Tema 2
Unidades de Memoria y Entrada/Salida

2.1. Módulos de Entrada/Salida

Los módulos de entrada/salida (E/S) tienen las siguientes funciones básicas:

- Conectar con la CPU y memoria vía bus del sistema.
- Conectar con los periféricos mediante conexiones de datos particularizadas.

Las funciones o requerimientos principales de los módulos de E/S se agrupan en las siguientes categorías:

- Control y temporización.
- Comunicación con la CPU.
- Comunicación con los periféricos.
- Buffer de datos.
- Detección de errores.

Los recursos internos del sistema, tales como memoria o bus, están compartidos por distintas actividades, entre las que está E/S. Por ello, los módulos E/S tienen requerimientos de control y temporización. Por ejemplo, el control de transferencia de datos entre un periférico y la CPU debe seguir la siguiente secuencia:

1. La CPU pide al módulo E/S el estado del periférico deseado.

2. El módulo E/S proporciona el estado.

3. Si el periférico está listo, la CPU solicita la transferencia de datos por medio de un comando al módulo E/S.

4. El módulo E/S obtiene el dato del periférico.

5. El dato se transfiere desde el módulo a la CPU.

Si el sistema emplea un bus, cada interacción entre CPU y E/S implica uno o más arbitrajes de bus.

JUAN F. GUERRERO MARTÍNEZ
JOSÉ V. FRANCÉS VILLORA

Una tarea esencial del módulo E/S es servir de buffer de datos. Mientras la transferencia es muy rápida entre éste y la CPU, con los periféricos es mucho más lenta. El E/S realiza así la conversión de velocidades de transmisión.

Finalmente, el módulo E/S es frecuentemente responsable de realizar una detección de errores, que pueden ser avisados por el periférico (por ejemplo, falta de papel en una impresora) o producirse por fallos de transmisión (error de paridad de un carácter transmitido).

Estructura de un modulo E/S

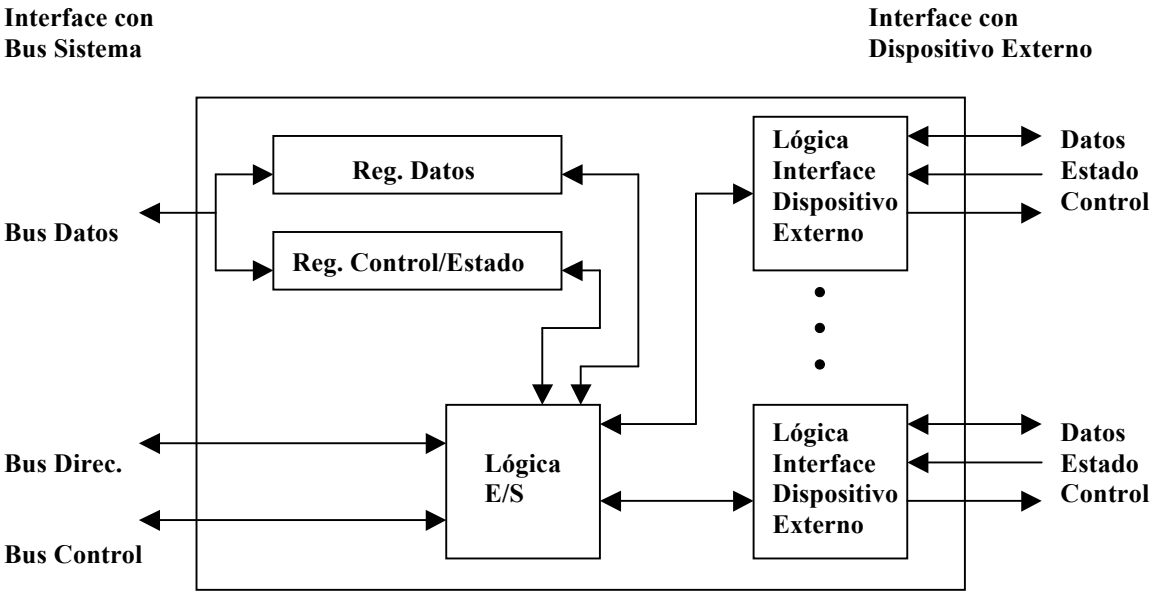

Figura 2.1. Diagrama de bloques de un módulo E/S.

En la figura 2.1 se muestra el diagrama general de un módulo E/S. Los datos transferidos desde o hacia el módulo se almacenan en uno o más registros de datos. Puede haber también uno o más registros de estados que proporcionan información sobre el estado actual. Un registro de estado puede actuar también como un registro de control, aceptando información de control de la CPU. La lógica del módulo interactúa con la CPU mediante un conjunto de líneas de control, que utiliza la CPU para ordenar funciones al módulo (lectura/escritura), o por éste mismo (líneas de arbitraje y estado). El módulo puede también reconocer y generar direcciones asociadas con los dispositivos que controla. Cada módulo

tiene una única dirección o, si controla más de un periférico, un único conjunto de direcciones (dirección base + direcciones para los distintos registros internos) El bus de direcciones es bidireccional para módulos E/S que pueden actuar de master del bus (DMA). Finalmente, el módulo E/S contiene lógica específica para conectar con el periférico que controla.

Un módulo de E/S permite a la CPU controlar al periférico únicamente con operaciones de lectura/escritura, ocultando otras funciones más específicas (por ejemplo, rebobinar la cinta para acceder al dato). Cuando el módulo asume funciones de alto nivel, se denomina canal E/S o procesador E/S. Los módulos de bajo nivel, que requieren control detallado, se denominan controladores E/S o controlador de periféricos.

Cuando la CPU, memoria principal e E/S comparten un bus común, hay dos modos de direccionamiento posibles: mapeado de memoria y aislado. Con E/S mapeado de memoria existe un único espacio de direcciones para las posiciones de memoria y los módulos E/S. La CPU trata los registros de datos y estado de los módulos como posiciones de memoria, y utiliza las mismas instrucciones máquina para acceder a memoria y E/S. Las líneas del bus de direcciones dan un número máximo de posiciones a direccionar, que estarán repartidas entre memoria y E/S en alguna proporción.

En el otro caso, el bus de control dispone de líneas de lectura/escritura en memoria más otras en entrada/salida. En este caso, el comando especifica cuándo una dirección se refiere a memoria o E/S. Se puede utilizar todo el rango del bus de direcciones para direccionar memoria y E/S de forma independiente, de ahí el nombre de E/S aislada.

