# LA MEMORIA CACHÉ 5

## 5.1- NECESIDAD DE LA CACHÉ

La arquitectura de los últimos microprocesadores está orientada a mejorar el rendimiento, de manera que se ejecuten más instrucciones por unidad de tiempo.

Para llevar a cabo este objetivo los nuevos microprocesadores se apoyan en tres recursos fundamentales:

- Arquitectura superescalar, característica de la arquitectura RISC. Paralelismo explícito. Se compone de instrucciones sencillas.
- Supersegmentación, es decir, segmentación con elevado número de etapas. Esta técnica precisa de técnicas sofisticadas para eliminar riesgos, especialmente en las instrucciones que contengan saltos condicionales.
- Potenciamiento de la memoria caché, para aumentar la velocidad de la memoria.

Un procesador segmentado ejecuta cada instrucción en cinco etapas que son:

1. Búsqueda de la instrucción. (*Fetch*): se accede a la memoria para buscar la instrucción.
2. Decodificación: es trabajo de la CPU y su ciclo es de 10 ns cuando trabaja a 100 MHz.
3. Búsqueda de los operandos: se accede.
4. Ejecución de la instrucción: realizada por la CPU.
5. Escritura del resultado: se accede de nuevo a la memoria .

De estas cinco etapas hay tres que consisten en un acceso a la memoria principal (DRAM), donde están las instrucciones y los datos (búsqueda de la instrucción, búsqueda de los operandos, escritura del resultado) y las otras dos son propias del procesador. Las dos etapas que afectan al procesador se realizan en un ciclo cada una. Las otras tres etapas ocupan un tiempo equivalente al de acceso a la memoria principal.

Si suponemos que la memoria principal DRAM trabaja con un tiempo de acceso de 50 ns, los tres accesos a la memoria principal suman un total de 150 ns . Si además el microprocesador trabaja a 1Ghz, su periodo es de 1 ns, por lo que el resultado anterior representa un desequilibrio notable entre etapas. Este hecho es la causa fundamental de un mal rendimiento.

Hay una laguna entre la tecnología de construcción de procesadores y la de la memoria.

Para que el rendimiento sea óptimo, el objetivo es que todas las etapas duren lo mismo. Debemos disponer de una memoria más rápida. Las cachés son ultrarápidas pero de poca capacidad y muy caras, por lo que no se pueden sustituir las DRAMs por cachés. Por lo que debemos emplear la jerarquía de memoria, la cuál consiste en interponer entre la CPU y la memoria DRAM una memoria ultrarápida (caché).

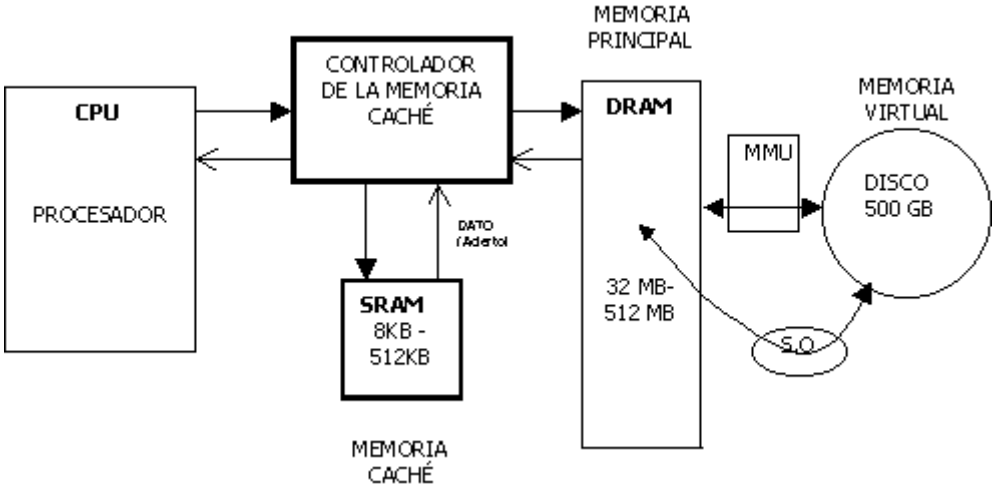


Figura 5.1. Jerarquía de memorias

La memoria caché es una SRAM (RAM estática) que tiene un tamaño comprendido entre 8 KB y 512 KB mientras que la Memoria Principal puede alcanzar cientos de MB.

Como se estudió en el tema de la Memoria Virtual, cuando la CPU solicita una información no presente en la Memoria Principal, la MMU trae el bloque necesario de la Memoria Virtual. Este mismo procedimiento también es empleado para el comportamiento entre la Memoria Principal y la memoria Caché.

La CPU se relaciona con la Memoria Caché y si ésta contiene lo solicitado se tardan unos pocos ns en el acceso. Si hay fallo se debe acceder a la Memoria Principal.

El movimiento de datos se genera de forma que produciéndose una ausencia, la caché recibe de la Memoria Principal el dato pedido y otros contiguos, que previsiblemente va a pedir la CPU. Esto es predecible dependiendo de la programación, ya que puede estar construida mediante las reglas clásicas de la contigüidad espacial y temporal. Es fácil prever la siguiente información que solicitará la CPU debido a la localidad espacial, la CPU tiende a requerir datos que estén en posiciones cercanas físicamente.

Si la caché tiene el dato, solo penaliza el tiempo de acceso a la misma, pero cuando la caché no dispone del dato solicitado, el tiempo empleado es el de acceso a la Memoria Caché más el de acceso a la Memoria Principal.

Así por ejemplo, si empleamos una memoria caché con un tiempo de acceso *tc* , con una tasa de acierto del 90% y una memoria principal con un tiempo de acceso *tm*, la formula que refleja el tiempo medio de acceso al sistema de memoria es:

$$\overline{t} = 0.9 \times tc + 0.1 \times (tc + tm)$$

Si suponemos que el tiempo de acceso a la memoria caché es de 5ns y el tiempo de acceso a la Memoria Principal es de 5 y 50 ns respectivamente, el resultado es:

$$\overline{t} = 0.9 \times 5ns + 0.1 \times (5ns + 50ns) = 10\ ns$$

Un sistema con estas características tiene como tiempo de acceso medio 10 ns , por lo que se ve reflejada la importancia del algoritmo de intercambio de información, es decir, la capacidad que tiene el sistema de predecir las siguientes peticiones de información.

