Memoria

Rafael Ignacio Zurita *Rodolfo del Castillo*
rafa@fi.uncoma.edu.ar *rdc@fi.uncoma.edu.ar*

Apunte de cátedra [a]

Resumen

La organización de la memoria de una computadora que permita conseguir el rendimiento de una memoria de gran velocidad al coste de una memoria de baja velocidad se denomina jerarquía de memoria. La memoria caché, parte de esta jerarquía, funciona como interfaz entre los requisitos de la CPU veloz y la memoria principal lenta. Finalmente, se presenta el mecanismo de memoria virtual, el cual posibilita de manera transparente utilizar el espacio de direcciones completo de la arquitectura, con protección de accesos no permitidos entre programas en un sistema multiprogramado/multiproceso.

Para cualquier arquitectura de computadoras existen, por lo general, muchas implementaciones con diferentes precios y rendimiento. Comprender las diferentes tecnologías de memoria disponible permite seleccionar (y utilizar eficientemente) una organización de memoria que mejor satisfaga los requerimientos de rendimiento y confiabilidad, y que, a su vez, cumpla con restricciones de costos.

2 Memoria

memoria

La memoria de la computadora es un repositorio para las intrucciones y para los datos.

Está compuesta de dispositivos que retienen dos estados distinguibles (que se pueden especificar y sensar), y se utilizan para representar los valores cero y uno. Por lo tanto, cada uno de estos dispositivos (flip-flop semiconductor, núcleo magnético, o una moneda) puede representar un digito binario, o bit.

Los bits en la memoria son organizados como un arreglo de unidades de información, donde cada unidad de información está compuesta de un número fijo de bits y reside en una ubicación unica en memoria. Por lo tanto, cada unidad de información en memoria está caracterizada por tres cosas:

- Su tamaño, que es el mismo para todas las unidades;
- Su dirección, la cual es su ubicación relativa en memoria; y
- Su contenido, que es el valor numérico que está fisicamente almacenado en esa ubicación particular en memoria.

En las computadoras modernas el elemento de información direccionable mas pequeño es siempre el byte (8 bits). Esto significa que cada unidad de información

[a] Obra derivada con permiso escrito (detalles de las obras originales y permisos en la última sección) de articulos de Alan Clements, para las materias de arquitecturas de computadoras, de la universidad de Teesside, Inglaterra; y de Computer Programming and Architecture the VAX-11, Henry Levy, Digital Press (1980).

almacenado en la memoria principal es un byte, y cada byte tiene una dirección unica. Es importante recordar la diferencia entre la dirección de un byte en memoria, y su contenido (el valor numérico almacenado físicamente en esa dirección).

ACLARACIÓN: Muchas veces se presenta la confusión entre si el elemento más pequeño direccionable en la memoria es el byte o es el bit. Esto sucede porque existen dos niveles (al menos) para exponer el concepto de *elemento direccionable en memoria*, y muchas veces la bibliografía al respecto no aclara el contexto. Un nivel de memoria es la *memoria de la computadora* y el otro nivel es la *memoria semiconductora* (que es con lo que se construye la memoria de la computadora). La memoria semiconductora está compuesta de *celdas de memoria*, y es importante distinguir este concepto de la unidad de información de la memoria una computadora.

La celda de una memoria semiconductora es un circuito electrónico que almacena un bit de información, y cada celda en la memoria semiconductora tiene una dirección única (es decir, cada bit almacenado tiene una dirección). Los fabricantes de memorias semiconductras especifican la manera de direcccionar y referenciar cada celda (bit), para que pueda ser leído (sensado) o escrito (definido). Algunas veces, en cambio, los fabricantes organizan el circuito integrado de memoria semiconductora de manera tal que cada dirección acceda a varios bits de almacenamiento (en lugar de un bit). Por ejemplo, una especificación podria indicar una organización de memoria de 256K palabras x 8 bits, o 512K palabras x 4 bits.

Las memorias semiconductoras son los elementos con los cuales se construye la memoria de una computadora. Por ejemplo, observe que cada DIMM DRAM de PC suele tener varios circuitos integrados, cada uno siendo una memoria semiconductora. Otro ejemplo observable son los sistemas embebidos, en donde suele existir un unico circuito integrado de memoria semiconductora en la placa, o como parte del sistema en el chip (System On Chip). En la organización de la memoria en una computadora moderna el elemento direccionable mas pequeño es el byte (8 bits), ya que es la unidad de información minima utilizada como tipo de dato básico por motivos historicos (aún si por ejemplo la memoria semiconductora que se encuentra en un nivel mas bajo tenga al bit como elemento de información).

Asi que ***RECUERDE:***

- El elemento más pequeño direccionable en la memoria debe ir acompañado delcontexto.
- El elemento más pequeño direccionable en la memoria de una computadora es el byte.
- El elemento direccionable más pequeño en una memoria semiconductora es usualmente el bit.

Las arquitecturas de computadoras presentan tambien el concepto de palabra (word), que define a la unidad con la cual opera la CPU (es decir, los operandos que puede direccionar y utilizar como entrada en los calculos aritméticos y lógicos). La

palabra (word) en maquinas modernas es de 32 y 64 bits de tamaño (por ejemplo, en intel x86, ARM y MIPS).

En estas arquitecturas el bus de direcciones y datos está organizado por palabra[b]. Esto significa que la CPU lee (o escribe) una palabra completa desde la memoria, en una unica instruccion de lectura (o escritura). Recuerde, sin embargo, que la unidad en las memorias es el byte. Entonces, por ejemplo, al realizar una lectura en una arquitectura de 32bits (palabra de 32bits), cada uno de los cuatro bytes de la palabra leída tiene una direccion diferente (cuatro direcciones consecutivas). Generalmente las arquitecturas permiten tambien (a nivel de programación) leer (o escribir) medias palabras o un byte.

El espacio de direcciones es el conjunto de todas las direcciones, o el número de unidades de información distintas que un programa puede utilizar (referenciar). El tamaño del espacio de direcciones es determinado, generalmente, por el número de bits del bus de direcciones, utilizados para referenciar una dirección. Una dirección es usualmente menor o igual que el tamaño de la palabra de la arquitectura. Una computadora intel x86 de 64bits tiene un espacio de direcciones a memoria de 2^64 o 18446744073709551616 direcciones únicas.

3 Jerarquía de memoria

El tiempo de acceso de las memorias (tiempo que demora leer un dato desde una ubicación en memoria) tiene una influencia sustancial en el rendimiento de una computadora. Al momento de diseñar la organización de memoria de un nuevo sistema (CPU) se tienen en cuenta diferentes tecnologías disponibles, ya que por ejemplo, el costo de la memoria ha sido el mayor factor en el precio general del sistema en las últimas décadas. Aunque las memorias se hayan convertido tambien en componentes baratos existen tecnologías de memorias diferentes en cuanto a rendimiento, capacidad de almacenamiento y costos.