La ventaja del primer tipo es que generalmente existen muchas más instrucciones referidas a memoria que a E/S, lo cual permite una programación más eficiente. Por contra, utiliza espacio del mapa de memoria.

2.2. Técnicas de entrada/salida

Entrada/salida programable

Con E/S programable, el módulo realiza la instrucción que le encarga la CPU y coloca los bits correspondientes en el registro de estado. Es responsabilidad de la CPU comprobar periódicamente el estado hasta que se complete la instrucción.

Supongamos, por ejemplo, el caso de lectura de datos de un periférico y almacenamiento en memoria. Para ejecutar la instrucción E/S, la CPU utiliza una dirección, que especifica el módulo particular, y un comando E/S (en este caso, lectura). Los datos se

leen de uno en uno. Para cada palabra, la CPU debe permanecer en un ciclo que comprobación hasta que determina que el dato está disponible en el registro de datos del módulo. Esta es la principal desventaja de este módulo: el gran consumo de tiempo de la CPU.

Entrada/salida controlada por interrupción

Una alternativa a la E/S programada, que requiere repetidas consultas por parte de la CPU, es el control por interrupción. La CPU encarga una operación al módulo y continúa realizando otras tareas. Cuando el módulo concluya la operación, interrumpirá a la CPU para tansferir los datos. La CPU realiza la transferencia y continúa después en el lugar en que se quedó con la otra tarea. Desde el punto de vista del módulo, éste recibe, por ejemplo, un comando READ de la CPU, lee el dato del periférico asociado y lo almacena en el registro de datos, produciendo una señal de interrupción a al CPU. El módulo espera hasta que responda ésta, pasa el dato y está disponible para otra operación. Desde el punto de vista de la CPU, ésta ordena un comando READ y sigue realizando otras tareas. Al terminar cada ciclo de instrucción, comprueba la existencia de interrupciones pendientes. Cuando existen, la CPU guarda el contexto de la tarea (generalmente PC y registros) y procesa la interrupción, en este caso lee el dato y lo almacena en memoria. A continuación recupera el contexto de la tarea y reasume la ejecución.

Este método es más eficiente que el anterior, pero implica un gran tiempo de dedicación de la CPU al servicio de interrupciones ya que la transferencia de cada palabra implica guardar y recuperar el contexto de la tarea en curso.

Hay dos cuestiones importantes en la implementación de E/S por interrupción:

1. Cómo determinar cuál de los módulos E/S ha producido la interrupción.
2. Si se producen interrupciones múltiples, priorizarlas.

Para la primera cuestión, existen 4 categorías:

- Múltiples líneas de interrupción.
- Consulta por software.
- Daisy chain (consulta por hardware, vectorizada).
- Arbitraje de bus (vectorizado).

La mejor solución es la existencia de múltiples líneas, pero resulta impracticable dedicar muchos pines de la CPU a esta cuestión. Consecuentemente, aunque se utilizasen varias líneas, siempre habría varios módulos conectados a cada línea, por lo que se debería usar alguna de las otras técnicas.

Una alternativa es la consulta por software. Cuando la CPU recibe una interrupción, salta a una rutina cuya misión es determinar qué módulo ha causado la interrupción, implementado operaciones de test, colocando la dirección de cada módulo en el bus de direcciones hasta que uno de ellos responda afirmativamente. Otra alternativa es disponer en cada módulo de un registro de estado que puede testear la CPU para identificar el origen de la interrupción. Una vez se ha identificado, la CPU salta a la rutina de servicio específica. La desventaja de esta técnica es que consume tiempo extra en la rutina de consulta.

Otra técnica más eficiente es la 'daisy chain'. que realiza una consulta hardware. En este caso, todos los módulos comparten una línea de interrupción. La línea de reconocimiento está conectada al primer módulo de la cadena y, a través de cada uno de ellos, al siguiente. Cuando la CPU quiere determinar el origen de la interrupción, activa un reconocimiento de interrupción que se propaga a través de la cadena hasta encontrar el módulo causante. Este módulo responde entonces colocando en el bus de datos una palabra, vector, que puede ser la dirección del módulo E/S o algún otro identificador. En cualquier caso, la CPU utiliza este vector como puntero a la rutina de servicio. Esta técnica se denomina 'interrupción vectorizada'.

Existe otra técnica que hace uso de las interrupciones vectorizadas: el arbitraje de bus. En este caso, un módulo debe primero conseguir el control del bus y posteriormente activar la línea de interrupción, de forma que sólo un módulo puede acceder a esta línea a la vez. Cuando la CPU detecta la interrupción, responde con una señal de reconocimiento y el módulo coloca su vector en el bus de datos.

La forma de implementar prioridades depende de la técnica utilizada. En la primera, la CPU elige el módulo conectado a la línea de mayor prioridad. Con consulta por software, el orden de consulta determina la prioridad. Similarmente, el orden de la daisy chain también lo determina. El arbitraje de bus utiliza un esquema comentado en el primer tema.

Existen módulos controladores de interrupciones, que incluyen la lógica de determinación del módulo interruptor, implementan prioridades y contienen los vectores de interrupción programados.

Acceso directo a memoria

Las técnicas anteriores necesitan la intervención de la CPU para transferir datos entre memoria y módulos E/S. Esto tiene dos inconvenientes:

- La velocidad de transferencia está limitada por el tiempo que necesita la CPU para testear y servir al periférico.
- La CPU debe ejecutar una serie de instrucciones por cada transferencia E/S.

Cuando se necesita transferir grandes cantidades de datos, el acceso directo a memoria (DMA) es más eficiente. Esto implica la inclusión de un módulo adicional al bus del sistema. El módulo de DMA es capaz de sustituir a la CPU tomando control del bus.