Hay dos factores destacables en la memoria que proporciona la caché:

1. **Factor de velocidad:** Es la relación entre el tiempo de acceso a la Memoria Principal y el tiempo de acceso a la Memoria Caché:

$$\gamma = \frac{tp}{tc}$$

2. **Factor de eficacia:** La eficacia depende en gran medida, del programa que se esté ejecutando, es decir, de cómo está escrito y estructurado el software:

$$\text{Eficacia} = \frac{tc}{t}$$

## 5.2- PRINCIPIO DE FUNCIONAMIENTO DE LA CACHÉ

Una caché está estructurada en tres bloques importantes:

- **BLOQUE DE ETIQUETAS: RAM-CAM:** Es una memoria de acceso por contenido. No se accede por dirección de memoria, sino que, se compara el valor o dato con los que hay dentro de la memoria y así sabemos si se encuentra o no contenido en ella.
- **BLOQUE DE DATOS ASOCIADOS: SRAM:** Es un conjunto de datos de forma que a cada dato le corresponde una etiqueta. Por ejemplo: dato 0 → etiqueta 0.
  Si hay acierto, la etiqueta que coincide con el dato introducido, devuelve el dato asociado a la misma.
- **LÓGICA DE CONTROL:** Comparadores de "n" bits, tantos como tenga la etiqueta.

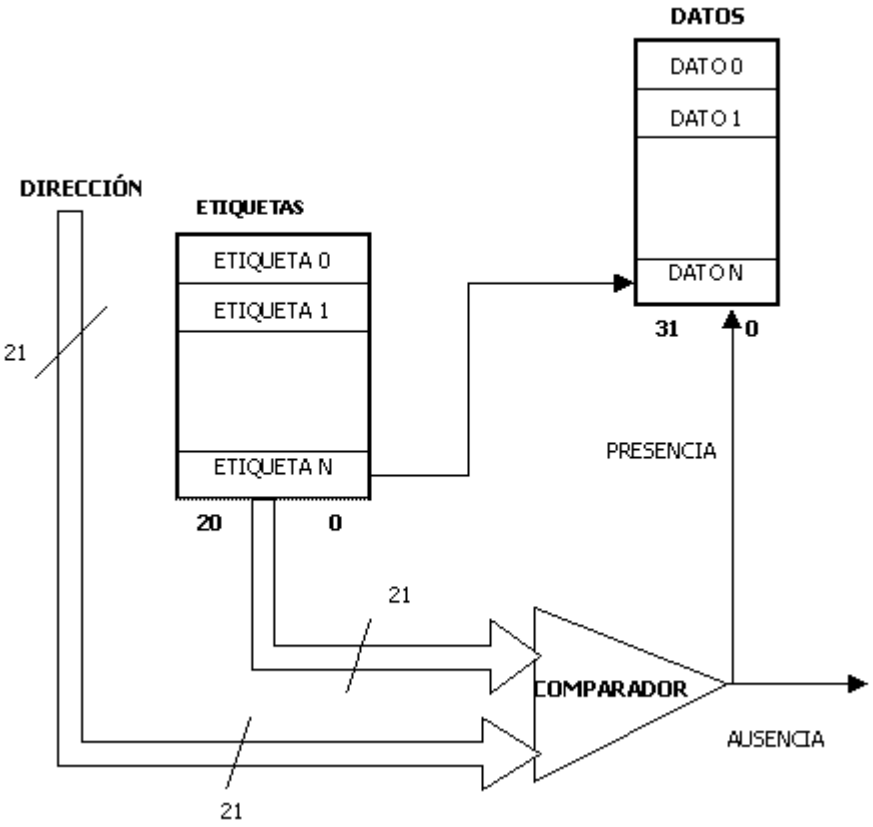


Figura 5.2. Principio de funcionamiento de la caché

El bus de direcciones va a la caché, que tiene dos partes: la de la etiquetas y la de datos. Hay N posiciones de etiquetas y cada etiqueta tiene 21 bits. Los 21 bits de la etiqueta se corresponden con los 21 bits de más peso de la dirección, por eso, el comparador coge las N etiquetas y las compara con los 21 bits de más peso de la dirección. Si alguna coincide, el comparador devuelve un uno, lo que implica presencia. En cambio, si no coincide ninguna, el comparador devuelve un cero lo que implica ausencia , lo que significa que esa dirección no está contenida en la caché.

Aunque hay múltiples organizaciones en la caché, desarrolladas más adelante, concretamente en el ejemplo de la figura 5.2 hay un dato por etiqueta, por lo tanto N datos, existiendo una correspondencia etiqueta-dato. Como por ejemplo: ETIQUETA1 →DATO1.

Si el comparador devuelve un uno (presencia), el dato correspondiente a la etiqueta es transferido a la CPU reflejando los bits restantes, la posición en la que se encuentra el dato. Este sistema es más rápido que el de localizar una dirección en memoria principal.

Si el comparador devuelve un cero (ausencia) se accede a la memoria principal.

## 5.3- TIPOS DE CONEXIONADO DE LAS MEMORIAS CACHÉ

La Memoria Caché se puede conectar a la CPU en serie o en paralelo.

### 5.3.1 – Conexión en serie

La CPU sólo se conecta con la caché por lo que todas las peticiones que hace la CPU al bus del sistema son a través de la memoria caché.

Por lo tanto todo lo que necesita la CPU del sistema se lo proporciona la memoria caché y cómo es de tamaño y tiempo de acceso reducido cada vez que el dato está almacenado en la caché, el tiempo de acceso es muy reducido y evita manejar el bus del sistema.

