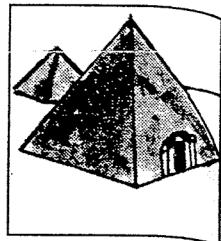


## 1.12 MEMORIA CACHE Y JERARQUÍA DE MEMORIAS EN UN COMPUTADOR



### *¿Qué es una memoria caché ?*

Este tipo de memorias, junto con el pipeline y el advenimiento de las arquitecturas RISC constituyen una de las mejoras sucesivas al modelo de Von Neumann (sección 1.14).

El bajo costo y crecimiento vertiginoso en la velocidad de los procesadores en relación con los Mhz de sus pulsos reloj, no ha sido acompañado históricamente en igual medida por las memorias a las que ellos acceden. Así en el periodo en que los Mhz de las UCP pasaron de 10 a 50, las DRAMS sólo mejoraron de 100 a 70 nseg. Se necesita que las memorias a las que las UCP acceden sean baratas, de gran capacidad de almacenamiento, y que puedan suministrar instrucciones y datos tan pronto como las UCP pueda procesarlos (en el tiempo de un ciclo reloj de la UCP). También deben permitir gran "ancho de banda" (bytes transmitidos por segundo).

Las memorias "caché" ubicadas entre la UCP y la memoria principal (fig. 1.77.a), de pequeña capacidad en relación con ésta, pero varias veces más rápidas, son una solución de compromiso entre costo y velocidad. Contienen las instrucciones y datos de memoria a los que la UCP accedió últimamente, siendo también a los que más probablemente accederá próximamente, como se discute a continuación. "Caché" significa oculta, escondida, en el sentido que la UCP envía una dirección por el bus de direcciones *desconociendo la existencia del caché*, que es un hardware también oculto al programador.

En general en los distintos niveles de los procesos de datos se trata que el acceso a la información requerida sea rápido y económico. Para tal fin en los procesos de datos se tiene en cuenta el *principio de localidad o proximidad*, el cual estipula que si ya se consultó información, es muy probable:

1. que la misma *pronto* sea consultada *otra vez (localidad o proximidad temporal)*.
2. que *pronto* se consulte información cercana, vecina a ella (*localidad o proximidad espacial*).

Por ejemplo, la apertura de archivos de un computador ha sido concebida de acuerdo con este principio. Así, cuando en el Word abrimos un archivo, y luego lo cerramos, si enseguida o en otra oportunidad queremos abrir un archivo, el nombre del último archivo cerrado será el primero en aparecer en una lista ordenada de los más recientes archivos consultados. De este modo se estima que es muy factible que volvamos a consultarlos (*proximidad temporal*). Por otro lado, si abrimos un archivo de una carpeta, y luego pedimos abrir otro, el Word automáticamente mostrará dicha carpeta (*proximidad espacial*), pues en la mayoría de los casos el archivo buscado es uno de la carpeta en uso. (*proximidad espacial*). En ambos casos se evita que el usuario abra carpetas innecesariamente, pudiendo así acceder más rápidamente a la información que necesita.

Vale decir, que del conjunto de todos los archivos almacenados en el disco rígido *no se accede a todos ellos con igual probabilidad*, sino que generalmente se consulta un subconjunto que pertenecen a una misma carpeta o a unas pocas en relación con todo el disco; y además se accede a ellos repetidamente. También en el caso de la memoria principal, en cualquier lapso breve de tiempo los accesos sucesivos a ella ocurren en un espacio limitado de direcciones consecutivas (*proximidad espacial*), y es corriente acceder repetidamente a esas direcciones (*proximidad temporal*).

Los programas constan de secuencias de instrucciones que están en direcciones consecutivas de memoria, siendo frecuente que secuencias se vuelvan a ejecutar n veces, dando lugar a los "ciclos". Lo primero implica que es muy probable que las direcciones de dos instrucciones que se ejecutan una tras otra sean muy cercanas.

A su vez, los datos que estas instrucciones procesan, si constituyen elementos ordenados de un vector o de una matriz, o los cercanos a la cima de una pila, se encuentran vecinos en una zona acotada de memoria, y los programas ordenan operar una y otra vez sobre estos datos.