Hay muchos tipos de mecanismo de almacenamiento, cada uno con sus propias características. Comenzamos enumerando algunos de los parámetros fundamentales de los sistemas de memoria:

Celda de memoria Una celda de memoria es la unidad de almacenamiento de información más pequeña y contiene unicamente un 0 o 1. Las celdas de memoria se agrupan a menudo en bytes (y palabras). La ubicación de cada byte en la memoria se especifica por su dirección, que se denomina dirección física, para distinguirla de la dirección lógica o virtual generada por el procesador.

Capacidad La capacidad de una memoria se expresa como el número de bits o bytes que puede contener. Los dispositivos semiconductores se especifican

[b] En algunas arquitecturas no siempre el ancho de la palabra coincide con el ancho del bus del sistema. Por ejemplo, la palabra en el microprocesador Z80 (1976) era de 8 bits, aunque su bus de direcciones fue de 16 bits.

normalmente en bits (por ejemplo, un chip DRAM de 256 Mbit), mientras que los CDs y discos se especifican en términos de bytes (por ejemplo, un CD de 600 Mbytes o un disco rígido de 2000 Gbytes). Algunos fabricantes utilizan la convención de que 1K = 1.000 y 1M = 1.000.000. Utilizaremos la convención normal de que 1K = 2^10 = 1.024 y 1 M = 2^20 = 1.048.576.

Densidad La densidad de un sistema de memoria es una medida de cuántos datos se pueden almacenar en un cierto tamaño (por unidad de área o por unidad de volumen); que es densidad = capacidad / tamaño.

Tiempo de acceso El parámetro más importante de una memoria es su tiempo de acceso, que es el tiempo que se tarda en leer datos desde una ubicación en memoria, y se mide desde el inicio del ciclo de lectura. El tiempo de acceso se compone de dos partes: el tiempo necesario para localizar el byte de memoria requerido dentro de la matriz de memoria y el tiempo necesario para que los datos estén disponibles para la CPU o dispositivo que generó la solicitud. Hablando estrictamente, debemos referirnos al tiempo de acceso del ciclo de lectura y al tiempo de acceso del ciclo de escritura. Debido a que muchas memorias de semiconductores tienen tiempos de acceso de lectura y escritura casi idénticos, consideramos el tiempo de acceso como el tiempo de acceso de lectura o escritura. Esto no es correcto para todos los tipos de memoria, ya que algunos dispositivos tienen tiempos de acceso de lectura y escritura diferentes. Algunas memorias también se especifican en términos de tiempo de ciclo, que es el tiempo que debe transcurrir entre dos accesos de lectura o escritura sucesivos. El tiempo de acceso y los tiempos de ciclo son a menudo idénticos. Sin embargo, la afirmación anterior no es cierta para las memorias dinámicas de semiconductores y las EPROMs flash.

Acceso aleatorio Cuando la memoria está organizada de tal manera que el tiempo de acceso de cualquier celda es constante y es independiente de la ubicación actual (del último acceso) se denomina memoria de acceso aleatorio (RAM). Es decir, el tiempo de acceso de la memoria de acceso aleatorio no depende de dónde se encuentren los datos a los que se accede. La CPU no debe preocuparse por el tiempo que se demora en leer una palabra desde la memoria porque todos los ciclos de lectura tienen la misma duración. Es lamentable que el término ROM se utilice muchas veces como opuesto al concepto RAM. Esto se debe a que se piensa generalmente que el termino RAM significa que la memoria es de lectura y escritura. Pero esto es incorrecto ya que una memoria de acceso aleatorio sólo indica la propiedad de tiempo de acceso constante, y no tiene nada que ver con la capacidad de la memoria para modificar sus datos (es decir, escribir).

Acceso en serie En una memoria de acceso en serie el tiempo necesario para acceder a los datos depende de la ubicación física de los mismos dentro de la memoria. Algunos ejemplos de memorias de acceso en serie son las cintas magnéticas, discos rígidos, y unidades de CD o DVD. El acceso en serie también se denomina acceso secuencial. Es fácil ver por qué las memorias de acceso en serie tienen tiempos de acceso variables. Si los datos se escriben en una cinta magnética, el tiempo necesario para leer los datos es el tiempo

que tarda la posición de la cinta que contiene los datos en desplazarse hasta la cabeza de lectura.

Ancho de banda El ancho de banda de un sistema de memoria indica la velocidad a la que se pueden transferir datos entre la memoria y el procesador. Se mide en bytes por segundo. El ancho de banda está determinado por el tiempo de acceso de la memoria, el tipo de bus e interfaz entre la memoria y la CPU. Por ejemplo, un disco rígido puede tener un ancho de banda de 40 Mbyte por segundo; lo cual significa que se pueden transferir 40 Mbytes entre el disco y la CPU en un segundo.

Latencia El ancho de banda indica que tan rápido se transfieren los datos una vez que la memoria tiene los datos listos para transferir. La latencia es el tiempo de demora que existe entre el inicio de un acceso a la memoria y el inicio de la transferencia. Cuando el término es aplicado a buses, entonces la latencia es el tiempo que toma obtener el control del bus antes de que se pueda iniciar un ciclo de lectura o escritura.

Memoria volátil La memoria volátil pierde sus datos cuando se quita la fuente de energía. La mayoría de las memorias semiconductoras en las que los datos se almacenan como carga en un capacitor o como estado de un transistor (encendido o apagado) en un circuito biestable son volátiles. Algunos dispositivos semiconductores Algunos dispositivos semiconductores como las memorias EPROM y las memorias flash no son volátiles y conservan los datos cuando la alimentación está apagada. Las memorias basadas en el magnetismo son generalmente no volátiles porque su estado magnético no depende de un suministro continuo de energía.

Memoria de sólo lectura El contenido de una memoria de sólo lectura (ROM) puede leerse pero no modificarse (en condiciones normales de funcionamiento). Las memorias de sólo lectura verdaderas son, por definición, no volátiles. La memoria ROM se utilizan frecuentemente para almacenar sistemas operativos, y también para software de sistema en sistemas con microprocesadores pequeños (por ejemplo, palmtops agendas personales).

Memoria dinámica Las memorias dinámicas (DRAM) almacenan la información en forma de una carga electrónica sobre la capacitancia entre electrodos de un transistor de campo efecto. Debido a que este capacitor no es perfecto, la carga gradualmente se escapa, descargando el capacitor y perdiendo los datos. Las memorias dinámicas requieren circuitos adicionales para restaurar la carga de los capacitores periódicamente (cada 2-16 ms) en una operación conocida como refresco de memoria (memory refreshing). Las DRAM son mucho más baratas que las memorias estáticas de la misma capacidad.

Memoria estática Una vez que los datos han sido escritos en una memoria estática (SRAM) permanecen sin modificación hasta que se alteren por sobreescritura con nuevos datos, o mediante el corte del suministro de energía si la memoria es volátil (a diferencia de las memorias dinámicas que deben ser continuamente "refrescadas").