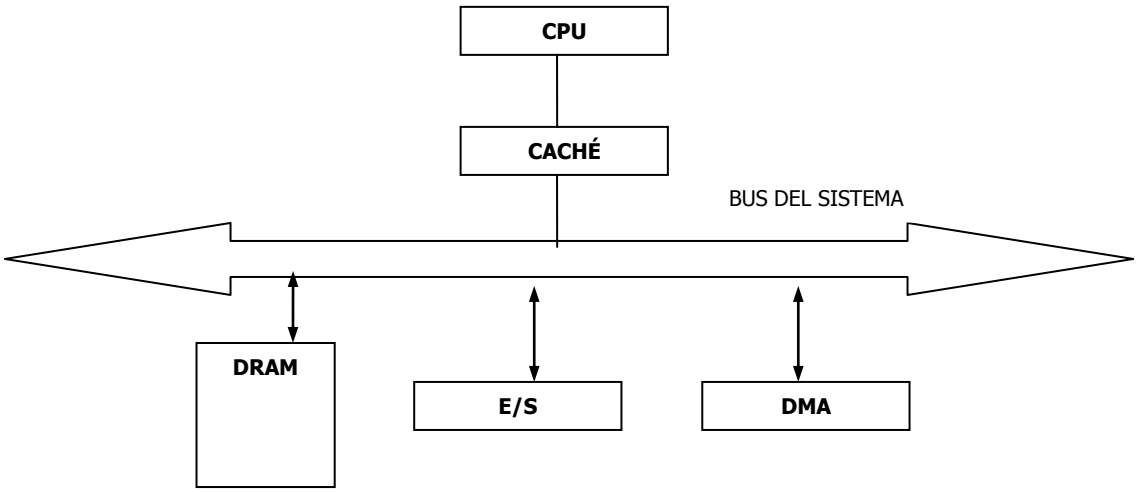


Figura 5.3. Conexión en serie

Si la memoria caché contiene el dato solicitado por la CPU, la parte superior de la figura funcionará de manera independiente y en pocos nanosegundos proporcionará la información, liberando de la búsqueda a la parte inferior de la figura.

De esta forma si el bus del sistema está desocupado, simultáneamente que la CPU ejecuta instrucciones, los módulos de entrada/salida pueden estar trabajando con la memoria principal. Se permite el paralelismo.

El inconveniente de la conexión en serie es la penalización de tiempo ya que cuando la información que necesita la CPU no está en la caché tiene que trasladar la petición al bus del sistema para poder acceder a la memoria principal.

Otro inconveniente es que la caché es de uso obligatorio ; no se puede desconectar la caché y conectar la CPU al sistema.

Cuando el bus del sistema queda libre, puede ser utilizado por todos los elementos que dependan de él.

## 5.3.2 – Conexión en paralelo

En la conexión en paralelo, todo depende del bus del sistema:

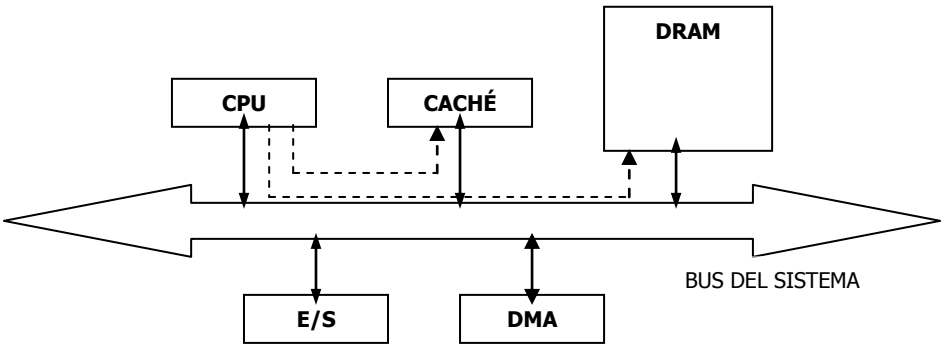


Figura 5.4. Conexión en paralelo

Cada vez que la CPU realiza una petición, la envía simultáneamente a la caché y a la Memoria Principal . Si se encuentra la información contenida en la caché es entregada al bus en pocos ns avisando a la memoria principal para que no continúe con la búsqueda de la información. De esta forma se parará el ciclo. Si la memoria caché no tiene el dato pedido, la memoria principal sigue trabajando, con lo cual no hay penalización de tiempo, lo que es una gran ventaja.

Este sistema se utiliza mucho porque cuenta con otra ventaja, y es que la caché es opcional, es decir, se puede añadir o eliminar sin alterar el funcionamiento del sistema.

El inconveniente es que la CPU realiza todas las peticiones por el bus del sistema, quedando sobrecargado por el continuo flujo de información, lo cual deja muy poco espacio libre para E/S y DMA, ya que el bus del sistema se libera muy pocas veces.

## 5.4 ARQUITECTURA DEL SUBSISTEMA DE MEMORIA CACHÉ

Las características fundamentales que determinan un subsistema caché son las siguientes:

- Tamaño de la caché.
- Organización.
- Estructura física de una caché.
- Actualización de la caché.
- Actualización de la memoria principal.

## 5.4.1 – El tamaño de la caché

El tamaño de la caché suele oscilar entre 8KB y 512KB. En muchas ocasiones, no se consigue aumentar el rendimiento mediante cachés de mayor capacidad. Lo que influye son los algoritmos de transferencia de la memoria caché.

Aumentos importantes del tamaño de la caché suponen variaciones pequeñas en el porcentaje de acierto, porque el porcentaje de aciertos se basa en el algoritmo. Según los algoritmos, la mejor capacidad de la caché está entre 32K y256K.

El precio sube de forma exponencial: aumentando el tamaño x8, la tasa de aciertos sólo aumenta un 4%. Se reflejá en la siguiente gráfica.

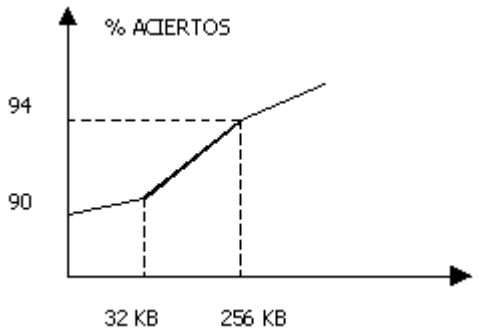


Figura 5.5.Gráfica número de aciertos respecto tamaño de la caché

## 5.4.2 – Tipos de organización

Existen tres tipos de organización de las memorias caché:

1. **Totalmente asociativa:** Se caracteriza porque cualquier posición de la memoria principal se puede ubicar en cualquier posición de la memoria caché.

   Se usan comparadores de 32 bits, muy caros, con buenos algoritmos.

   Todas las direcciones de la memoria principal que quepan se pueden guardar sin ningún orden preestablecido, no hay normas, hay una flexibilidad total.
   Esta flexibilidad es un problema grave, puesto que la caché necesita los 32 bits de la dirección para compartirla con la etiqueta.

   Cada partición de la caché se puede ubicar en cualquier parte de la memoria principal. Como hay flexibilidad total, la etiqueta ha de contener TODOS los bits de la dirección de la memoria principal a los que corresponden los datos.