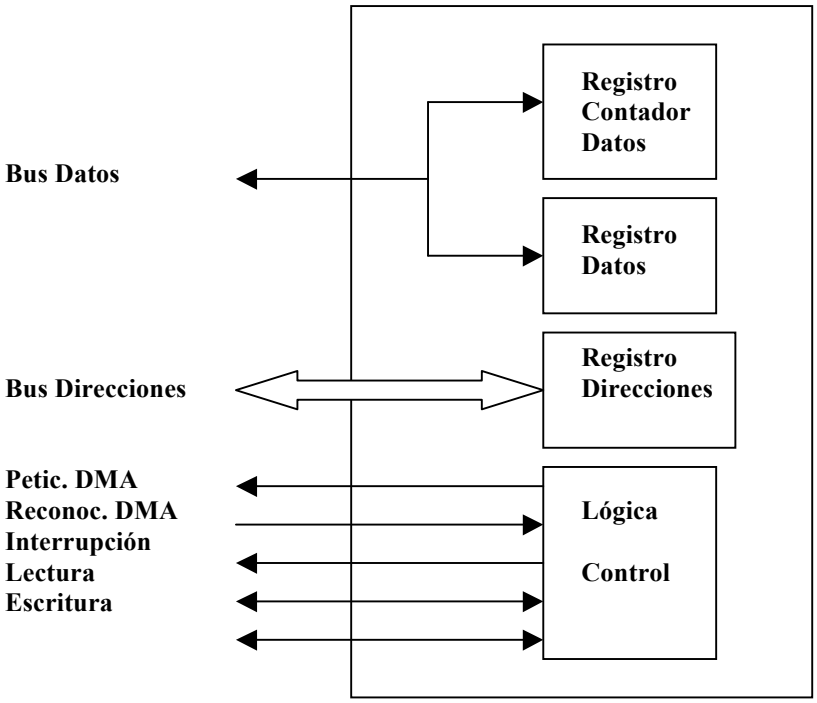

Figura 2.2. Diagrama de bloques de un módulo DMA.

Cuando la CPU desea leer o escribir un bloque de datos, envía un comando al módulo DMA con la siguiente información:

- Petición de lectura/escritura. (CONTROL)
- Dirección del módulo E/S implicado. (REG. DIRECCIONES)
- La posición de comienzo en memoria de la que leer/escribir. (REG. DIRECCIONES)
- El número de palabras a leer/escribir. (CONTADOR)

La CPU puede continuar con otra tarea, mientras el DMA transfiere el bloque de datos, palabra a palabra. Cuando la transferencia se ha completado, el DMA interrumpe a la CPU.

El módulo DMA necesita controlar el bus para realizar la transferencia, sólo cuando la CPU no lo utiliza o suspendiendo temporalmente el funcionamiento de la CPU. Esta última técnica se conoce como 'robo de ciclo'.

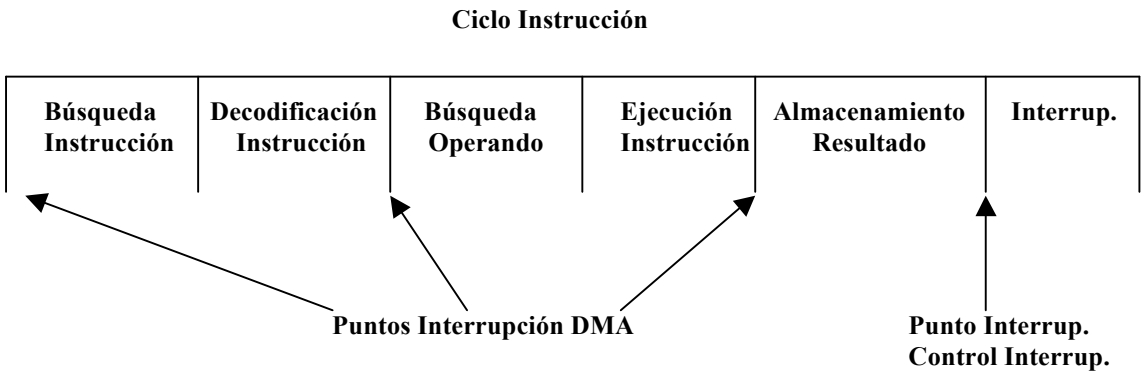

Figura 2.3. Puntos de interrupción durante el ciclo de instrucción.

En la figura anterior se muestran los puntos en los que la CPU debe ser suspendida por el DMA, coincidiendo con un momento antes de necesitar usar el bus, haciendo una pausa durante un ciclo. El efecto que produce esto es ralentizar la ejecución de la CPU. No obstante, para transferencias de múltiples palabras, la técnica DMA es más eficiente que las otras.

Existen diversas maneras de implementar el mecanismo DMA. En el primer caso, todos los módulos comparten el mismo bus. El DMA, que actúa como una CPU subsidiaria, utiliza E/S programada para intercambiar datos entre memoria e E/S. Esta configuración es la más barata e ineficiente.

El número de ciclos de bus requeridos para la transferencia disminuye integrando las funciones de DMA e E/S, mediante la existencia de conexiones entre el módulo DMA y los E/S distintos del bus. Por último, puede generalizarse este concepto utilizando un bus separado para E/S, lo que reduce el número de interfaces entre E/S y DMA a uno y permite una expansión fácil. En los dos últimos casos, el bus del sistema sólo es utilizado por el DMA para intercambiar datos con la memoria.