La memoria principal de una computadora típica (por ejemplo, una PC) varía en tamaño, desde 1GB de capacidad a varios Gigas; y tiene un tiempo de acceso de aproximadamente 50 ns. Incluso esta memoria se divide a veces en una memoria caché de alta velocidad y una memoria principal más lenta. El programa que inicia la ejecución de la computadora se denomina firmware UEFI (o antiguamente BIOS) y se almacena en una memoria de sólo lectura de semiconductores (llamada memoria flash). Finalmente, el PC suele tener tambien uno o más discos rígidos, unidades de DVD y CD, y lectores de memorias SD y/o microSD. Si todos estos dispositivos almacenan datos, ¿por qué necesitamos tantos?. En una computadora ideal quisiéramos tener una memoria con las siguientes características: veloz (tiempo de acceso bajo), tamaño reducido (físico), bajo consumo energético, robusta (que su contenido no cambie por errores), y de bajo precio.

Si el costo no es una restricción, el sistema de memoria entero podría ser construido utilizando la memoria disponible mas rápida, como se hizo para la computadora CRAY-1 (1975), y como podría hacerse en las supercomputadoras, donde el costo no es un problema. [c] Para computadoras de bajo costo, esto es obviamente imposible.

jerarquía de memoria

De cualquier manera, una opción para muchas computadoras tipicas es organizar el sistema de memoria dentro de una jerarquía de niveles, donde cada nivel está compuesto de tecnologías de memorias diferentes (en cuanto a costo, capacidad y velocidad de acceso). El objetivo es estructurar diferentes tipos de memoria en un mismo sistema, para conseguir el rendimiento de una memoria de gran velocidad al coste de una memoria de baja velocidad. Esta organización se denomina ***jerarquía de memoria*** de una computadora, y puede ser llevado a cabo gracias al conocimiento estadístico de los patrones de acceso a memoria que realizan los programas en ejecución.

La Figura 1 ilustra la jerarquía de memoria utilizada en muchas computadoras.

[c] Esta afirmación es unicamente válida en cuanto a costos, y para algun sistema y aplicación muy específica. Actualmente los sistemas operativos y las aplicaciones para supercomputadoras son grandes y complejas, por lo que se implementan otras estrategias diferentes (como ccNUMA).

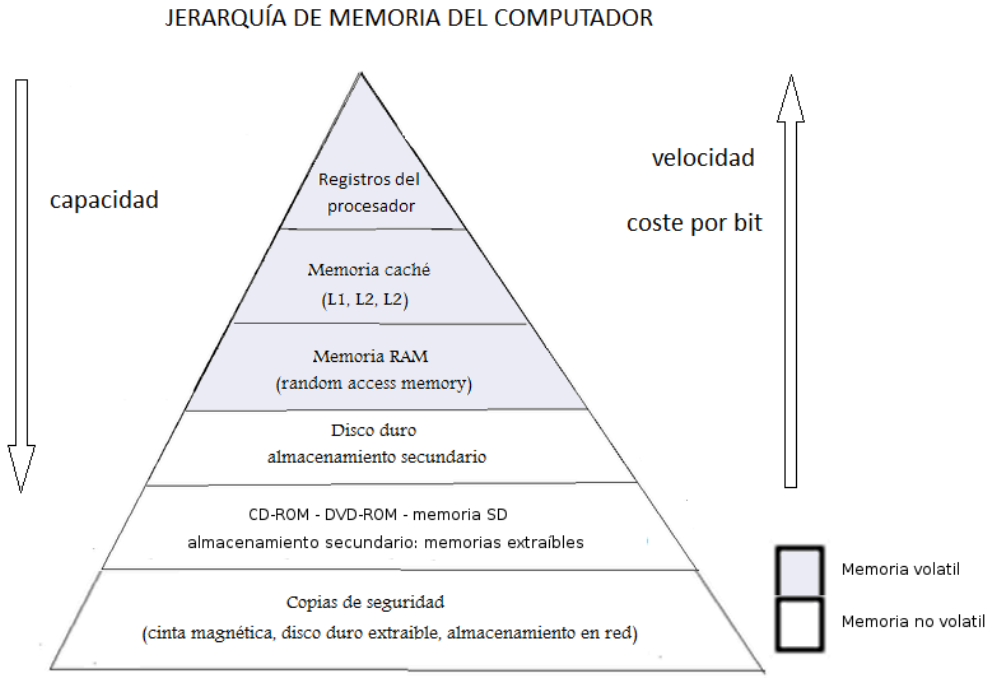

Figure 1: Jerarquía de memoria.

Los dispositivos en el tope de la jerarquía son caros, rápidos y de poca capacidad. Los dispositivos en la base de la jerarquía son baratos y almacenan una gran cantidad de datos, pero son extremadamente lentos. El diagrama no es muy exacto de todas maneras, ya que por ejemplo, el CD-ROM o DVD-ROM tienen capacidades de entre 600 Mbytes y 4 Gbytes y (desde el punto de vista de la capacidad) deberían aparecer por encima de los discos rígidos en la figura.

En la punta de la jerarquía de memoria de la figura 1 tenemos la memoria interna de la CPUs: los registros. Los registros en las CPUs tienen tiempos de acceso muy bajos y están construidos con la misma tecnología que la CPU misma. Son muy caros (en términos de recursos de silicio que ocupan) limitando el número de registros internos y scrachpad memory que pueden haber dentro de la CPU.

La memoria principal (RAM) contiene los programas en ejecución, y sus datos. Esta memoria es relativamente rápida (de 10 ns a 70 ns), y se implementa invariablemente como memoria estática o dinámica de semiconductores. Históricamente, hasta los años 70, las memorias de núcleo de ferrita y las de alambre cromado (creada por los Laboratorios Bell en 1957) se encontraban en las principales tiendas.

Existen dos tipos de memorias de acceso aleatorio: las memorias caché y la memoria principal. La memoria caché contiene copias de los datos utilizados con frecuencia, y está construída típicamente de tecnología SRAM (Static RAM). La celda básica de una memoria SRAM está construída de un flip-flop D latch, y cuatro compuertas (gates) en un diseño discreto, y de cuatro a seis transistores. La memoria principal está construída, comunmente, de tecnología DRAM (Dynamic RAM), en la cual cada celda básica almacena información en un pequeño capacitor accedido a través de un transistor MOS. (para más información al respecto ver Digital Design: Principios y Práctica. 5ta edición. John Wakerly).

El disco rígido es de tecnología magnética, y almacena grandes cantidades de datos en poco espacio, a un costo muy bajo por bit. Desafortunadamente, acceder a los datos de una pista en particular es un proceso en serie y el tiempo de acceso de un disco, aunque rápido desde nuestra perspectiva, es de magnitudes lentas en comparación con la memoria principal. Una unidad de disco típica puede almacenar desde varios Gbytes de datos a Terabytes, y proximamente a Petabytes, y tiene un tiempo de acceso de 8 ms. Históricamente, a finales de la década de 1990 se produjo un crecimiento explosivo en la tecnología de discos y aparecieron discos rígidos de bajo costo con capacidades mucho mayores a los CD-ROMs y DVD-ROMs.