Luego el inconveniente de este tipo de organización de la memoria caché es que tiene que tener apuntadas todas las direcciones en la zona de etiquetas y por lo tanto éstas serán muy largas Consecuentemente el comparador va a ser muy lento y caro ya que va a necesitar muchos bits.

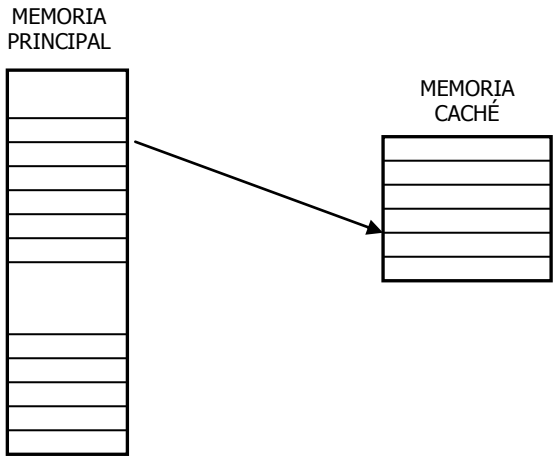


Figura 5.7. Organización de memoria caché asociativa

2. **Asociativa de una vía:** Suponemos que la memoria principal tiene 256 KB y la caché 256 Bytes. La memoria principal se divide en bloques de 256 bytes. Habrá 1K bloques, es decir, 1024 bloques.

   Cada posición de 1 bloque de la memoria principal sólo puede ir a la misma posición de la caché.

   Solo es necesario precisar cuál es el bloque, porque sabiendo el bloque ya se conoce la posición de la caché debido a que todas las direcciones de un bloque están en la misma posición que en la caché.

   La ventaja es que para conexionar la memoria principal necesitamos que el bus de direcciones tenga 18 líneass. La etiqueta, por tanto, sólo necesita 10 bits: $2^{10}$ = 1K es decir, lo que ocupa un bloque. Sólo necesito los bits necesarios para definir el bloque y me ahorro así los 8 bits.

   Como sólo cabe una dirección de cada bloque en una posición , tendremos que machacar la dirección anterior, cada vez que queramos modificarla. La caché estará sufriendo continuamente modificaciones.

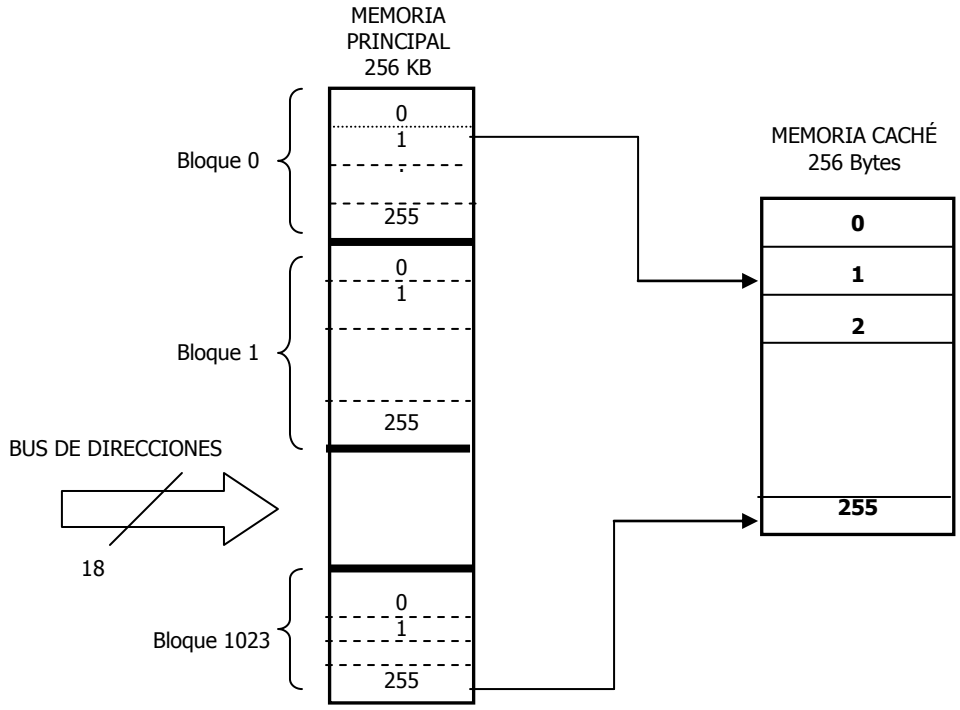


Figura 5.8. Organización asociativa de una sóla vía de la memoria caché

3. **Asociativa de "n" vías:** Su funcionamiento es similar al de una vía pero la caché se va a descomponer en varias vías, no en una sola

La memoria principal se divide en fragmentos iguales a cada uno le corresponde una de las vías funcionando de la misma manera que en el caso anterior, incorporando la ventaja de que no es necesario machacar las posiciones de memoria inmediatamente que surja una modificación .

Con la caché asociativa de N vías, se ahorra gran cantidad de bits debido a que hay correspondencia entre las posiciones de las páginas de la memoria principal y las posiciones de la caché.

Esta organización de la memoria ofrece un mayor rendimiento pero las vías son de menor tamaño. Dependiendo de la aplicación será mejor utilizar la memoria asociativa de 1 vía o la de n vías.