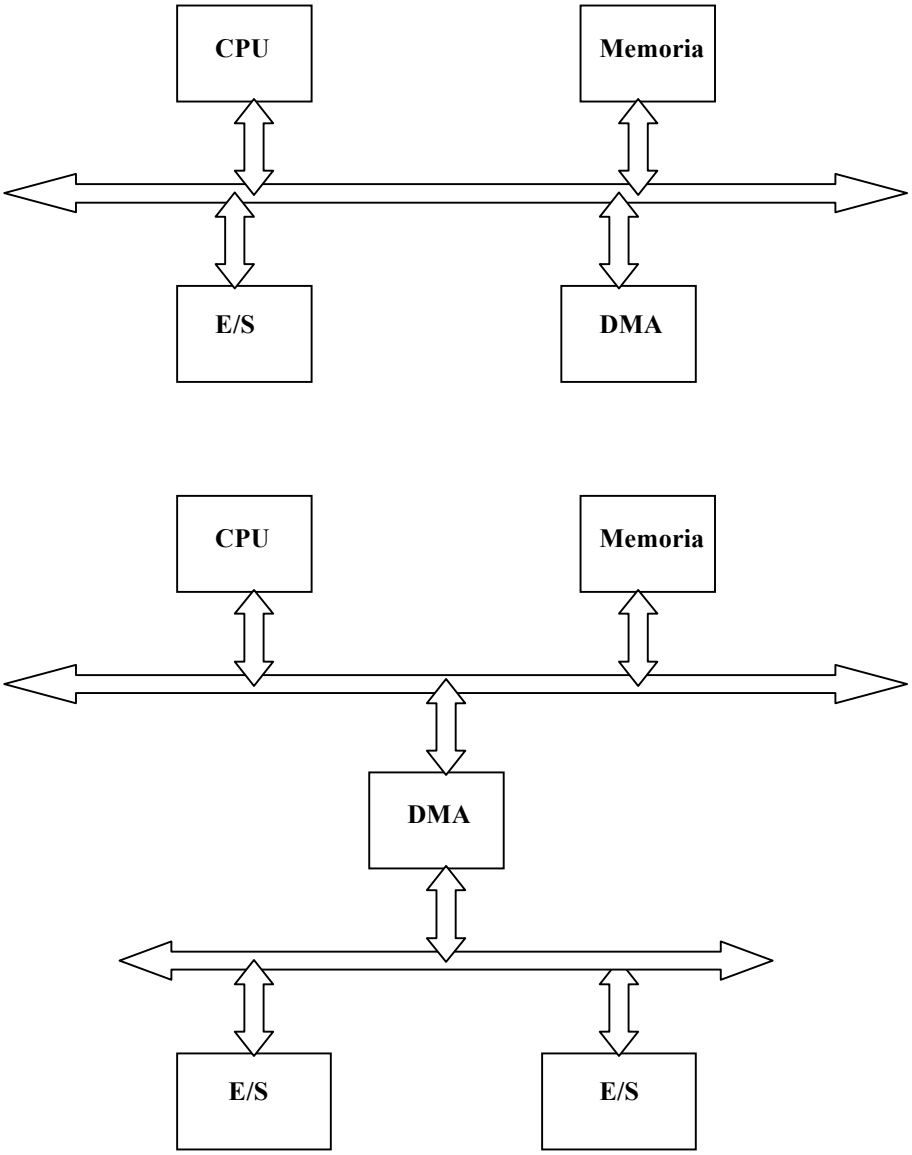

Figura 2.4. Estructuras de buses E/S.

2.3. Características de los sistemas de memoria

No existe una tecnología que sea óptima en todos los aspectos para la implementación de la memoria de un computador, por lo que éste suele estar equipado con una serie de subsistemas organizados jerárquicamente, algunos internos al sistema (directamente accesibles por la CPU) y otros externos (accesibles vía E/S).

Podemos clasificar los sistemas de memoria de acuerdo a las siguientes características:

LOCALIZACION

- CPU (registros+mem.control)
- Interna (principal)
- Externa (secundaria)

VELOCIDAD

- Tiempo acceso
- Tiempo de ciclo
- Razón de transferencia

CAPACIDAD

- Longitud de palabra
- Número de palabras

TIPO FISICO

- Semiconductor
- Superficie magnética
- Soporte óptico

UNIDAD TRANSFERENCIA

- Palabra
- Bloque

CARACTERISTICAS FISICAS

- Volátil/No volátil
- Borrable / No borrable

METODO ACCESO

- Acceso secuencial
- Acceso directo
- Acceso aleatorio
- Acceso asociativo

ORGANIZACION

LOCALIZACION.

No toda la memoria interna es memoria principal. La CPU puede acceder, y necesita, de otro tipo de memoria como son los registros internos o la memoria a la que accede la unidad de control en sistemas microprogramados. La memoria externa está constituida por dispositivos de almacenamiento periférico, tales como disco y cinta, que son accesibles por la CPU vía controladores de E/S.

CAPACIDAD.

Para memoria interna, suele especificarse en términos de bytes o words. Longitudes típicas de palabras son 8, 16 y 32 bits.

UNIDAD DE TRANSFERENCIA.

Para memoria interna, la unidad de transferencia es igual al número de líneas de datos del módulo de memoria. Esto es frecuentemente igual a la longitud de palabra, pero no siempre. Por ejemplo, el CRAY-1 tiene bus de datos de 64 bits pero representa los enteros en 24 bits. Por otra parte, aunque generalmente la unidad direccionable es la palabra, algunos sistemas permiten direccionar a nivel de byte y múltiplos de byte. En general, la unidad de transferencia es el número de bits transmitido a la memoria a la vez. No necesariamente coincidirá con una palabra o una unidad direccionable (para memoria externa, la transferencia se suele hacer en unidades más largas que una palabra, denominándose entonces bloques).

METODOS DE ACCESO.

Existen 4 tipos básicos:

1. Acceso secuencial: la memoria se organiza en unidades de datos llamados 'records'. El acceso debe hacerse siguiendo una secuencia lineal específica. La información de direccionamiento almacenada se usa para separar records durante el acceso. Se utiliza un mecanismo lectura/escritura compartido, que es posicionado en el lugar deseado desechando los record intermedios que encuentra en el camino. Por tanto, el tiempo de acceso a un record arbitrario depende de su posición inicial. Un ejemplo de estos dispositivos son las unidades de cinta.

2. Acceso directo: Implica también un mecanismo compartido de lectura/escritura. No obstante, los bloques individuales o records tienen una única dirección basada en su localización física. El acceso se realiza buscando una vecindad general a la psoición,

más búsqueda secuencial o esperando encontrar la posición final. De nuevo, el tiempo es variable. Las unidades de disco son dispositivos de acceso directo.

3. Acceso aleatorio: Cada posición direccionable tiene un único mecanismo físico de direccionamiento. El tiempo de acceso a una posición dada es independiente de la secuencia previa. La memoria principal es de este tipo.

4. Asociativo: Este tipo de memoria de acceso aleatorio permite hacer una comparación de las posiciones de los bits deseados en una palabra para un patrón especificado y hacerlo para todas las palabras simultáneamente. Por tanto, una palabra es accedida basándose en una porción de su contenido más que en su dirección. Como ocurre con la memoria de acceso aleatorio ordinaria, cada posición tiene su propio mecanismo de direccioamiento, y el tiempo de acceso es constante . Las memorias caché utilizan este tipo de acceso.

VELOCIDAD.

1. Tiempo de acceso: para memorias de acceso aleatorio, es el tiempo que cuesta realizar una operación de lectura/escritura, es decir, el tiempo transcurrido desde el instante en que una dirección se presenta a memoria y el instante en que el dato está almacenado o está disponible para su lectura. Para memorias no aleatorias, correspondería al tiempo necesario para que el mecanismo de lectura/escritura estuviese posicionado en la posición deseada.

2. Tiempo de ciclo de memoria: fundamentalmente aplicado a memoria aleatoria, corresponde al tiempo de acceso más el adicional necesario antes de que se pueda realizar un segundo acceso. Este tiempo adicional puede ser necesario para que desaparezcan los transitorios o para regenerar datos en memorias dinámicas.

3. Razón de transferencia: Es la razón a la que se pueden transferir datos con la memoria. Para memorias aleatorias, es igual a la inversa del tiempo de ciclo. Para no aleatorias, se cumple la relación:

$$T_N = T_A + N/R$$

donde

T_N = tiempo promedio para leer/escribir N bits.