El CD-ROM fue desarrollado por la industria de la música para almacenar sonido en discos de plástico delgado llamados CD (discos compactos). A diferencia de los discos rígidos, los CD-ROM utilizan medios intercambiables. La tecnología de CD-ROM utiliza un rayo láser para leer pequeños puntos incrustados en una capa dentro del disco. Los CDs son muy baratos y almacenan hasta unos 600 Mbytes, pero tienen tiempos de acceso más largos que los discos rígidos convencionales. En general, el CD-ROM se utiliza para distribuir software. Las unidades que permiten grabar CDs o DVDs son comunes hoy en dia, pero en sus orígenes eran muy caras en comparación con las unidades de sólo lectura. Los medios magnéticos requieren usualmente que la cabeza de lectura y escritura esté muy cerca de la superficie del medio. En cambio el CD y DVD utiliza tecnología óptica y el cabezal de lectura no entra en contacto con la superficie. Esto es lo que posibilita que el CD y DVD sea un buen medio intercambiable.

La cinta magnética es un medio de acceso en serie extremadamente barato y puede almacenar varios terabytes en una cinta que cuesta pocos dólares. Desafortunadamente, el tiempo de acceso promedio de las unidades de cinta es muy largo en comparación con otras tecnologías de almacenamiento y, por lo tanto, se utiliza principalmente para fines de archivo. Los CDs grabables han reemplazado a las cintas en varias aplicaciones, pero por ejemplo en datacenters, se utilizan las cintas en grandes equipos que realizan backups.

Mediante la combinación de todos estos tipos de memoria en un único sistema el ingeniero informático puede obtener lo mejor de todos los mundos. Se puede construir un sistema de memoria de relativamente bajo costo con un rendimiento similar ha si se utilizara unicamente memoria RAM cara de alta velocidad. La clave para el diseño de la memoria de la computadora es tener los datos correctos en el lugar correcto en el momento adecuado. Un gran sistema informático puede tener miles de programas y millones de archivos de datos. Afortunadamente, la CPU necesita unos pocos programas y archivos en un mismo momento. . Si el sistema operativo puede transferir los datos del disco a la memoria principal para que la CPU siempre (o casi siempre) encuentre los datos que necesita en la memoria principal, el sistema parece tener una memoria gigante de alta velocidad, y que cueste sólo una fración del costo. Este mecanismo implementado por el hardware y el sistema operativo se denomina memoria virtual. En una PC moderna, por ejemplo, la memoria podría parecer de 2 Tbytes, aunque en realidad tal vez existan simplemente 8 Gbytes de memoria principal pero 2000 Gbytes de espacio en disco. La Figura 2 presenta los distintos tipos de memoria actualmente disponibles según el tipo (aunque la lista no está completa).

4 Organización de la memoria utilizando la tecnología mas rápida

Una manera de estructurar el sistema de memoria total es implementar un segmento de la memoria con la tecnología mas rápida disponible. Ejemplificando, suponga que en un sistema existen 64Kbytes de memoria más rápida posible, en la cual el usuario puede completar con datos y código ejecutable. El usuario entonces ubicaría allí (al momento de cargar el programa) el código y los datos mas criticos, en cuanto a cantidad de accesos y tiempo de ejecución. Esto le permite predecir el rendimiento y tiempo de ejecución con exactitud. De cualquier manera, este método beneficia unicamente a sistemas de propósitos específicos (ejemplo: sistemas embebidos), pero es improbable que beneficie a un sistema de tiempo compartido (time-sharing system), en el cual la memoria física es asignada y reasignada entre los usuarios durante distintos momentos (computadoras de propósito general). Otra desventaja de esta aproximación es que el programador debe conocer la arquitectura de la memoria física para utilizarla correctamente.

5 Memoria Caché

La tecnología más común utilizada en una jerarquía de memoria de una computadora es la ***memoria caché*** (pronunciado "cash", del francés "cacher" que siginifica "ocultar/esconder"). Nota al pie de pagina: está oculta desde el punto de vista del programador, aparece como parte del espacio de memoria del sistema.

caché

La ***memoria caché*** es una memoria pequeña, costosa y de muy alta velocidad. Se encuentra ubicada junto a la CPU y mantiene las instrucciones y datos mas recientemente utilizados. Cuando un programa realiza un requisito a memoria, la CPU verifica primero si el dato está en la caché. Si está, entonces el dato es traído rapidamente sin necesidad de acceder la memoria principal, la cual es mucho mas lenta. El objetivo primordial es reducir el tiempo medio de acceso a los datos de la memoria principal. A diferencia de la organización explicada en la sección anterior (una cierta cantidad de memoria SRAM en el espacio de direcciones) la caché es transparente al programador.

Historicamente, luego de que las memorias cachés aparecieron para acortar la brecha entre la velocidad de la CPU y la memoria principal, los sistemas se hicieron mas complejos, y la diferencia de velocidad entre la caché y la memoria principal incrementó nuevamente. Aumentar el tamaño de la caché del primer nivel no fue economicamente viable en ese momento, por lo que se agregó un segundo nivel de caché, un poco mas lenta que la de primer nivel, pero mas grande. En la actualidad existen máquinas que llegan a tener tres niveles de memoria caché, y con el incremento del número de núcleos (cores) en una misma CPU los niveles de memoria caché podrían aumentar aún mas en el futuro. En la figura 2 puede observarse un diagrama sencillo de una arquitectura moderna, que utiliza tres niveles de memoria caché.

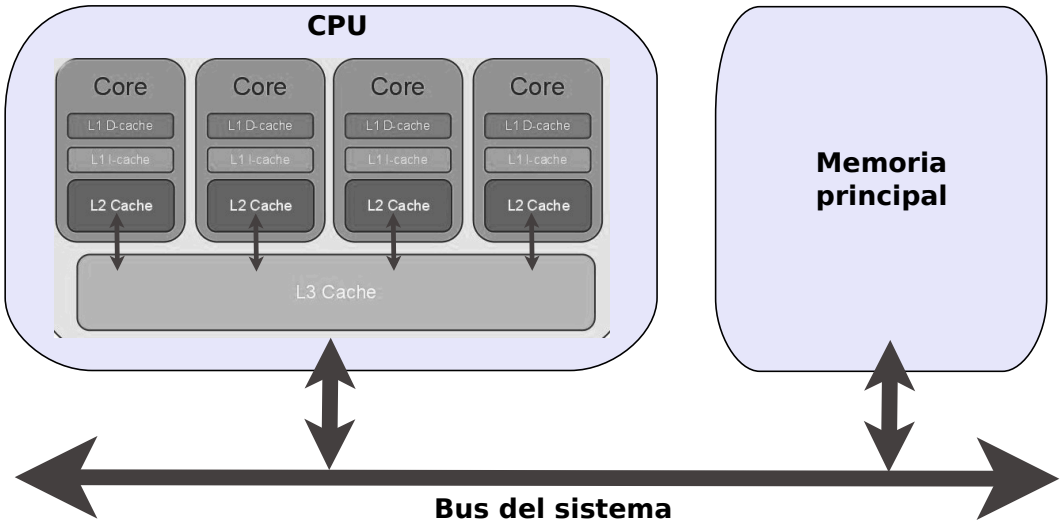

Figure 2: Esquema de una arquitectura con 4 núcleos y 3 niveles de caché.