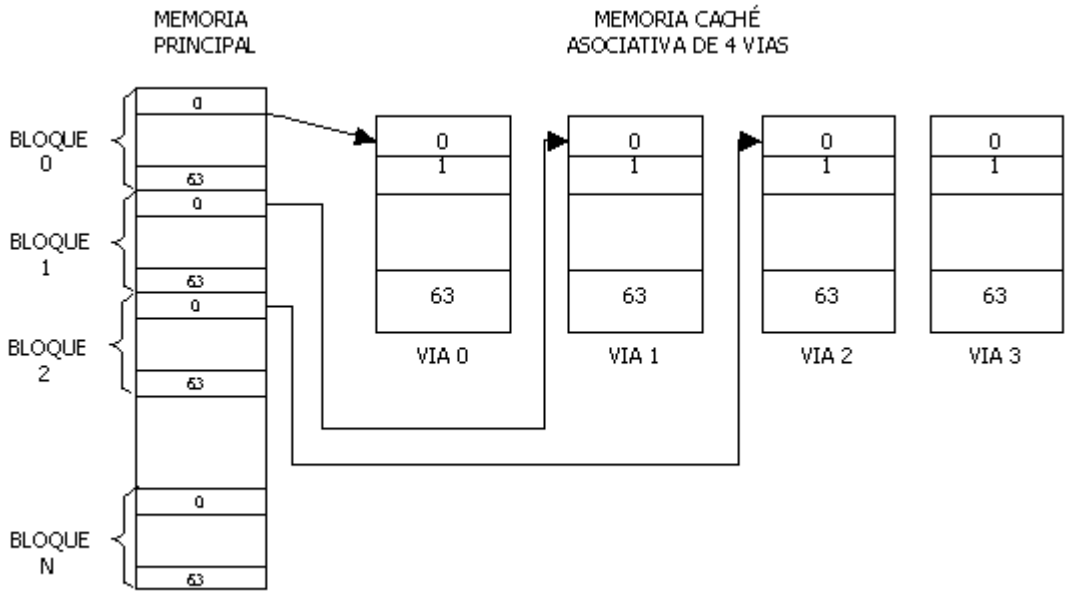


Figura 5.9. Organización asociativa de "N" vías de la memoria caché

## 5.4.3 – La estructura física de una caché

Las cachés son el elemento diferencial que tienen los nuevos procesadores pentium con respecto a los anteriores. Para referirnos a una caché real presentaremos como ejemplo la del procesador Pentium.

El Pentium tiene 2 memorias cachés independientes: una para datos y otra para instrucciones, permitiendo el paralelismo entre instrucciones y datos. Cada una de ellas es de 8 KB.

Son cachés asociativas de 2 vías. Las etiquetas contienen los 20 bits de más peso de la dirección además de dos bits de código: el WP protege contraescritura y V predice si es línea váilda o no.

Los datos asociados a la etiqueta tiene cada uno 32 bytes ( = una línea ), es decir, cada vez que la caché se actualiza se le introducen 32 bytes. Cada vía tiene 128 etiquetas. Cada vía guarda 128 x 32 bytes = 4 Kbytes de datos, luego la memoria la dividimos en 4KB.

La sustitución de direcciones se realiza mediante el algoritmo LRU que es aquel que selecciona la dirección que menos se ha utilizado.

Cuando dicha dirección se va a incorporar en una línea y estas están ocupadas, se consultan los dos bits del LRU, y la dirección menos utilizada será la que se machaque.

La estructura interna de la caché del Pentium es la siguiente:

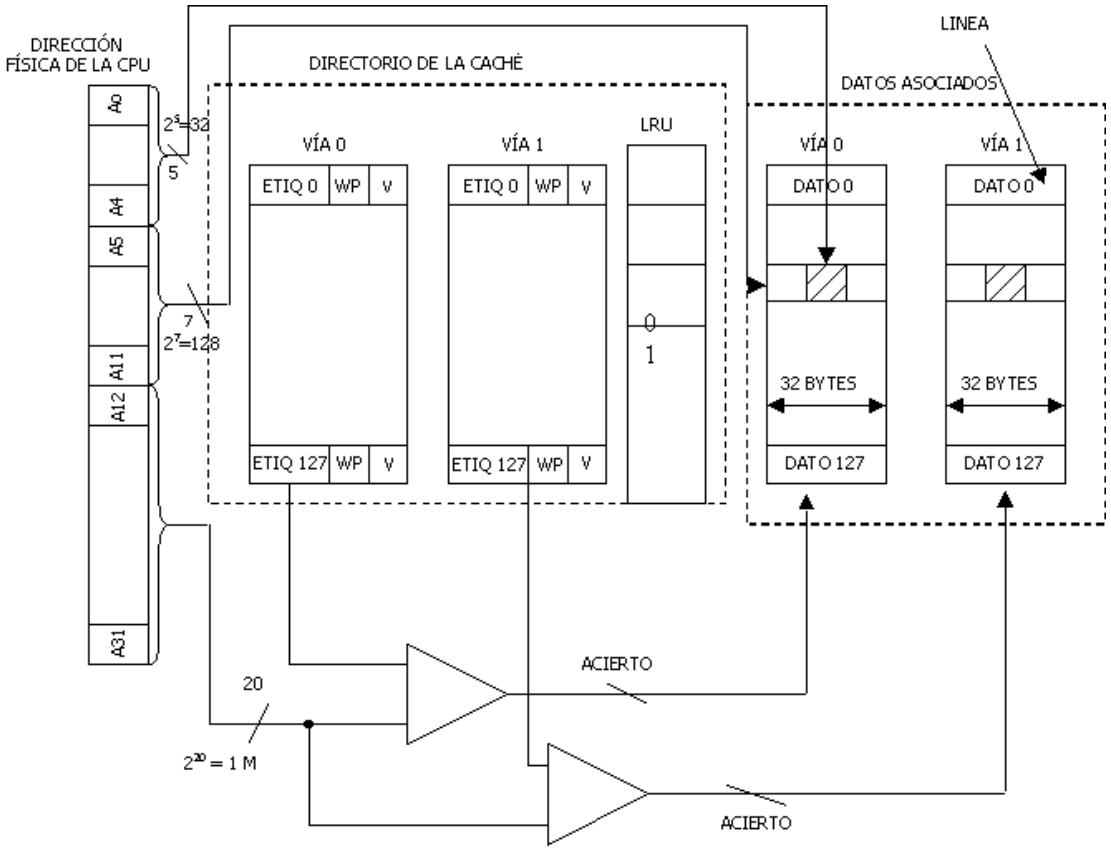


Figura 5.6 Estructura interna de la caché del Pentium

## 5.4.4 – Actualización de la memoria caché

En el pentium cuando se produce una ausencia en la caché se busca la información de una línea completa en la memoria principal y se carga en la caché, en una vía que esté libre.

Si fuera totalmente asociativa, cualquier línea se podría almacenar en cualquier posición de la caché. Si es asociativa de una vía sólo hay una posibilidad. Si es de varias vías, como el Pentium, hay varias posibilidades.