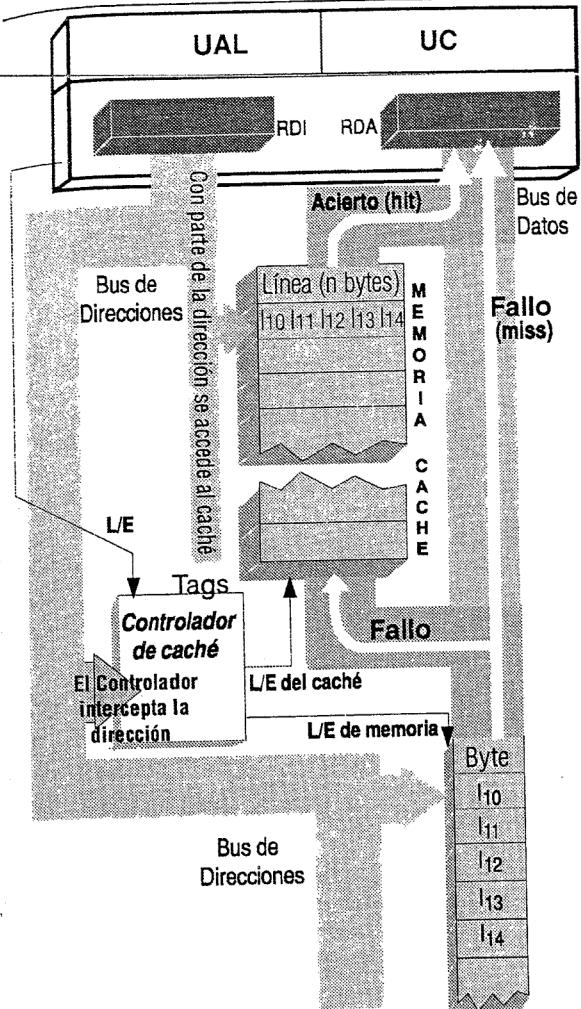


Figura 1.77.a

## MEMORIA PRINCIPAL

Por lo tanto, la ejecución de un programa se va realizando en secuencias de instrucciones relativamente cortas, que mayormente se ejecutan muchas veces, siendo que también los datos que cada secuencia procesa están en direcciones próximas, los cuales asimismo en general son accedidos repetidamente. En este caso también resulta para un conjunto de instrucciones y datos a procesar almacenados en memoria, que sus elementos no son accedidos con igual probabilidad; sino que durante cada lapso del proceso de datos que realiza la UCP sólo se accede con mayor probabilidad a dos subconjuntos: uno de instrucciones y otro de datos (que ellas operan), localizados en dos zonas reducidas de memoria principal (cuyos espacios de direcciones varían a medida que se ejecuta el programa). En especial se accede a bloques de datos de gran longitud en el procesamiento de textos, planillas y multimedia.

Los contenidos de estas dos zonas limitadas de memoria DRAM que almacenan instrucciones y datos que la UCP necesita acceder *por estar procesándolos o para procesarlos próximamente*, se copian en una memoria SRAM pequeña (como ser 512 KB ó 1 MB frente a 128 MB de la DRAM) pero de acceso mucho más rápido para la UCP que la DRAM: el "cache" o "antememoria". De este modo la UCP leerá del caché instrucciones y datos más rápidamente que de memoria, sin necesidad de acceder a ella.

Desde otra perspectiva, un caché permite *simular* una memoria principal DRAM de gran capacidad, pero con un tiempo de acceso semejante al de la SRAM de la caché. Este tiempo no debe superar el lapso T que media entre un pulso reloj y el siguiente, a fin de que la UCP no se quede "esperando" inactiva más de un pulso reloj. Cada pulso extra

de espera ("wait state") en cada lectura o escritura, implica una pérdida de performance de la UCP. Esto se manifiesta en un mayor tiempo en la ejecución de los programas.

Así, mientras los procesadores como el 80286 no superaban los 10 Mhz, o sea periodo  $T = 100 \text{ nseg}$ , ([sección 1.7](#)), dado que el tiempo de acceso de una DRAM entonces era de unos 90 nseg, en general no superaba dicho tiempo T. Pero ya en el 80386 de 33 Mhz, o sea  $T = 30 \text{ nseg}$ , el acceso a la DRAM apenas bajó a 70 nseg, superando el tiempo T (sin contar el tiempo que suma el controlador de memoria), por lo que se usó un caché externo (Level 2 = L2) para minimizar los "wait states". Con el 80486 rondando los 100 Mhz ( $T = 10 \text{ nseg}$ ) el acceso a un caché L2 superaba dicho T, debiéndose incorporar un caché dentro del 486 (Level 1 = L1), usando también un L2 para acceder más rápido a éste en lugar de la memoria, si la información a acceder no estaba en L1. Estos dos niveles de caché se usan hoy día para un mejor rendimiento, pudiendo existir más niveles.

Un caché guarda en las celdas que constituyen sus "líneas" un subconjunto de bloques de la memoria. Cada línea almacena un bloque de bytes consecutivos de la memoria (por ejemplo 16 ó 32). El caché requiere un subsistema circuital complejo denominado "*controlador de cache*" que cumple varias funciones, siendo las más importantes:

- Intercepta la dirección que envía la UCP por el bus de direcciones a fin de determinar si el contenido correspondiente a ella está (acierto = "hit") o no (fallo = "miss"). En el primer caso permitirá que la UCP acceda al caché, y en el segundo a la memoria con la consiguiente pérdida de tiempo. Para tal determinación cuenta con una rápida memoria auxiliar: la "tag memory".
- En los cachés con correspondencia asociativa, decide cuál bloque será reemplazado por otro proveniente de memoria cuando ocurre un fallo.
- Implementa la forma en que los resultados obtenidos por la UCP se guardarán en memoria.