La memoria caché se puede comprender, en términos cotidianos, por su analogía con un agenda o cuaderno utilizado para registrar números de teléfono. Una guía telefónica completa (paginas blancas) contiene cientos de miles de números de teléfono, y nadie lleva un directorio telefónico todo el tiempo. Sin embargo, la mayoría de la gente tiene una agenda o cuaderno de notas con un centenar de números de teléfono de gente conocida. Aunque la fracción de todos los números de teléfono guardados en un cuaderno de alguien podría ser inferior al 0,01% del total de la guía telefónica, la probabilidad de que su próxima llamada sea a un número del cuaderno es muy alta. ¿Por qué? Porque la gente tiende a llamar a amigos y colegas con mucha frecuencia.

Un sistema de memoria con caché opera exactamente con el mismo principio del cuaderno de notas, encontrando la información que la CPU requiere frecuentemente en la memoria caché, en vez de en la memoria principal, la cual es mucho más lenta. Desafortunadamente, a diferencia del cuaderno personal, la computadora no puede saber, a priori, qué datos serán posiblemente accedidos. Las caches de computadora funcionan según un principio de aprendizaje. Por experiencia aprenden qué datos se utilizan con más frecuencia y luego lo transfieren a la memoria caché.

La estructura general de una memoria caché se puede observar en la Figura 3. Un bloque de memoria caché se encuentra conectado a los buses de dirección y datos, en paralelo con la memoria principal mucho más grande. Tenga en cuenta que la afirmación de paralelo significa que los datos en la caché también se mantienen en la memoria principal. Para volver a la analogía con el cuaderno telefónico, escribir el número de un amigo en el cuaderno no elimina el número del directorio.

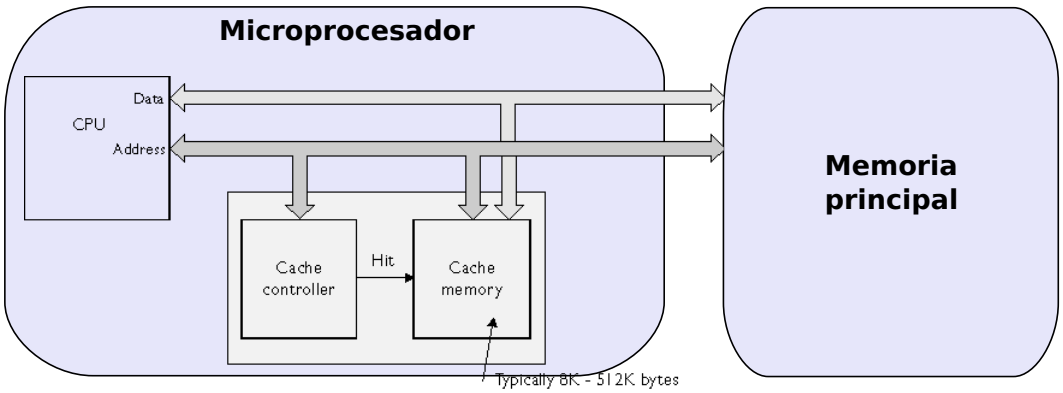

Figure 3: Diagrama de bloques de una memoria caché general.

El funcionamiento de la memoria caché se basa en el mismo principio que el cuaderno con números de teléfono, ya que el procesador no ejecuta instrucciones aleatoriamente. Debido a la naturaleza de los programas y sus estructuras de datos correspondientes, los datos requeridos por el procesador suelen estar muy agrupados en la memoria. Frecuentemente se acceden en forma secuencial (por ejemplo, cuando la CPU accede a las instrucciones del programa) o desde zonas cercanas a direcciones recientemente accedidas (cuado por ejemplo se procesa un arreglo o matrix de información).

principio de localidad

La observación del comportamiento de los programas muestra que las referencias a memoria realizadas en un intervalo corto de tiempo tienden a utilizar una pequeña fracción del total de la memoria. Este comportamientno de acceso ha sido llamado ***principio de localidad de las referencias***[DENN68], o simplemente principio de localidad.

Existen al menos dos tipos básicos de localidad: localidad espacial y temporal. La localidad temporal se refiere a la reutilización de datos especificos, dentro de un tiempo relativamente corto. Cuando un programa accede a la ubicación en memoria de un dato o instrucción es bastante probable que vuelva a acceder a la misma ubicación pronto. La localidad espacial, en cambio, se refiere a la utilización de datos en ubicaciones de memoria cercanas a los elementos accedidos recientemente. Si un programa accede a un dato o instruccion en memoria es altamente probable que tambien referencie a datos o instrucciones alojados en direcciones proximas.

Este fenómeno demostrado ha dado una base estadística para el diseño de la estructura de la memoria de un computador basado en una jerarquía de memoria.

En este caso, si la caché puede mantener suficientes datos para evitar un gran número de referencias a la memoria principal, la velocidad de ejecución incrementa substancialmente, y la utilizacion de la memoria principal y del bus del sistema es también reducida beneficiosamente. El objetivo es lograr una gran capacidad de memoria (esto se consigue con memorias baratas y lentas) pero que puedan accederse con alta velocidad (a través de la utilización de pequeñas memorias cachés entre la CPU y la memoria principal).

5.1 Tiempo medio de acceso

tasa de aciertos

El parámetro principal de un sistema de caché es su ***tasa de aciertos (hit ratio)***, que define la proporción de aciertos a todos los accesos. La tasa de aciertos es determinada a través de observaciones estadísticas del funcionamiento de un sistema real, y no puede calcularse facilmente.

Además, la proporción de aciertos depende de la naturaleza real de los programas que se están ejecutando. Es posible tener algunos programas con tasas de aciertos muy altas y otros muy bajas. Afortunadamente, el efecto de la localidad de las referencias provoca que la proporción de aciertos es, generalmente, del 98%. Antes de calcular el efecto de una memoria caché en el rendimiento de un procesador, necesitamos introducir algunos términos.

Si un dato es leído o escrito k veces en un intervalo corto de tiempo entonces la computadora necesita 1 referencia a memoria principal, y k - 1 referencias a memoria caché. Cuanto mas grande sea k mejor será el rendimiento del sistema. Es posible formalizar este cálculo utilizando el tiempo de acceso a caché, que llamaremos c, y el tiempo de acceso a memoria principal, llamado aquí m. h es la tasa de aciertos, la cual es la fracción de las referencias que están disponibles en caché.

h = (k -1)/k. Con estas variables y definiciones el tiempo medio de acceso = c + (1 - h) m

Si h tiende a 1 entonces todas las referencias pueden ser resueltas por la caché y el tiempo de acceso se aproxima a c. Por otro lado, si h tiende a 0, cada referencia debe ser satisfecha por la memoria principal y el tiempo de acceso se aproxima a c + m (ya que para cada acceso primero se intenta resolver la referencia en la caché, pero al haber un fallo, se accede a memoria principal). Una estrategia de mejora para este peor caso es realizar el acceso a memoria principal en paralelo con la referencia a cache', pero esto requiere de mecanismos para descartar o detener la lectura a memoria si ocurre un cache' hit.

