciéndose ni la lectura ni el éxito del rastreo.
- Si estando marcada una línea con I se escribe desde uno de los procesadores, dicha línea debe ser actualizada en la memoria principal, el estado de la caché que ha sido escrita pasará a E y el resto de copias en otras cachés a modo (I – Inválido).



Protocolos de coherencia de Caché

- **VENTAJAS DEL PROTOCOLO MESI**
  - Es un protocolo sencillo y robusto
- **DESVENTAJA DEL PROTOCOLO MESI**
  - La escritura de una línea en una cualquiera de las caches supone su anulación en el resto.

Protocolos de coherencia de Caché

- **Protocolo FUTUREBUS + :**El protocolo FUTUREBUS + es un protocolo estándar del bus IEEE que toma la ventaja del uso de un bus síncrono basado en el protocolo MESI pero implementando post-escritura.
- Cuando un procesador intenta acceder a datos contenidos en una caché pero el bus está ocupado, es capaz de detectar si la línea buscada pasa por el bus y en ese caso captarla, ahorrando así el tiempo de tener que ocupar el bus con su petición.
- Este protocolo usa los cuatro estados MESI.
- **OTROS PROTOCOLOS :**
  - PROTOCOLO MOESI
  - PROTOCOLO N+1
  - …