*¿Cómo es la estructura de un caché de correspondencia directa?*

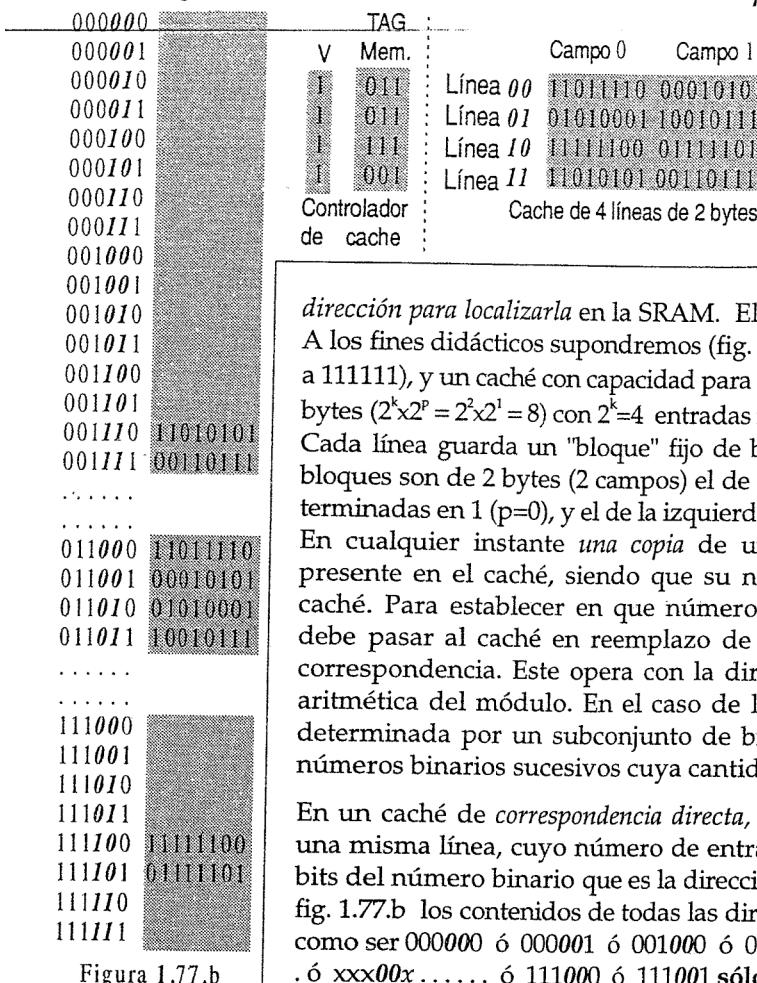


Figura 1.77,b

Mientras que un chip de memoria est<sup>o</sup> organizado como una sucesión de celdas independientes que pueden guardar un byte identificable cada una por su dirección (fig. 1.77.a), un caché se organiza en un conjunto de líneas que típicamente guardan un bloque de 16 ó 32 bytes cada una. Cada línea se identifica y localiza en el caché por su número de "entrada" o "índice", que viene a ser como su

dirección para localizarla en la SRAM. El número de líneas es una potencia de dos. A los fines didácticos supondremos (fig. 1.77.b) una memoria de 2<sup>6</sup> bytes (direcciones 000000 a 111111), y un caché con capacidad para 8 bytes estructurado en 2<sup>k=4</sup> líneas que guardan 2<sup>p=2</sup> bytes (2<sup>k</sup>×2<sup>p</sup> = 2<sup>2</sup>×2<sup>1</sup> = 8) con 2<sup>k=4</sup> entradas numeradas de 00, 01, 10, 11 (números de k=2 bits). Cada línea guarda un "bloque" fijo de bytes consecutivos de memoria. En este ejemplo los bloques son de 2 bytes (2 campos) el de la derecha de la línea para direcciones de memoria terminadas en 1 (p=0), y el de la izquierda para las terminadas en 0 (p=1). En cualquier instante una copia de un subconjunto de bloques de la memoria está presente en el caché, siendo que su número es mucho mayor que el de líneas de un caché. Para establecer en qué número de línea se asigna un bloque de memoria que debe pasar al caché en reemplazo de uno residente en él, se utiliza un algoritmo de correspondencia. Este opera con la dirección de memoria que produjo el fallo usando aritmética del módulo. En el caso de las direcciones se reduce a una correspondencia determinada por un subconjunto de bits de cada una de ellas, por ser las direcciones números binarios sucesivos cuya cantidad es una potencia de dos.

En un caché de correspondencia directa, cada bloque de memoria se adjudica siempre a una misma línea, cuyo número de entrada (dirección en el caché) es un subconjunto de bits del número binario que es la dirección de cualquiera de los bytes del bloque. Así, en la fig. 1.77.b los contenidos de todas las direcciones de memoria cuyos bits en "italica" son 00, como ser 000000 ó 000001 ó 001000 ó 001001 ó 010000 ó 010001 ó 011000 ó 011001 ó .. ó xxx00x ..... ó 111000 ó 111001 sólo se pueden guardar en la línea 00 del caché. Según terminen en 0 ó 1 irán a la izquierda o derecha, respectivamente.

Análogamente, los contenidos de direcciones con bits indicados 01 tales como 0000010 ó 000011 ó 001010 ó 001011 ó 010010 ó 010011 ó 011010 ó 011011 ó ... ó xxx01x ..... ó 111010 ó 111011 se guardarán siempre en la línea 01.

Direcciones del tipo xxx10x y xxx11x deben guardarse en las líneas 10 y 11, respectivamente.

En general, si un caché tiene 2<sup>k</sup> líneas (4 en este ejemplo) y cada una guarda 2<sup>p</sup> bytes (2 en este caso), para cada dirección se usarán los últimos p bits de la derecha para indicar en qué posición de la línea se guardará el byte correspondiente a esa dirección; y los siguientes k bits (a la izquierda de esos p bits) se emplean para indicar el número de línea dónde estará dicho byte. En el ejemplo, k = 2 bits de línea (líneas 00, 01, 10, 11), y p=1 bit (el byte puede estar en la posición 0 ó 1 dentro de una línea).

Como se verá, para cada dirección de memoria, los t bits que están a la izquierda de los k bits de línea (en este caso t=3) sirven para que el controlador determine si el contenido de dicha dirección está o no en el caché. Dichos t bits constituyen el "tag" (etiqueta) de una dirección. Así, 010 es el tag de la dirección 010011.