En la práctica la situación real no es tan simple como las ecuaciones sugieren. Las computadoras son dispositivos sincronizados por relojes, y funcionan a la velocidad determinada por el reloj. Consecuentemente, los accesos a memoria operan en uno o más ciclos de reloj. Suponga hipotéticamente que un procesador accede a memoria principal en uno o dos ciclos de reloj (un ciclo de reloj con suficiente tiempo para acceder a la memoria principal lenta). Entonces, agregar una memoria caché no hará al sistema más rápido.

Para comprender la importancia de la memoria caché en las computadoras modernas puede observar los valores presentes en la Tabla 1, la cual presenta el tiempo de acceso a cada nivel de la jerarquía de memoria de un procesador de PC Intel Core i7 (arquitectura X86). En esta clase de procesadores cada core puede ejecutar una instruccion cada ~2 nanosegundos, y un acceso a memoria principal toma alrededor de ~60 nanosegundos. Se puede comprender con estos tiempos la importancia de los diferentes niveles de caché para mejorar el rendimiento. Sin caché el procesador debería esperar demasiado tiempo a que el dato referenciado esté disponible en la CPU (tiempo en el que podría ejecutar varias instrucciones).

Tabla 1. Tiempos de acceso a memoria en una sistema moderno (Core i7 Xeon 5500 Series)

Memoria - Nivel	Nro de ciclos (aprox)	Tiempo de acceso (aprox)
L1 CACHE hit	~4 cycles	~2.0 ns
L2 CACHE hit	~10 cycles	~3.0 ns
L3 CACHE hit, line unshared	~40 cycles	~12.0 ns
L3 CACHE hit, shared line in another core	~65 cycles	~19.5 ns
L3 CACHE hit, modified in another core	~75 cycles	~22.5 ns
remote L3 CACHE	~100-300 cycles	~30 ns
DRAM	~200 cycles	~60 ns

El efecto del rendimiento de una memoria caché de una computadora depende de muchos factores, incluyendo la manera en que la caché está organizada, la forma en que los datos son escritos a memoria principal (desde la caché), y la manera en que los programas acceden a los datos en memoria. Si la caché puede almacenar suficienten datos para evitar un gran numero de referencias a memoria principal (lenta) entonces no sólo la velocidad de ejecución incrementa, sino que tambien se reduce el uso del bus del sistema y memoria.

> La memoria caché puede mejorar el rendimiento de una computadora dramaticamente, a cambio de un costo adicional relativamente bajo.

5.2 Organizacion interna de la memoria caché

La estructura interna y el funcionamiento de una memoria caché describen dónde debe colocarse un bloque de memoria principal cuando se almacena en la caché.

Hay al menos tres maneras de organizar una memoria caché: caché de mapeo directo (direct-mapped cache), caché asociativa (associative cache), y caché asociativa por conjunto de n-vias (n-vias set associative cache).

5.2.1 Caché de mapeo directo

La forma más sencilla de organizar una memoria caché es utilizar un mapeo directo, que se basa en un algoritmo simple: asignar el bloque de datos i de la memoria principal al bloque de datos i en la memoria caché.

línea de caché

La figura 4 ilustra la estructura de una memoria caché simple de mapeo directo. La memoria está compuesta de 32 palabras y se accede mediante un bus de direcciones de 5 bits que lo conecta a la CPU y caché. El espacio de memoria ha sido dividido en conjuntos y los conjuntos en líneas.

La caché contiene "entradas" llamadas comunmente ***líneas de caché (cache line)***. En este ejemplo particular cada línea de caché mantiene un bloque datos pequeño,

formado por dos palabras consecutivas.

Cada dirección es de 5 bits, y está compuesta por un campo de 2 bits para indicar el conjunto, un campo de 2 bits para indicar la línea de caché, y un campo de selección de palabra de 1 bit. La memoria caché tiene 4 líneas de dos palabras cada una. Cuando el procesador genera una dirección, se accede a la línea apropiada en la caché. Por ejemplo, si el procesador genera la dirección de 5 bits 01010, se accede a la línea 2 del conjunto 2.

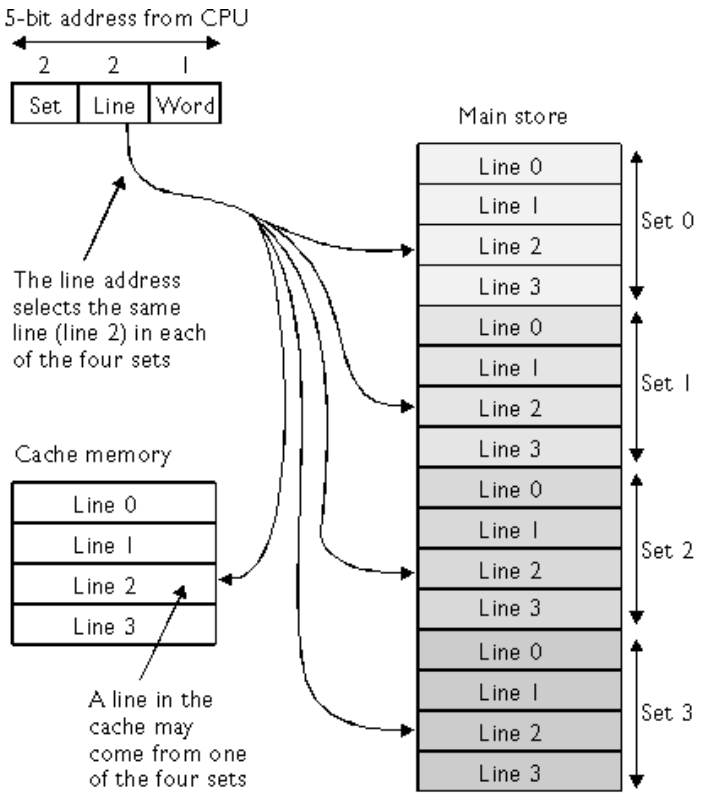

Figure 4: La memoria caché simple de mapeo directo.

Un vistazo a la figura 4 revela que hay cuatro líneas número dos posibles: una línea 2 en el conjunto 0, una línea 2 en el conjunto 1, una línea 2 en el conjunto 2 y una línea 2 en el conjunto 3. En este ejemplo, el procesador accedió a la línea 2 en el conjunto 2. La pregunta obvia es, '¿cómo sabe el sistema si la línea 2 accedida en la caché es la línea 2 del conjunto 2 en la memoria principal?'

acierto de caché (hit cache)

La Figura 5 muestra que existe tambien una ***etiqueta (tag)*** asociada a cada línea en la memoria caché, que determina a qué conjunto pertenece esa línea. De esta manera es posible confirmar que la línea en la caché pertenece o no al conjunto especificado en la direcciión originada por el procesador. Cuando el procesador accede a la línea 2, la etiqueta que pertenece a la línea 2 de la memoria caché se envía a un comparador. Al mismo tiempo, el campo del conjunto de la dirección originada por el procesador también se envía al comparador. Si son iguales, la línea en la caché es la línea deseada y se produce un ***acierto (hit)***.