T_A = tiempo de acceso promedio.

N = número de bits.

R = razón de transferencia en bits/segundo (bps).

TIPO FISICO.

Los tipos más comunes actualmente son memoria de semiconductores, usando tecnología LSI o VLSI, memoria de superficie magnética, utilizada para discos y cintas, y soporte óptico (CD-ROM).

CARACTERISTICAS FISICAS.

En memoria volátil, la información decae de forma natural o se pierde cuando se desconecta la alimentación. En memoria no volátil, la información grabada permanece mientras no se cambie de forma deliberada: no se necesita alimentación para mantener la información. Las memorias no borrables no pueden ser alteradas (ROM).

ORGANIZACION.

La organización es una clave importante en el caso de diseño de memorias de acceso aleatorio. Se entiende por organización al arreglo físico de los bits para formar palabras. Por ejemplo, una memoria de 1 kbit puede organizarse en 512 palabras de 2 bits o 128 palabras de 4 bits, etc.

Jerarquía de memoria

Hay tres características básicas que definen el diseño de un sistema de memoria: capacidad, tiempo de acceso y coste. Existen diversas tecnologías para implementar la memoria; no obstante, para todas ellas se cumplen las siguientes relaciones:

- Menor tiempo de acceso, mayor coste por bit.
- Mayor capacidad, menor coste por bit.
- Mayor capacidad, mayor tiempo de acceso.

Para encontrar la solución óptima a un diseño, teniendo en cuenta las características contrapuestas comentadas, se emplea una jerarquía de memoria, es decir, una combinación jerárquica, organizada por niveles, de diferentes tecnologías con el fin de obtener la

combinación óptima para las tres características. Conforme se baja en nivel de jerarquía (ver figura), ocurre lo siguiente:

a) Decrece el coste por bit.

b) Se incrementa la capacidad.

c) Se incrementa el tiempo de acceso.

d) Se decrementa la frecuencia de acceso de la memoria por parte de la CPU.

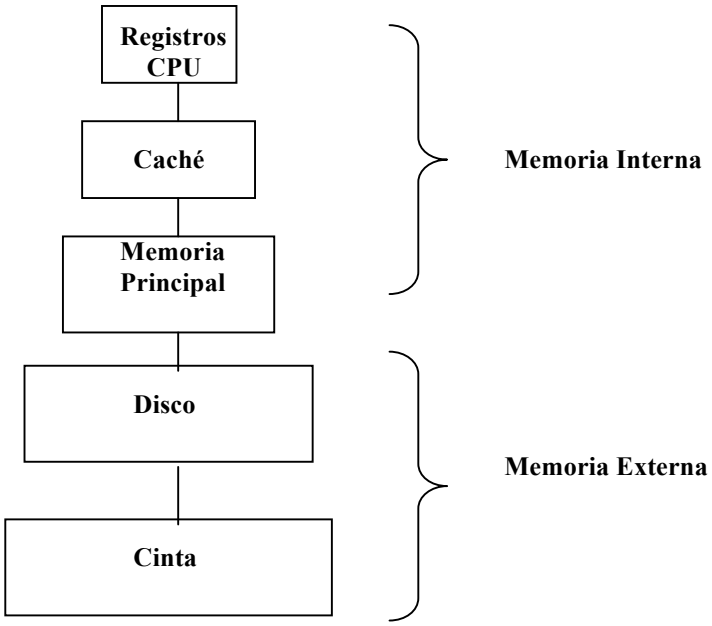

Figura 2.5. Jerarquía de memoria. Algunos niveles pueden subdividirse (por ejemplo, varios niveles de caché o distintos tipos de disco).

Utilizando, pues, diversas tecnologías, se pueden conseguir las condiciones a) hasta c). La validez de la condición d) se basa en la localización de referencias. Los programas tienden a contener bucles iterativos y subrutinas. Una vez se entra en uno de ellos, se repiten las referencias a un pequeño conjunto de instrucciones o datos. En base a esto, es posible organizar los datos en la jerarquía de manera que el porcentaje de accesos a cada nivel inferior es sustancialmente menor que a niveles superiores. Por tanto, una cantidad pequeña

de memoria rápida, pero más cara, se complementa con una cantidad mayor de memoria lenta. La clave de esta organización es la menor frecuencia de acceso a la memoria lenta.

Para ilustrar de qué manera esta estructura reduce costes manteniendo un nivel predeterminado de características, supongamos que la CPU accede a 2 niveles de memoria. El primero contiene 1.000 palabras y un tiempo de acceso de 1 μs. El nivel 2 contiene 100.000 palabras y un tiempo de 10 μs. Supongamos que si una palabra está en el nivel 1, la CPU accede directamente. Si está en el 2, el dato debe ser transferido a la 1 y después accedido (supondremos despreciable el tiempo que necesita la CPU para determinar en qué nivel está el dato). La figura muestra el tiempo de acceso total promedio como función del porcentaje de tiempo está presente en el nivel 1. Para porcentajes altos, el tiempo de acceso está más cercano al del nivel 1 que al del 2.

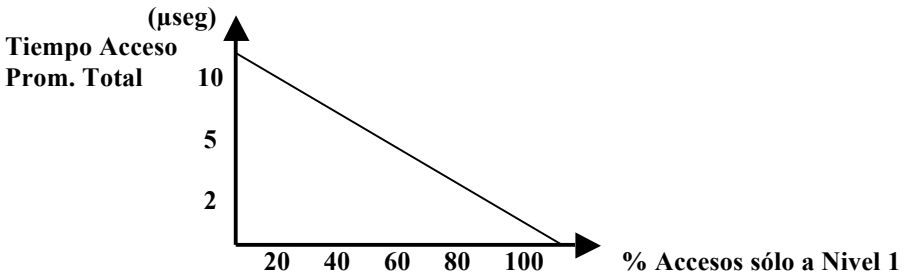

Figura 2.6. Prestaciones de una memoria de dos niveles.

Volvamos de nuevo al ejemplo de dos niveles. Supongamos inicialmente que todo el programa y datos están en el nivel 2. Las secciones actuales se cargan en el nivel 1. De vez en cuando, estas secciones se reemplazan por otras nuevas pero, en promedio, la mayoría de referencias se harán a datos contenidos en el nivel 1. Se obtiene, pues, un sistema de memoria de capacidad 101 kB con coste (suponiendo el 1er nivel 10 veces más caro que el segundo) 11 veces el coste y tiempo de acceso aproximadamente 3 veces el del primer nivel para un 80% de referencias a dicho nivel.