Puesto que cada línea guarda un bloque de 2<sup>p</sup> bytes consecutivos de memoria, las direcciones de los bytes de un bloque que está en una línea tendrán igual tag, ya que ellas sólo difieren en sus últimos p bits de la derecha.

Determinaremos t, k, p para una memoria DRAM de 2<sup>32</sup> bytes (4 GB) que usa un caché SRAM de 512 KB = 2<sup>19</sup> bytes organizado en 32768 líneas = 2<sup>15</sup> líneas que guardan 16 bytes = 2<sup>4</sup> bytes (se verifica que 2<sup>15</sup>×2<sup>4</sup> = 2<sup>19</sup> bytes).

Puesto que 2<sup>15</sup> = 2<sup>k</sup> resulta que k=15 (números de línea de 0000000000000000 a 11111111111111), y p=4 (en cada línea cada una de las 2<sup>p</sup>=16 posiciones que guardan un byte se identifica por los últimos p=4 bits de su dirección: 0000 a 1111). Dado que cada dirección tiene 32 bits, resulta: t = 32 - (15 + 4) = 13 bits para cada tag. Por ejemplo, en una dirección de 32 bits como 10110100111001101110111010101101, debe considerarse que 1011010011100 es el tag; 110111011101010 es el número de línea, y 1101 indica dentro de ésta la posición del byte correspondiente a la dirección dada. Son comunes cachés de 32 bytes por línea.

Por lo tanto la estructura del caché determina los números de bits de k y p, siendo que los de t son los que restan del número n de bits de cada dirección de memoria:  $t = n - (k + p)$ .

El controlador de caché posee una "tag memory": una SRAM que contiene un tag por cada línea del caché, que es el mismo para todas las direcciones de los  $2^p$  bytes que ella guarda (fig. 1.77.b), pues las mismas sólo difieren en sus últimos  $p$  bits de la derecha ( $p = 1$  en la fig. 1.77.b).

Cuando luego de un fallo se transfiere un bloque de bytes consecutivos de memoria hacia una línea del caché, el controlador escribe en su "tag memory" el tag de las direcciones de esos bytes.

Si luego la UCP ordena leer el contenido de una dirección de memoria, ella será interceptada por el controlador, que rápidamente comparará el tag de la misma con el tag de la "tag memory" vinculado al número de línea correspondiente a esa dirección. Si el circuito comparador indica igualdad de tags, es un acierto, o sea que en la línea del caché correspondiente a ese tag está el byte que la UCP direccionó, y por lo tanto dicho byte será proporcionado por el caché casi al mismo tiempo de la indicación de acierto.

De no ser así resulta un fallo, por lo que el controlador permitirá leer en la memoria el contenido de dicha dirección (fig. 1.77), el cual llegará a la UCP (en la que ocurrirán "wait states"), y el mismo también pasará a la línea del caché citada -que antes no lo contenía- junto con el byte de la dirección adyacente, reemplazándose así el anterior contenido de esa línea. Asimismo, en la "tag memory" se cambia el tag de esta línea por el tag del nuevo bloque.

Trataremos de concretar esta descripción en el caché "básico" de la fig. 1.77.b, para lo cual supondremos que sucedieron las siguientes asignaciones, siendo que al hacer cada asignación se escribió el tag de cada línea -entre paréntesis- en la tag memory del controlador.

línea 00 : contenidos 11011101 y 00010101 de las direcciones 011000 y 011001 (011)

línea 01 : contenidos 01010001 y 10010111 de las direcciones 011010 y 011011 (011)

línea 10 : contenidos 11111100 y 01111101 de las direcciones 111100 y 111101 (111)

línea 11 : contenidos 11010101 y 00110111 de las direcciones 001110 y 001111 (001)

Supongamos ahora que la UCP quiera leer la memoria, para lo cual por el bus de direcciones envía la dirección 011010, que será interceptada por el controlador. Así se determina que se trata de la línea 01, y se compara su tag (011) con el de la dirección. Como ambos son iguales resulta un acierto, por lo que el caché proporcionará el contenido 01010001 correspondiente a la posición 0 de esa línea, siendo 0 el último bit de la dirección. A tal fin a la salida del caché existe un multiplexor que mediante el valor de este último bit permite seleccionar si por sus salidas aparecerá el byte de las posiciones (campos) 0 ó 1.

Si la dirección hubiese sido 101010, al comparar para la línea 01 su tag existente (011) con el 101 de esa dirección, resultaría un fallo. Entonces se accederá a memoria para obtener los contenidos de las direcciones 101010 y 101011, los cuales reemplazarán a los anteriores 01010001 y 10010111. Luego el nuevo contenido de la posición 0 de dicha línea pasará a la UCP. En la "tag memory" se reemplazará 011 por 101. Como en el bloque traído de memoria también está el byte de la dirección *siguiente* (101011) a la solicitada, muy probablemente dicho byte será pedido por la UCP en el próximo acceso, por lo cual en el mismo ocurrirá un acierto. Teniendo en cuenta que en la práctica un bloque puede tener 16 ó 32 bytes, al ser llevado al caché junto con el byte solicitado se guardarán bytes de las direcciones consecutivas siguientes a la que ocasionó el fallo. De este modo al llegar más bytes que los solicitados, *en forma automática* se trata de aprovechar la proximidad espacial, pues es dable esperar que los bytes que llegaron "de más" anticipadamente en el bloque, sean los próximos en ser accedidos en el caché, resultando una sucesión de aciertos.