Si se necesita introducir una posición y hay alguna vía vacía, ésta se ocupa. Si hay una posición que las cuatro vías tengan ocupada, se aplica uno de estos dos algoritmos para extraer una de ellas y machacarla:

- **RANDOM:** Aleatoriamente se elige y se machaca una de las posiciones ocupadas de una de las cuatro vías. El inconveniente es que quizá se machaque la información que a continuación se necesite. Se dice que es el algoritmo de las cachés de bajo coste.
- **LRU:** Se elimina la posición de la vía que menos se haya empleado últimamente, ya que suponemos que es la que menos se va a seguir utilizando. Su funcionamiento se basa en dos bits que apuntan a la vía que menos se ha empleado. Hay dos alternativas para actualizarla:

1) EL DATO PEDIDO VA EN ÚLTIMO LUGAR: El dato último es el que se ha direccionado y se traen los 31 bytes que van delante. Tiene un inconveniente, y es que la CPU hasta que no se carga el último byte no puede trabajar, , ya que los 31 anteriores bytes no le sirven. Tiene que esperar a que termine de cargarse la línea para transferir la información a la CPU. Es un método simple y rápido.

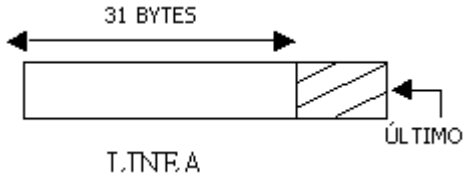


Figura 5.10. Dato pedido en el último lugar

2) EL DATO PEDIDO VA EN PRIMER LUGAR.: La caché, desde el primer dato que llega, ya tiene la información que ha solicitado la CPU. El problema es que es un algoritmo más complejo y por tanto, se necesita más circuitería interna, más transistores de silicio.

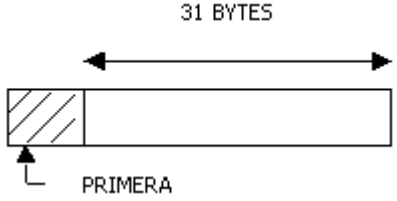


Figura 5.11. Dato pedido en primer lugar

Cuando la caché está en serie la CPU funciona de la siguiente forma:

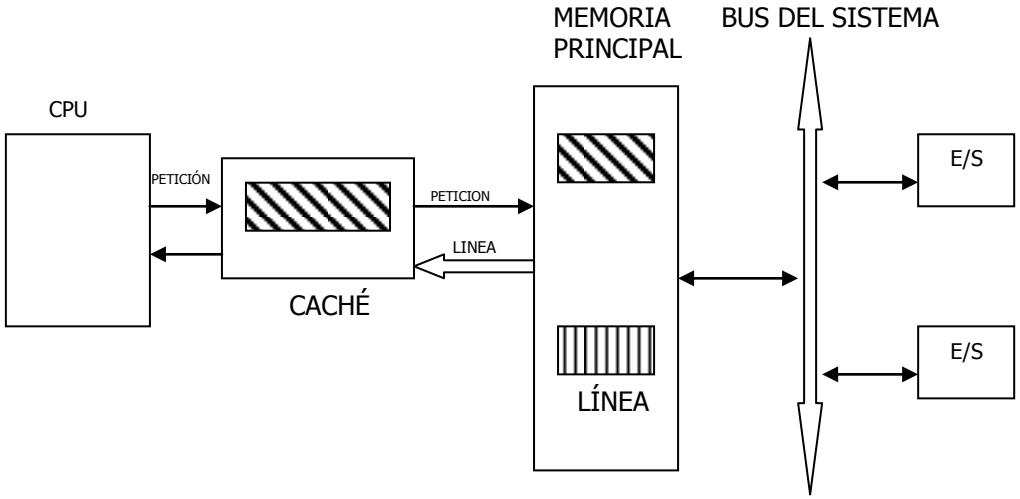


Figura 5.12. Gráfico de la actualización de la memoria principal

La CPU siempre se dirige a la caché. Si va a leer un dato a la caché no tiene ningún inconveniente, pero si la CPU necesita escribir en la caché y modifica una de sus posiciones, la CPU da por concluida su operación resultando que esa posición de la caché tiene una imagen en memoria principal que no tiene constancia de esta modificación, por lo que hay una discordancia. Esto puede dar lugar a errores graves. Por lo tanto se debe escribir lo que hay modificado en la caché en la memoria principal.

La memoria principal se actualiza mediante uno de estos tres métodos:

- **Actualización por escritura inmediata**: Cada vez que la CPU modifica la caché, ésta última manda una orden al bus del sistema y se transfiere la información a la CPU, consiguiendo que no haya errores de coherencia y actualizando así la memoria principal. Muchas veces la escritura de la CPU es repetitiva, ya que la escritura en la caché es continua, y bloquea el bus del sistema. Es anormalmente seguro, pero baja el rendimiento. De esta forma se evitan muchos errores.

- **Actualización por escritura diferida:** La caché dispone de registros intermedios donde cargas temporalmente las modificaciones que ha habido en la caché. Actualiza la memoria principal cuando el bus del sistema está libre, sólo hay que esperar a que el bus del sistema está inactivo. Puede existir falta de coherencia mientras se espera y los periféricos pueden leer datos erróneos de la memoria principal. Es más rápido, pero hay un pequeño riesgo de fallo.

- **Actualización por escritura obligada:** La actualización de memoria principal se produce cuando no queda otro remedio, por lo que no hay nunca fallo.

Hay que actualizar la memoria obligatoriamente cuando:

1. Se accede a una posición de la memoria principal modificada en la caché por la CPU. Antes de dar paso a esa lectura, hay que escribir el dato modificado.
2. Cuando hay que eliminar una línea en la caché porque está llena, en la cual hay un dato modificado. Antes de borrarlo hay que enviar dicho dato a la memoria principal.

## 5.5 PROTOCOLO MESI

En los sistemas multiprocesador puede haber varias cachés, por tanto, puede suceder que una misma posición de la memoria principal la están empleando dos CPU´s y como consecuencia, permanecer en las dos cachés. Cuando se plantea esta situación, surge la necesidad de asegurar que cualquier acceso a la memoria lea el dato más actualizado. Para evitar esta inconsistencia de la caché Intel desarrolló el protocolo MESI ( modified exclusive shared invalid ).