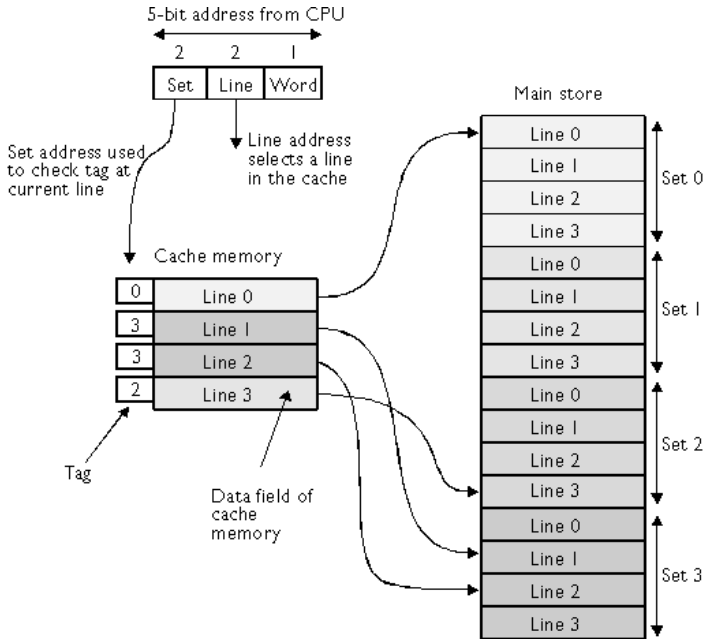

Figure 5: Resolución del conjunto de cada línea en una caché de mapeo directo.

fallo de caché (cache miss)

Si las etiquetas (conjuntos) no son los iguales, se produce un ***fallo (miss)*** y se debe actualizar la memoria caché. La antigua línea 2 del conjunto 1 es descartada o reescrita de nuevo a la memoria principal, dependiendo de cómo está organizada la actualización de la memoria principal.

señal de acierto (hit signal)

La figura 6 muestra la estructura de un sistema de memoria caché de mapeo directo. La caché es simplemente un bloque de memoria de lectura / escritura de acceso aleatorio de muy alta velocidad. Tambien incorpora un comparador y una memoria rápida para las etiquetas. El bus de direcciones contiene la dirección de entrada a la caché el cuál es utilizada para acceder a una única ubicación de línea de caché. La etiqueta en esa línea es entonces comparada con el valor del conjunto presente como parte de la dirección de entrada proveniente del bus de direcciones. Si la etiqueta concuerda con el número de conjunto de la dirección la ***señal de acierto (hit signal)*** es activada.

Como muestra la Figura 6, la memoria de etiquetas de la caché no es más que una simple memoria de acceso aleatorio de alta velocidad con un comparador de datos incorporado. Algunos de los principales fabricantes de semiconductores han implementado esta memoria de etiquetas de caché de un solo chip.

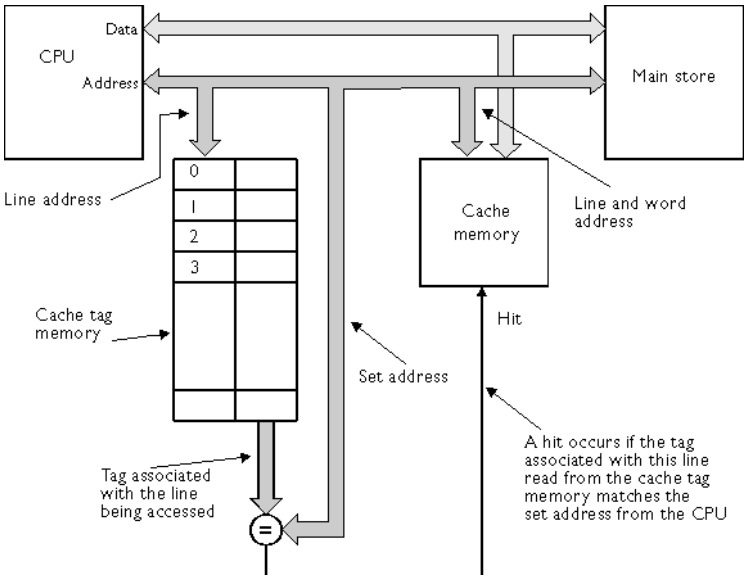

Figure 6: Implementación de caché de mapeo directo.

La ventaja de la memoria caché de mapeo directo es casi evidente. Debido a su sencillez es veloz y no presenta complejidad extra (costo de construcción bajo). Esta caché es un dispositivo ampliamente disponible que, aparte de su velocidad, no son más complejos que cualquier otro circuito integrado.

Esta caché tampoco requiere ningún algoritmo de sustitución de línea. Si se accede a la línea x en el conjunto y se produce un fallo de caché, la línea x del conjunto y se carga de la memoria principal y se almacena en la línea x de la memoria caché. Por lo tanto, no hay mecanismos de decisión que deba seleccionar cuál línea de caché debe ser reemplazada cada vez que una nueva línea es cargada desde la memoria principal.

Otra ventaja importante es su paralelismo inherente. Debido a que la caché y la memoria de etiquetas son independientes, ambas se pueden acceder simultáneamente. Una vez que la etiqueta ha coincidido y se ha producido un acierto, los datos presentados por la caché también serán válidos (suponiendo que la caché y la memoria de etiquetas tengan tiempos de acceso aproximadamente iguales).

La desventaja de esta caché es casi un corolario de su ventaja. Un caché con n líneas tiene una restricción: en cualquier instante puede contener sólo una línea numerada x. Por lo tanto, no puede mantener una línea x del conjunto p y una línea x del conjunto q. Esta restricción existe porque hay un único bloque de datos en la caché para cada una de las lineas posibles. Para ejemplificar, observe el siguiente fragmento de código que llama a dos subrutinas en un bucle:

```
REPETIR
LLAMAR Get_data
LLAMAR Compare
HASTA coincidencia o fin_de_los_datos
```

Suponga que en la versión compilada de este código parte de la subrutina

Get_data está en el conjunto x, línea y; y que parte de la subrutina Compare está en el conjunto z, línea y. Debido a que una caché de mapeo directo puede contener sólo una línea y a la vez, la trama correspondiente a la línea y debe recargarse dos veces para cada conjunto a través del bucle. El rendimiento de la caché para este caso es muy bajo (fallos de caché frecuentes).

Supongamos ahora que una memoria caché de mapeo directo está casi vacía, y que la mayoría de sus líneas aún no se han cargado con datos. Sin embargo, las únicas líneas con datos válidos deben reemplazarse con frecuencia debido a que los accesos a memoria son direcciones con diferentes números de conjuntos pero misma línea. Este caso también tendrá un rendimiento pobre (fallos de caché frecuentes) aunque haya muchas líneas aún vacías.

A pesar de los ejemplos anteriores, mediciones estadísticas de programas reales indican que el rendimiento bajo del peor caso no tiene un impacto significativo en su rendimiento general. Además, para ciertos casos, puede trabajar mejor que una caché totalmente asociativa, como veremos luego.

La caché de mapeo directo es muy popular debido a su bajo costo de implementación y alta velocidad.