Lo anterior también sirve para subrayar el hecho de que *un caché recibe información desde memoria solamente cuando ocurre un fallo*, o sea en el caso que el byte direccionado por la UCP no se encuentra en él. Ello implica por un lado, que la información fluye de memoria hacia el caché en forma discontinua, pues la UCP accede continuamente al caché. Por lo tanto el tráfico desde memoria hacia la UCP se reduce notablemente, posibilitando a su vez que la memoria pueda ser accedida con menos conflictos con la UCP por parte de dispositivos de E/S que realizan ADM: acceso directo a memoria (fig. 1.72). Por otra parte se verifica que la próxima información a la que seguramente la UCP accederá llega *en forma automática*: como se accede a un bloque de direcciones consecutivas de memoria, junto con los bytes que se direccionaron (que provocaron el fallo) vienen a la línea del caché que se reemplaza los bytes de las direcciones siguientes que próximamente serán accedido por la UCP.

Además de seguido, por lo general los controladores también piden el bloque con las direcciones siguientes al bloque que contenía la dirección del fallo, el cual irá a otra línea del caché. Así en éste por lo menos existirán dos bloques (líneas) con direcciones consecutivas.

En la "tag memory" SRAM se determina si un contenido de ella (tag) *está o no*, para lo cual se *compara* el tag de la dirección generada por la UCP con el tag de la línea del caché accedida, utilizando un circuito comparador por línea.

Este funcionamiento es propio de las denominadas "**memorias asociativas**", en las cuales se puede así determinar si un número -en este caso el de un tag- se encuentra o no en ellas.

Para no perder tiempo, a la par que se accede al tag de una línea mediante el campo de  $k$  bits de la dirección (00, 01, 10 ó 11 en el ejemplo), también se accede a una copia del bloque contenido en esa línea, para que aparezca en las salidas del caché, haya acierto o fallo. Si hay acierto, parte del mismo (seleccionado con los  $p$  bits de la dirección) irá a la UCP. En caso de fallo simplemente no pasa a la UCP.

Un caché con más bytes por línea, responderá mejor cuando la UCP accede a una zona con buena proximidad espacial, como ser una larga secuencia de instrucciones consecutivas. En cambio cuando ocurre un fallo (< 10 % de los accesos) tendrá que acceder a un mayor número de bytes sucesivos en memoria que un caché con menos bytes por línea.

En el controlador (fig. 1.77.b) además de los "tags" existe para cada línea un bit de validación (V) que sirve al controlador para saber si los contenidos de una línea son válidos. Por ejemplo, cuando se enciende el computador todas las líneas tendrán su V=0, de modo que si la UCP genera una dirección que por casualidad está en el caché así se indica que los contenidos de la línea correspondiente son inválidos, puesto que son "basura" accidental. Luego, en los primeros instantes de la ejecución de un programa, las primeras direcciones de instrucciones y datos no estarán en la "tag memory", ocurriendo un gran número de fallos; pero con cada nuevo bloque que es traído de memoria, el mismo contendrá bytes de direcciones que siguen a la que la UCP quiere acceder, con lo cual comenzará a funcionar la proximidad espacial y temporal. Asimismo, para cada línea donde hubo reemplazo de bloque, el controlador pondrá su V=1.

Los caches de correspondencia directa si bien presentan una circuitería sencilla, el hecho de que se adjudique cada bloque de memoria siempre a la misma línea (por ej cada dirección xxx01x irá a la línea 01) puede ocasionar a veces una disminución en la tasa de aciertos. Tal sería el caso en que se ejecutan instrucciones que direccionan en forma repetitiva datos de dos bloques de memoria, los cuales por sus direcciones deban ser adjudicados, casualmente, a una misma línea del caché. Por lo tanto, en ese lapso constantemente el controlador debería ir cambiando el bloque que está en esa línea, pasando al cache en forma alternada uno y otro de esos dos bloques de la memoria. Estas situaciones se solucionan con compiladores más inteligentes, que no generen este tipo de accesos.

Los problemas del tipo anterior no pueden ocurrir en los cachés con *correspondencia totalmente asociativa* donde *cualquier* bloque puede ser adjudicado por el controlador a *cualquier* línea, lo cual permite aprovechar mejor el principio de proximidad en el acceso al caché, posibilitando una mayor tasa de aciertos.

En la fig 1.77.b, por ejemplo el bloque de direcciones 101010 y 101011 sería factible asignarlo a una cualquiera de las líneas del caché (suponiendo igualmente 8), según mejor convenga. y no sólo a la línea 01.

El controlador de este caché cumplimenta un algoritmo que permite determinar a cuál línea (por ej la *menos accedida últimamente*) se asignará el bloque de memoria que debe entrar en el caché, en reemplazo de otro bloque. O sea que el número de línea no se obtiene a partir de la dirección que produjo un fallo.

Para determinar si hay o no acierto se requiere comparar el tag de la dirección a la que la UCP quiere acceder, con el tag de cada una de las líneas en la "tag memory". Puesto que se requiere rapidez habrá que hacer múltiples comparaciones simultáneas, resultando compleja y limitada en velocidad la circuitería de comparación. Además los comparadores deben tener muchas entradas, dado que cada tag comprende bastantes bits, pues son los de cada dirección menos los  $p$  bits de la derecha (para la fig. xxx, el tag sería de  $t=5$  bits, y  $p=1$ ). Por ello sólo esta correspondencia sólo es viable en cachés relativamente pequeños, con pocas líneas.