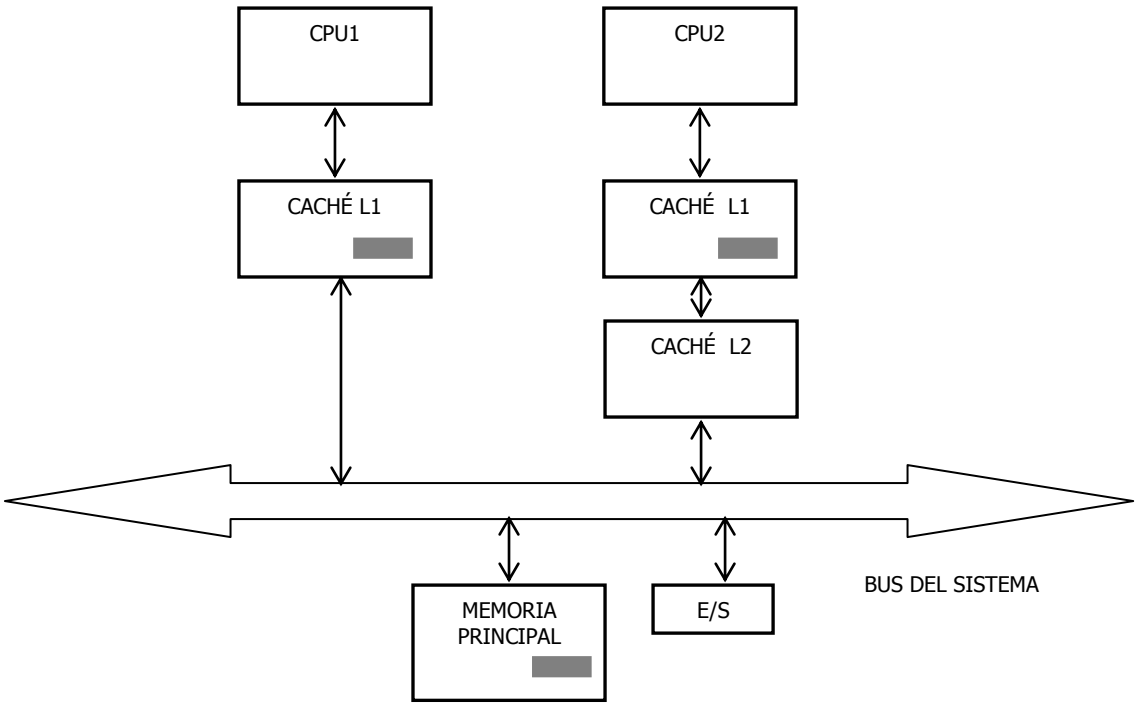


Figura 5.13. Multiprocesador con más de una caché

El protocolo MESI asigna cuatro estados diferentes a cada línea (gestionados por los bits MESI), que definen si una línea es válida (es decir, si existe presencia o no). Si está disponible para otras cachés, o ha sido modificada. Estos estados pueden ser modificados bien por el propio procesador, o bien por unidades lógicas externas tales como otros procesadores o el controlador caché L2.

Los posibles estados de la línea son:

- **M:Modificado:** La línea que lleva dicha M está modificada por 1 escritura del procesador y a la espera de actualizar la memoria principal si es preciso.
- **E:Exclusiva:** La línea sólo la tiene una caché, y está sin modificar. La memoria principal tiene una copia .
- **S:Simultáneo:** Esta línea está repetida en otras cachés. Si se escribe en una de ellas, las demás se invalidan automáticamente.
- **I:Inválido:** Esta línea está invalidada. La lectura de esta posición por parte del procesador genera una ausencia y por tanto un llenado con los datos que provienen de la memoria principal. Si el procesador escribe en esta posición, se procede a actualizar la memoria principal de inmediato.

## 5.6 NIVELES DE JERARQUÍA EN LA CACHÉ

Una de las técnicas de aumentar el rendimiento, es disminuir el tiempo de acceso a las cachés y otra aumentar la tasa de aciertos.

Si queremos mejorar el rendimiento disminuyendo el tiempo de acceso a las cachés nos resultará complicado ya que el tiempo de acceso a las cachés es una característica tecnológica.

Sin embargo, podremos aumentar dicho rendimiento aumentando la tasa de aciertos.

Para realizar dicho aumento de aciertos debemos:

- **Mejorar los algoritmos de carga en la caché:** Si ese algoritmo es inteligente y se adapta al programa tendrá que tener en cuenta la vecindad temporal y la espacial.
- **Aumentar el tamaño de la caché:** El tiempo de acceso aumenta si aumenta el tamaño, así que tendrá que haber un equilibrio. Como la tecnología no se puede forzar, utilizaremos la jerarquía de memoria, utilizando cachés de primer, segundo nivel, etc...

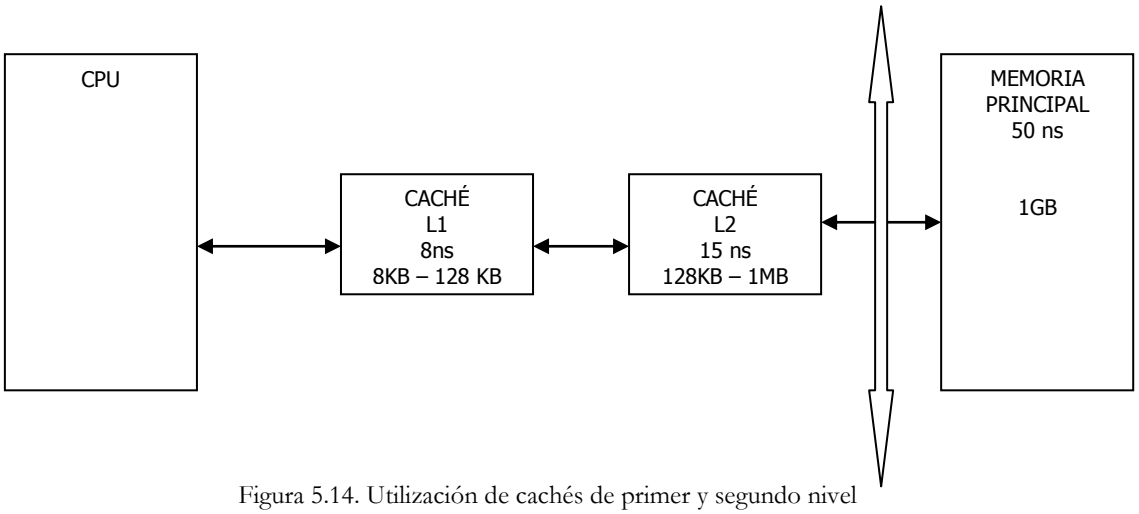


Figura 5.14. Utilización de cachés de primer y segundo nivel

La caché L1 es pequeña y rápida y los algoritmos que se cargan son los algoritmos clásicos (vecindad temporal y espacial). Si la caché L1 no tiene el dato, se transfiere la petición a L2. La caché L2 tendrá todo lo que tiene la caché L1 y algo más. Por lo tanto la caché L2 tiene mayor capacidad y es algo más lenta.