2.2. Tipos de memoria

Memoria caché

En todos los ciclos de instrucción, la CPU accede a memoria al menos una vez, para obtener el código de operación, o más veces si debe obtener operandos o guardar el resultado. Por tanto, la velocidad con que la CPU puede ejecutar instrucciones está ligada al ciclo de memoria. Idealmente, la memoria debería estar implementada con la misma tecnología que los registros de la CPU, pero esto es una estrategia demasiado cara. La solución es explotar el principio de localización, manteniendo una memoria rápida, pero de tamaño reducido, entre la CPU y la memoria principal. Esta memoria se conoce como memoria caché.

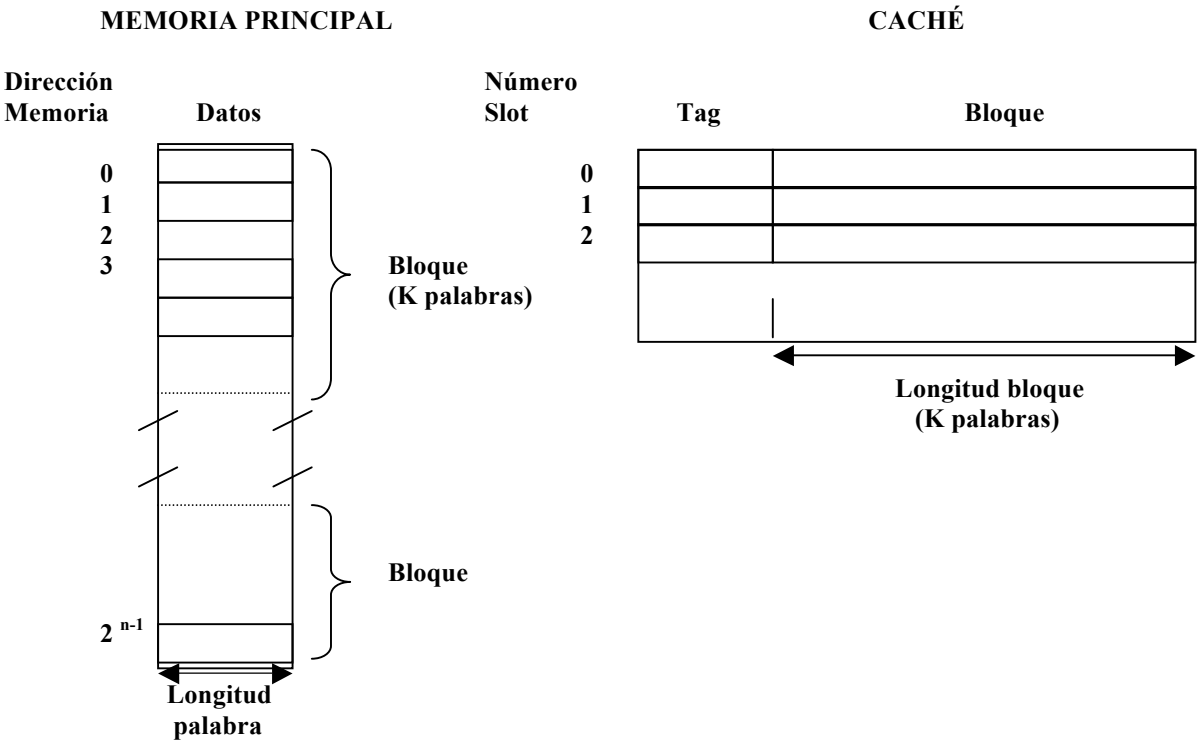

Figura 2.7. Estructura de caché/memoria principal.

La utilización de caché permite una aproximación a velocidades de memoria rápidas a la vez que tamaños de memoria grandes al precio de tipos de memoria de semiconductores más baratos. Para ello, se dispone de una memoria principal relativamente grande y lenta, junto con una pequeña y rápida caché. La caché contiene una copia de porciones de la

memoria principal. Cuando la CPU intenta leer un dato de memoria, se realiza una comprobación para determinar si está en la caché. Si está, se pasa el dato a la CPU. Si no, se lee un bloque (consistente en un número fijo de palabras) de la principal a la caché y se pasa el dato a la CPU. A causa del fenómeno de localización de referencias, cuando un bloque se trae a la caché para satisfacer una única referencia, es probable que se cumpla para otras referencias inmediatas.

Como se ve en la figura, la memoria principal consta de 2^n palabras direccionables, con direcciones de n-bits independientes. Desde el punto de vista de mapeado, se considera que consta de cierto número de bloques de longitud fija y K palabras cada uno. Es decir, hay $N_B = 2^n/K$ bloques. La caché consiste en N_S slots de K palabras cada uno, y el número de slots es considerablemente menor que el número de bloques de memoria ($N_S << N_B$). En un momento dado, un subconjunto de bloques de memoria reside en los slots de la caché. Si una palabra de un bloque de memoria es leída, ese bloque se transfiere a un slot de la caché. Puesto que hay más bloques que slots, un slot individual no puede dedicarse única y permanentemente a un bloque. Por tanto, cada slot incluye un identificador que indica qué bloque particular esta almacenado actualmente allí. El indicador es usualmente una porción de la dirección de memoria principal.

Elementos de diseño de la Caché

Los elementos clave en el diseño de la memoria caché son:

- TAMAÑO
- FUNCIÓN DE MAPEADO
  - Directo
  - Asociativo
  - Conjuntos asociativos
- ALGORITMO REEMPLAZAMIENTO
  - LRU (Menos Usado Recientemente)
  - FIFO
  - LFU (Menos Frecuentemente Usado)
  - Aleatorio
- MÉTODO ACTUALIZACIÓN
  - Write through
  - Write back
  - Write once
- TAMAÑO DEL BLOQUE

TAMAÑO.

Se pretende mantener el tamaño de la caché lo suficientemente pequeño para que el coste promedio por bit esté cercano al de la memoria principal y suficientemente grande para que el tiempo de acceso promedio esté cercano al de la caché.

Consideremos en primer lugar el coste:

$$C_S = \frac{C_C S_C + C_M S_M}{S_C + S_M}$$

donde:

C_T = coste promediado por bit, memoria principal + caché.

C_C = coste promediado por bit, caché.

C_M = coste promediado por bit, memoria principal.

S_C = tamaño de caché.

S_M = tamaño de memoria principal.

Deseamos que C_T sea aproximadamente igual a C_M. Puesto que $C_C >> C_M$, esto implica que $S_C << S_M$.