A diferencia de esto, en el caché de correspondencia directa tratado, cada bloque a asignar iba una línea predeterminada, fija, según su dirección. En él se reemplaza al bloque existente, sin considerar si éste fue o no recientemente accedido, o sea que no se explota a fondo la proximidad temporal de los contenidos.

Además de seguido, por lo general los controladores también piden el bloque con las direcciones siguientes al bloque que contenía la dirección del fallo, el cual irá a otra línea del caché. Así en éste por lo menos existirán dos bloques (líneas) con direcciones consecutivas.

En la "tag memory" SRAM se determina si un contenido de ella (tag) *está o no*, para lo cual se compara el tag de la dirección generada por la UCP con el tag de la línea del caché accedida, utilizando un circuito comparador por línea.

Este funcionamiento es propio de las denominadas "**memorias asociativas**", en las cuales se puede así determinar si un número -en este caso el de un tag- se encuentra o no en ellas.

Para no perder tiempo, a la par que se accede al tag de una línea mediante el campo de  $k$  bits de la dirección (00, 01, 10 ó 11 en el ejemplo), también se accede a una copia del bloque contenido en esa línea, para que aparezca en las salidas del caché, haya acierto o fallo. Si hay acierto, parte del mismo (seleccionado con los  $p$  bits de la dirección) irá a la UCP. En caso de fallo simplemente no pasa a la UCP.

Un caché con más bytes por línea, responderá mejor cuando la UCP accede a una zona con buena proximidad espacial, como ser una larga secuencia de instrucciones consecutivas. En cambio cuando ocurre un fallo (< 10 % de los accesos) tendrá que acceder a un mayor número de bytes sucesivos en memoria que un caché con menos bytes por línea.

En el controlador (fig. 1.77.b) además de los "tags" existe para cada línea un bit de validación ( $V$ ) que sirve al controlador para saber si los contenidos de una línea son válidos. Por ejemplo, cuando se enciende el computador todas las líneas tendrán su  $V=0$ , de modo que si la UCP genera una dirección que por casualidad está en el caché así se indica que los contenidos de la línea correspondiente son inválidos, puesto que son "basura" accidental. Luego, en los primeros instantes de la ejecución de un programa, las primeras direcciones de instrucciones y datos no estarán en la "tag memory", ocurriendo un gran número de fallos; pero con cada nuevo bloque que es traído de memoria, el mismo contendrá bytes de direcciones que siguen a la que la UCP quiere acceder, con lo cual comenzará a funcionar la proximidad espacial y temporal. Asimismo, para cada línea donde hubo reemplazo de bloque, el controlador pondrá su  $V=1$ .

Los caches de correspondencia directa si bien presentan una circuitería sencilla, el hecho de que se adjudique cada bloque de memoria siempre a la misma línea (por ej cada dirección xxx01x irá a la línea 01) puede ocasionar a veces una disminución en la tasa de aciertos. Tal sería el caso en que se ejecutan instrucciones que direccionan en forma repetitiva datos de dos bloques de memoria, los cuales por sus direcciones deban ser adjudicados, casualmente, a una misma línea del caché. Por lo tanto, en ese lapso constantemente el controlador debería ir cambiando el bloque que está en esa línea, pasando al cache en forma alternada uno y otro de esos dos bloques de la memoria. Estas situaciones se solucionan con compiladores más inteligentes, que no generen este tipo de accesos.

Los problemas del tipo anterior no pueden ocurrir en los cachés con *correspondencia totalmente asociativa* donde *cualquier* bloque puede ser adjudicado por el controlador a *cualquier* línea, lo cual permite aprovechar mejor el principio de proximidad en el acceso al caché, posibilitando una mayor tasa de aciertos.

En la fig 1.77.b, por ejemplo el bloque de direcciones 101010 y 101011 sería factible asignarlo a una cualquiera de las líneas del caché (suponiendo igualmente 8), según mejor convenga, y no sólo a la línea 01.

El controlador de este caché cumplimenta un algoritmo que permite determinar a cuál línea (por ej *la menos accedida últimamente*) se asignará el bloque de memoria que debe entrar en el caché, en reemplazo de otro bloque. O sea que el número de línea no se obtiene a partir de la dirección que produjo un fallo.

Para determinar si hay o no acierto se requiere comparar el tag de la dirección a la que la UCP quiere acceder, con el tag de cada una de las líneas en la "tag memory". Puesto que se requiere rapidez habrá que hacer múltiples comparaciones simultáneas, resultando compleja y limitada en velocidad la circuitería de comparación. Además los comparadores deben tener muchas entradas, dado que cada tag comprende bastantes bits, pues son los de cada dirección menos los  $p$  bits de la derecha (para la fig. xxx, el tag sería de  $t=5$  bits, y  $p=1$ ). Por ello sólo esta correspondencia sólo es viable en cachés relativamente pequeños, con pocas líneas.

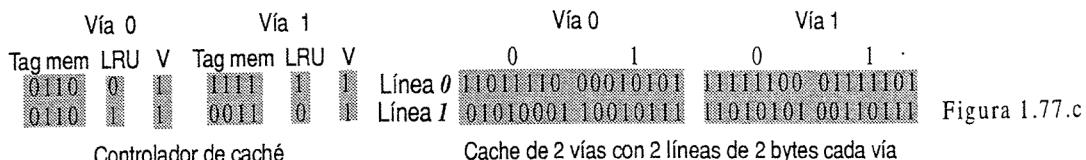
A diferencia de esto, en el caché de correspondencia directa tratado, cada bloque a asignar iba una línea predeterminada, fija, según su dirección. En él se reemplaza al bloque existente, sin considerar si éste fue o no recientemente accedido, o sea que no se explota a fondo la proximidad temporal de los contenidos.