Los algoritmos que cargan a L2 son especiales. Intentan ajustarse al programa.

En la actualidad, en el ITANIUM, la jerarquía de niveles ya llega a tres niveles para lo cual destinan 300 millones de transistores para los niveles de caché y 25 millones de transistores para la CPU.

## 5.6.1 – Conexionado de cachés de varios niveles:

Hay 2 tipos de conexionado: conexionado en paralelo y conexionado en serie:

- **Conexionado en paralelo:**

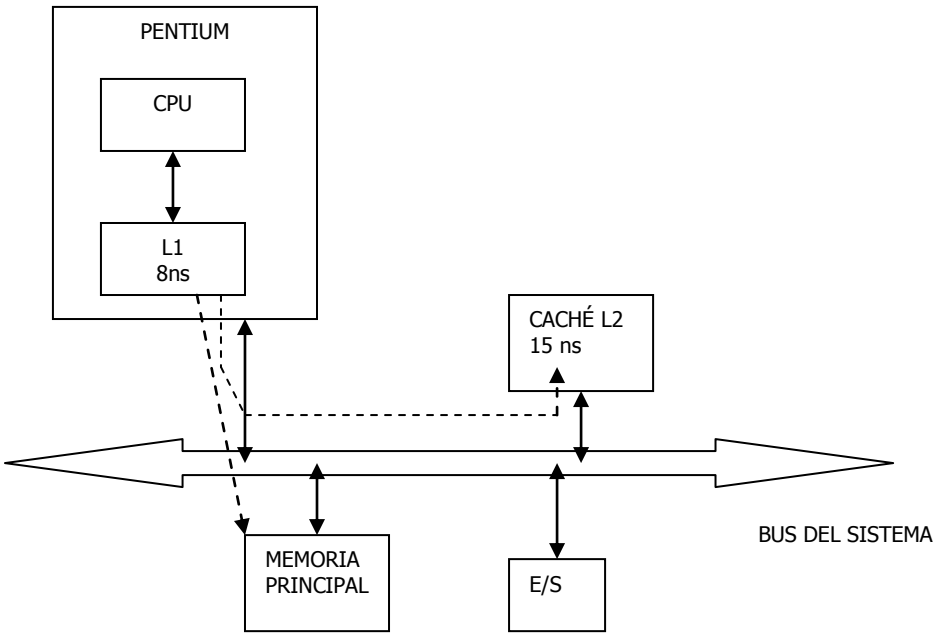


Figura 5.15. Conexionado paralelo

Si la caché L1 da fallo, no importa porque a través del bus del sistema se envía la petición a la caché L2 y a la memoria principal. Si la caché L2 no la tiene en su memoria, no habrá penalización de tiempo puesto que la memoria principal y la caché L2 reciben al mismo tiempo la petición. La caché L2 es optativa, y podremos añadirla o eliminarla cuando queramos sin dañar ni alterar el conexionado.

El bus del sistema recibe todos los fallos de la caché L1 y por lo tanto siempre se está empleando, lo cual es un inconveniente a tener en cuenta.

- **Conexionado en serie:**

Desventajas del conexionado en serie:

1. Hay penalización de tiempo: Si la caché L2 da fallo transcurrirá tiempo hasta que la caché L1 reciba la petición.
2. La caché L2 es obligatoria porque la caché L1 no se puede conectar directamente al bus del sistema.

Ventajas del conexionado en serie:

1. El tráfico de peticiones a la memoria principal disminuye considerablemente y por lo tanto el bus del sistema está desocupado la mayor parte del tiempo y puede hacer frente a los sistemas adheridos.

El conexionado en serie es el siguiente:

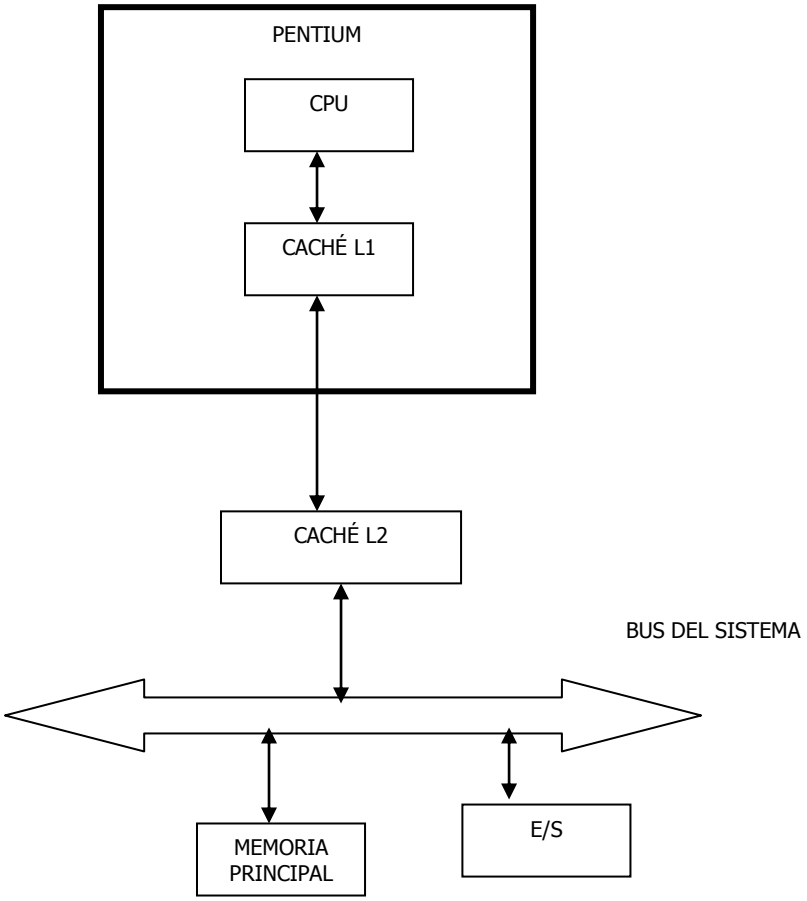


Figura 5.16. Conexionado serie