Consideremos ahora el tiempo de acceso. Este es función de las velocidades de la caché y la memoria principal, y también de la probabilidad de que una referencia de la CPU se encuentre en la caché. Esta probabilidad se conoce como 'hit ratio'. Tendremos:

$$T_S = H T_C + (1 - H) T_M$$

donde:

T_S = tiempo de acceso promediado del sistema.

T_C = tiempo de acceso a la caché.

T_M = tiempo de acceso a la memoria principal.

H = Hit ratio.

Deseamos que $T_S \approx T_C$. Puesto que $T_C << T_M$, se necesita una H cercana a 1.

Para mantener bajo el coste, nos interesa un tamaño de caché pequeño. Pero para conseguir un incremento significativo de prestaciones necesitamos una H alta. En general, la caché tiene tiempos de acceso de 5 a 10 veces más rápidos que la memoria principal. Por tanto, un H de 0.75 o mayor da unas características razonables.

FUNCION DE MAPEADO

Esta función relaciona bloques de memoria con posiciones de caché. Las técnicas más usuales: directa, asociativa y conjuntos asociativos.

Supongamos como ejemplo una caché de 1 Kbyte, y una memoria principal de 64 Kbytes. Los datos se transfieren desde memoria principal en bloques de K=8 bytes. Las cantidades de slots en caché y de bloques en memoria principal vienen dadas por:

$C_S = S_C / K = 1\ kB / 8 = 128$ $\qquad$ $C_B = S_C / K = 64\ kB / 8 = 8\ kbloques$

siendo:

K: número de bytes por bloque o slot
C_B: cantidad de bloques en memoria principal.

Puesto que hay menos slots de caché que bloques de memoria, se necesita un algoritmo para mapear los bloques en los slots, es decir, decidir en qué slot se almacena el bloque que se trae de memoria. Además, se necesitará un mecanismo que determine qué bloque está ocupando en un momento dado un slot. La técnica más simple es la de mapeado directo, que sólo permite a cada bloque ocupar un slot determinado. El mapeado cumple:

$$N_S = N_B \text{ módulo } C_S$$

donde:

N_S: número de slot de caché.

N_B: número de bloque de memoria principal.
C_S: cantidad de slots en la caché.

En este ejemplo, C_S =128 y N_S = N_B módulo 128.

La función mapeado se implementa utilizando los 16 bits de dirección de memoria (2^{16}=64 kB):

1. Los 3 bits menos significativos sirven para identificar un byte determinado dentro del bloque de memoria principal. Los restantes 13 bits especifican uno de los 2^{13} = 8K bloques de memoria principal.
2. El campo de 7 bits designado como slot da un número del bloque módulo 128. Por tanto, los bloques 0, 128, 256, ..., 8064 se mapean en el slot 0; los bloques 1, 129, ..., 8065 en el slot 1, y así sucesivamente.
3. Los 6 bits finales, el campo tag, sirve para identificar el bloque en un slot, ya que no existen dos bloques en el mismo slot que tengan el mismo número de tag. El tag es el patrón de acceso asociativo, y permite comprobar simultáneamente si el bloque que contiene la palabra buscada está presente en un slot. El acceso posterior a una palabra concreta es aleatorio.

Una operación de lectura actuaría de la siguiente manera. La caché tiene 16 bits de direcciones. El número de slot, de 7 bits, se utiliza como índice para acceder a un slot particular. Si el número de tag (de 6 bits) coincide con el número de tag para ese slot, entonces el número de palabra (de 3 bits) se utiliza para seleccionar uno de los 8 bytes que hay en el slot. En otro caso, el campo tag-slot (de 13 bits) se utiliza para buscar el bloque en memoria principal.

La técnica de mapeado directo es simple y fácil de implementar. Su principal desventaja es la existencia de una posición fija para cada bloque. Si un programa realiza repetidas referencias a dos bloques que se mapean en el mismo slot, éstos estarían continuamente conmutándose en la caché, por lo que la 'hit ratio' disminuiría.

El mapeado asociativo intenta solventar estos problemas. La dirección de memoria principal tiene un tag de 13 bits y un número de palabra de 3 bits. Un bloque de memoria puede cargarse en cualquier slot, y su tag se almacena con él. Para determinar cuándo un bloque está en la caché, se necesita una lógica que examine simultáneamente cada tag del slot para encontrarlo.