La correspondencia asociativa por conjunto (de  $2^k$  líneas) con  $c$  conjuntos o vías (que implican  $c$  alternativas de asignación), es una solución de compromiso entre la correspondencia directa y la totalmente asociativa.

Como en la fig. 1.77.b el número de línea donde irá un bloque está determinado por  $k$  bits comunes a todas las direcciones de los bytes de ese bloque. Pero hay  $c$  líneas (vias) con igual número de línea. El controlador conforme a un algoritmo, adjudica el bloque a una de dichas  $c$  líneas: la menos accedida últimamente.

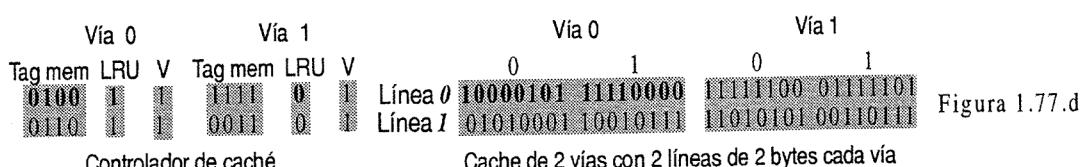
Reformaremos (fig. 1.77.c) el cache de 4 líneas de la fig. 1.77.b de modo de tener para cada número de línea 2 líneas que guardan un bloque de 2 bytes cada una. Así se forman  $c=2$  conjuntos de líneas designados vía 0 y vía 1. En cada vía la posición de cada byte se identifica con 0 y 1.

Todas las direcciones del tipo xxxx0x irán a la línea 0, y las del tipo xxxx1x a la línea 1, siendo que el último bit de la derecha indica si irá a la posición 0 ó 1 de la línea. Ahora cada tag tendrá 4 bits.



Así, el bloque de direcciones 011000 y 011001 (tag 0110) irá siempre a la línea 0, y el controlador decidirá si será la línea 0 de la vía 0 ó la línea 0 de la vía 1. Para tal fin además del tag, cada línea de cada vía, junto con su bit de validez V, tiene un bit "LRU" (least recently used, o sea usada menos recientemente). Cada vez que una línea de una vía es accedida, el controlador pone en 1 su bit LRU, y 0 el bit LRU de igual número de línea de la otra vía.

En la fig. 1.77.c para un caché de igual tamaño que el de la fig. 1.77.b, aparecen reubicados los mismos bloques de esta última como se indica, con un valor supuesto para el bit LRU de cada uno. Supongamos que hubo un fallo al querer asignar el bloque de direcciones 010000 y 010001 (de contenidos 10000101 y 11110000, no dibujados en la fig. 1.77.b): El mismo deberá ir al número de línea 0, y se asignará a la vía cuyo bit LRU vale 0, pues ello implica que el bloque que está en dicha vía (0 en este caso) ha sido accedido antes que el bloque de la otra vía, dado que éste con LRU=1 es más probable que sea accedido próximamente. Asimismo se colocará el tag (0100) en lugar del tag del bloque reemplazado. La nueva situación del caché y del controlador aparece en la fig. 1.77.d Si luego la UCP quiere acceder otra vez a la dirección 010000, el controlador comparará el tag 0100 con los dos tags de los dos bloques de número de línea 0 para determinar en qué vía está el bloque. En cada acierto, un multiplexor -ubicado en las salidas del caché- seleccionará por su tag el bloque accedido entre los dos posibles para un número de línea, el cuál aparecerá en dichas salidas.



Con este esquema se evita el conflicto antes citado en la correspondencia directa, cuando dos bloques que iban a un mismo número de línea eran accedidos en forma alternada. Ahora estarían en igual número de línea, pero cada bloque en una vía distinta. La alternancia en el acceso sólo produciría el cambio de valor de los bits LRU de los dos bloques de igual número de línea.

Si bien estos cachés necesitan más comparadores que los de correspondencia directa y en consecuencia un mayor tiempo de acceso, son menos complejos y más rápidos que los de correspondencia total, lo que hace posible que resulten una buena solución de compromiso entre ambos.

Siguiendo el controlador tiene 4 posibilidades para asignar un bloque a un número de línea.

Si se tiene 4 vías, el controlador tiene 4 posibilidades para asignar un bloque a un tramo de linea.

La estrategia de usar dos cachés multinivel, uno muy pequeño dentro del procesador (level 1 - L1) y otro más grande en la memoria RAM (level 2 - L2).

La estrategia de usar dos cachés (L1 y L2), busca que L1 sea de acceso sea extra rápido, mientras que L2 se encargue

y otro externo (L2), busca que el sea de acceso rápido. Una vez que el L2 maneja los fallos de L1, de manera de minimizar los "wait states" de la UCP. Cuando el caché L2 accede a un bloque de memoria principal debido a un fallo, lo hace en modo "ráfaga" ("burst"), esto es el tiempo de acceso al primer byte del bloque (por ej. de 32 bytes) es mayor, pero los siguientes bytes por estar en direcciones consecutivas se acceden varias veces más rápido.

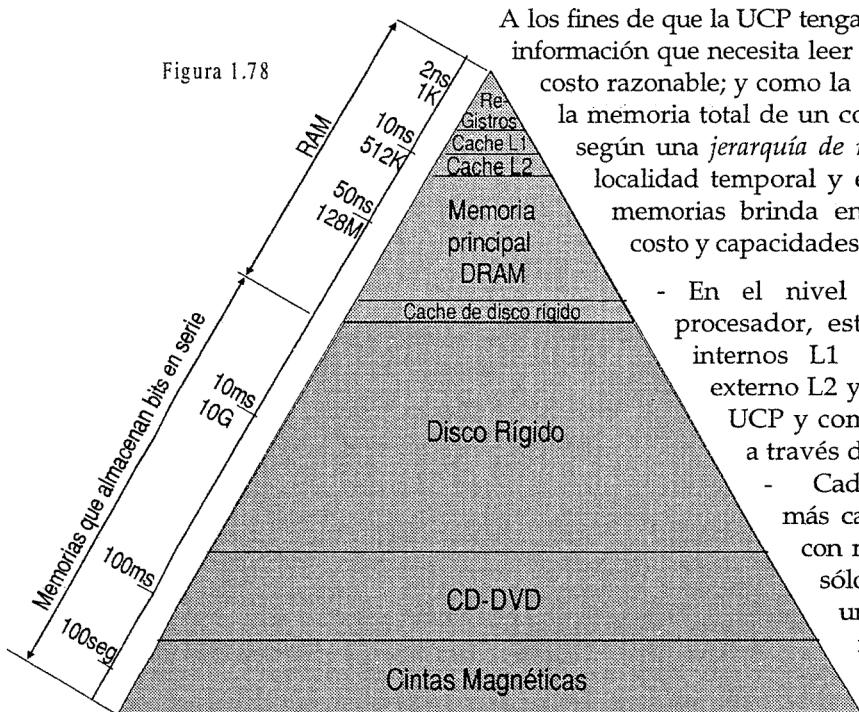
Hasta ahora hemos tratado las lecturas de instrucciones y datos en un caché, siendo que ellas predominan sobre las escrituras. Pero también la UCP requiere en menor grado guardar en memoria principal resultados de datos que ha procesado cuando una instrucción así lo ordena.

El controlador de caché también maneja la estrategia de escritura del caché y memoria principal. Una forma denominada "*write through*" ("a través" del caché) consiste en escribir (actualizar) simultáneamente ambos, lo cual conlleva tiempo. Para mejorar esto, suele usarse un "buffer" de escritura, que es escrito rápidamente junto con el caché, y que guarda resultados hasta que sean escritos en memoria. En caso que la dirección a actualizar no se encuentre en el caché, sólo se escribe la memoria.

Otra forma, "*write back*", o "post-escritura" consiste en marcar en el caché las líneas que la UC escribió en él, de forma que cuando la línea así marcada sea reemplazada, el bloque que contiene sea escrito en memoria principal. Esto último supone que durante un lapso de tiempo, el caché y la memoria tendrían información diferente, situación conocida como "*incoherencia*". Puede ocurrir entonces que una porción de memoria tengan información desactualizada. Para hacer por ejemplo una lectura de esa porción hacia el disco rígido, se debe acceder al caché, lo cual hace más complejo el hardware.

## ¿Qué es una jerarquía de memorias ?

Figura 1.78



A los fines de que la UCP tenga un tiempo de acceso a la cantidad de información que necesita leer o escribir lo más bajo posible y a un costo razonable; y como la memoria más lenta es la más barata, la memoria total de un computador está organizada (fig. 1.78) según una *jerarquía de niveles de memorias*. Ella combina la localidad temporal y espacial con lo que la tecnología de memorias brinda en cuanto a velocidades de acceso, costo y capacidades de almacenamiento. Así se tiene:

- En el nivel superior (fig. 1.78), dentro del procesador, está la memoria más rápida (cachés internos L1 y registros). Le siguen el caché externo L2 y la memoria principal, cercanos a la UCP y comunicados directamente con la UCP a través de buses externos.
- Cada nivel es más pequeño, más rápido, más caro por byte almacenado, y accedido con más frecuencia que el nivel inferior; y sólo puede intercambiar información con un nivel adyacente. Se busca tener la mayor capacidad con el menor costo por byte almacenado (como proveen los discos), a la par que el tiempo de

acceso de un nivel inferior a otro superior vecino sea lo menor posible, en especial para la UCP.

- La información contenida en un nivel también está en el nivel siguiente inferior.

Al tratar los cachés se vio que una UCP con una memoria principal DRAM de 64MB y  $t_{acc} = 50$  nseg. y un caché SRAM de 512KB y  $t_{acc} = 20$  nseg., pueden hacer que la UCP en más del 90% de los accesos que éstos sean de 20 nseg. O sea, es como si la UCP tuviera una memoria de 64MB y  $t_{acc} = 20$  nseg., siendo que resultaría muy caro tener 64MB de SRAM.

Con igual técnica, entre esa memoria DRAM y el disco duro se puede simular mediante el sistema operativo una *memoria virtual* de muchos GB, con un  $t_{acc}$  muy aceptable y sin costo adicional.

Igualmente los CDs amplían económicamente la capacidad del rígido.

Se trata siempre, entre dos niveles adyacentes, de soluciones de compromiso entre costo y velocidad.

En síntesis, para cada computador se busca construir una jerarquía de memorias tal que para el usuario simule una memoria con una capacidad de almacenamiento auxiliar virtualmente ilimitada, y con un tiempo de acceso tan rápido hoy día, como el primer nivel de memoria caché incorporado al procesador.