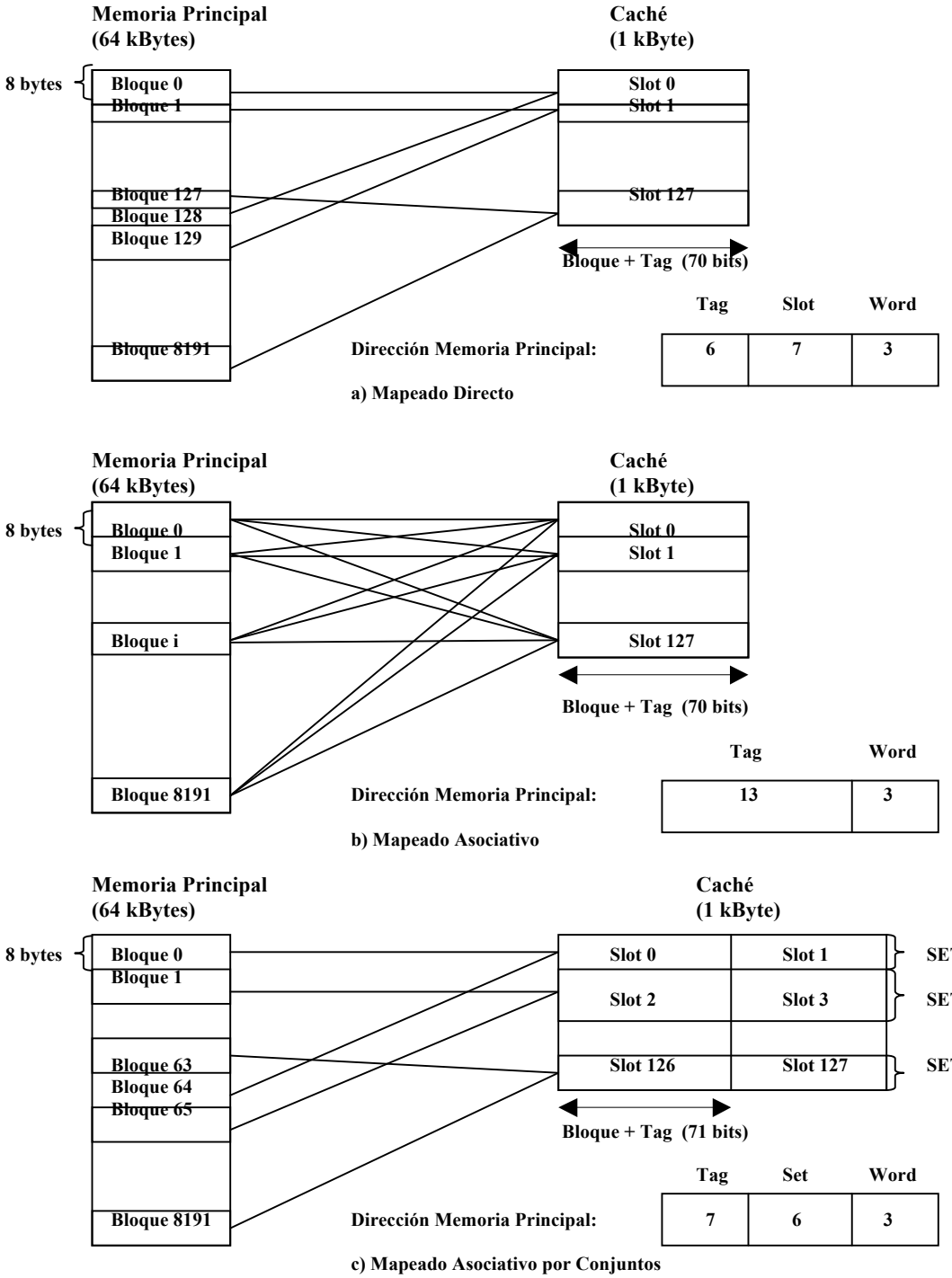

Figura 2.8. Mapeado caché/memoria principal.

Con este mapeado, hay flexibilidad respecto a qué bloque reemplazar con uno nuevo. Los algoritmos de reemplazamiento se diseñan para maximizar la 'hit ratio'. La principal desventaja es la complejidad de la circuitería necesaria.

Los conjuntos asociativos son una solución híbrida de las dos técnicas descritas. En este caso, se divide la caché en C_C conjuntos, cada uno de los cuales tiene C_S slots. Tendremos:

Número de conjunto: $N_C = N_B$ módulo C_C

donde N_C es el número de conjunto de la caché. Con este algoritmo, el bloque de dirección N_B puede mapearse en cualquier slot del conjunto C_C. En la figura, se tiene un mapeado por conjunto asociativo de dos vías, es decir, dos bloques por slot. En el caso límite en que C_S =1, esta técnica se reduce al mapeado directo, y para C_C =1 se reduce al mapeado asociativo.

El uso de dos slots por conjunto, es la organización de conjuntos asociativos más común.

ALGORITMOS DE REEMPLAZAMIENTO.

Para mapeado directo, no se necesita decidir en qué slot situar el nuevo bloque, ya que las posiciones están prefijadas. Para los otros tipos de mapeado, se han desarrollado distintos algoritmos.

Quizá el más efectivo sea el LRU (Menos Usado Recientemente): reemplaza el bloque que ha estado en la caché más tiempo sin ser referenciado. Para un conjunto asociativo de dos vías, es fácil de implementar. Cada slot incluye 1 bit de uso. Cuando se referencia un slot, el bit de uso se pone a '1', y el del otro slot del conjunto se pone a '0'. Cuando se va a colocar un bloque en el conjunto, el slot cuyo bit de uso está a cero es sustituido.

Otro tipo es el FIFO (primero en entrar, primero en salir): reemplaza el bloque del conjunto que ha estado más tiempo en la caché. Se puede implementar fácilmente mediante técnicas de buffer circular.

El LFU (Menos Frecuentemente Usado): reemplaza el bloque que ha sido referenciado menos veces. Para implementarlo, se asocia un contador a cada slot.

Otra técnica es la aleatoria: se elige un candidato a sustituir de forma aleatoria entre todos los slots. Estudios de simulación han demostrado que esta técnica proporciona prestaciones sólo ligeramente inferiores a las técnicas basadas en el uso.

MÉTODO DE ACTUALIZACIÓN.

Antes de reemplazar un bloque, es necesario considerar si ha sido alterado en la caché pero no en la memoria principal. Si no lo ha sido, puede sobreescribirse con el nuevo bloque. Si lo ha sido, la memoria principal debe actualizarse previamente. Hay dos problemas relacionados con esto:

- Más de un dispositivo puede tener acceso a memoria principal, por ejemplo un módulo E/S puede leer/escribir directamente en memoria. Si una palabra se ha alterado durante su estancia en la caché, y no se ha producido ninguna otra operación, entonces la correspondiente palabra en memoria principal no es válida.
- Si un dispositivo ha accedido a la memoria principal (escrito), esa palabra de la caché tampoco es válida. Un problema más complejo se da cuando varias CPU están conectadas al mismo bus y cada CPU tiene su caché local.

La técnica de asegurar la validez de los datos en caché y memoria principal se denomina “consistencia de la caché”. Los métodos de actualización más usuales son: escritura inmediata (write through) y post-escritura (write back).

- Escritura inmediata implica duplicar siempre las escrituras en la propia caché en la memoria principal, asegurándose así que ésta es siempre válida. Las otras CPU monitorizan el bus para mantener la consistencia de sus propias caché. Esta técnica es efectiva, pero implica un gran número de escrituras en memoria.

- Post-escritura minimiza las escrituras en memoria. Con esta técnica, las actualizaciones se realizan en la caché. Cuando esto ocurre, un bit de actualización asociado con el slot es puesto a '1'. Entonces, cuando el bloque es reemplazado, se reescribe en memoria principal si y solo si el bit de actualización está a '1'. El problema de esta técnica es que las porciones de memoria principal invalidadas, y a las que pueden acceder otros procesadores, sólo son accesibles vía la caché, por lo que la circuitería se complica y puede resultar un 'cuello de botella' potencial.

Estudios realizados sobre el porcentaje de referencias modificadas en la caché indican que sólo ocurre en un 15% de los casos, por lo que el criterio más simple de escritura inmediata se utiliza más comúnmente que otros.

TAMAÑO DEL BLOQUE.

Un bloque pequeño minimiza el tiempo de transmisión entre memoria y caché, y suele contener menos información no necesaria. Un bloque grande necesita pocos slots de caché, lo que reduce los bits necesarios para identificarlos y simplifica la circuitería. La relación entre ambos criterios es compleja y no está delimitado el valor óptimo.