CUADRO: de mejor caso que asociativa

5.2.2 Caché asociativa

caché asociativa

Una excelente forma de organizar una memoria caché que no presenta las limitaciones de la caché de mapeo directo se describe en la Figura 7, la cual se denomina memoria caché asociativa. Este tipo de caché no tiene restricciones en cuanto a qué datos puede contener cada entrada. En otras palabras, cada entrada en esta caché puede almacenar cualquier bloque (línea) de la memoria principal.

Una dirección del procesador se divide en tres campos: la etiqueta, la línea y la palabra. Al igual que la memoria caché de mapeo directo, la unidad más pequeña de datos transferido dentro y fuera de la caché es la línea. A diferencia de la caché de mapeo directo, no existe una relación entre la ubicación de las líneas en la memoria principal con respecto a la ubicación de las líneas en la caché. La línea p en memoria principal se puede almacenar en la entrada q de la caché, sin restricciones sobre los valores de p y q.

Como ejemplo, considere un sistema con 1 Mbyte de memoria principal y 64Kbytes de caché asociativa. Si el tamaño de una línea es de cuatro palabras de 32 bits (es decir, 16 bytes), la memoria principal está compuesta de 220/16 = 64K líneas y la caché está compuesta de 216/16 = 4096 líneas. Debido a que una caché asociativa permite que cada una de sus entradas se cargue con cualquier línea de la memoria principal, cada entrada puede contener cualquier de las 64K líneas posibles de la memoria principal. Para identificar de manera unívoca a qué línea de la memoria principal pertenecen las palabras almacenadas en cada entrada de la caché se utiliza, en este caso, una etiqueta de 16bits asociada a cada entrada.

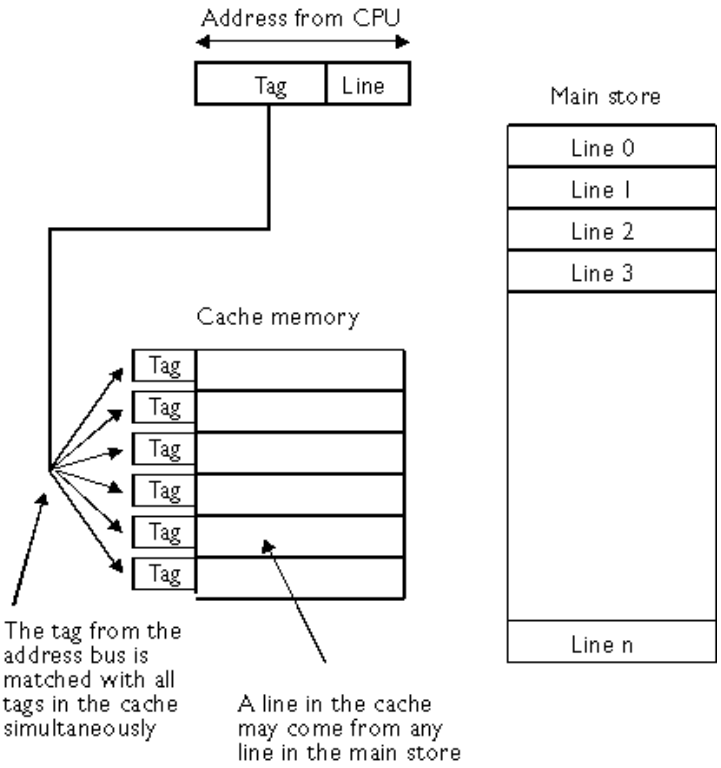

Figure 7: Caché asociativa.

Cuando el procesador genera una dirección, los bits de la misma seleccionan una ubicación tanto en la memoria principal como en la memoria caché. Los bits que identifican la línea en la dirección no se pueden utilizar para direccionar una entrada en la memoria caché (a diferencia de la memoria caché de mapeo directa). Como la caché asociativa puede almacenar cualquier línea de la memoria de 64K en una de sus entradas, requiere una etiqueta de 16 bits (suponiendo que hay 216 líneas posibles en la memoria principal) para cada una de sus entradas. Más importante aún, ya que las entradas de la caché (es decir, líneas) no están ordenadas, las etiquetas no están ordenadas y no se pueden almacenar en una tabla de búsqueda simple como la memoria caché de mapeo directo. En otras palabras, cuando la CPU requiere acceder a la línea i, el dato puede estar en cualquier entrada de la caché o puede no estar en la caché.

La caché asociativa emplea un tipo especial de memoria llamada memoria asociativa. Una memoria asociativa tiene una entrada de n bits (proveniente del bus de direccion), pero no necesariamente 2n ubicaciones internas únicas. La entrada de dirección de n bits es una etiqueta que es comparada con la etiqueta de cada una de sus entradas simultáneamente.

Si la etiqueta de entrada coincide con una etiqueta almacenada en la caché, los datos asociados con esa ubicación se presentan como salida. De lo contrario, la caché asociativa produce un ***fallo de caché (caché miss)***. Una memoria asociativa no es direccionada de la misma manera que la memoria principal. La memoria principal de la computadora requiere la dirección explícita de una localización, mientras que una memoria asociativa es accedida preguntando, "¿tiene este elemento almacenado en alguna parte?"

Las memorias caché asociativas son eficientes porque la etiqueta que especifica

la línea que se quiere obtener se compara simultáneamente con la etiqueta de cada entrada de la memoria caché. En otras palabras, se accede a todas las ubicaciones al mismo tiempo. Desafortunadamente las memorias asociativas, aunque pequeñas, son muy costosas, y cachés asociativas grandes no existen. Además, una vez que la caché asociativa está llena, sólo se puede introducir una nueva línea sobrescribiendo una existente. Por lo tanto, las cachés asociativas deben utilizar alguna política de sustitución para la entrada (línea) siendo descartada.

caché asociativa por conjuntos

Las memorias caché totalmente asociativas no son prácticas, debido a su alto costo. Sin embargo, la mayoría de las computadoras emplean un tipo de caché organizada de maneria mixta, utilizando algunos mecanismos de las cachés de mapeo directo y algunos de las cachés totalmente asociativas. Este sistema combinado se llama ***caché asociativa por conjuntos de n vías***.

Una caché asociativa por conjuntos está compuesta varias cachés de mapeo directo operados en paralelo. La disposición más simple se denomina ***caché asociativa por conjunto de 2 vías*** y consta de dos memorias caché de mapeo directo. Cada línea en el sistema de caché está duplicado; Por ejemplo, hay dos líneas números 5 en la memoria caché. En consecuencia, ahora es posible almacenar dos líneas número 5, una línea 5 del conjunto x y una línea 5 del conjunto y.

La Figura 8 es una descripción de una ***caché asociativa por conjunto de 4 vías***, la cuás es muy utilizada. Cuando el procesador presenta una dirección de memoria a ser accedida la línea apropiada en cada una de las cuatro cachés de mapeo directo es accedida simultáneamente. Debido a que hay cuatro líneas, se utiliza un comparador sensicillo para determinar cuál (si presente) de las líneas en las cachés es la apropiada para recuperar los datos. En la figura 8, la salida de acierto (hit) de cada caché de mapeo directo es entrada a una puerta OR el cual genera un acierto (hit) si alguna de las cachés generó un acierto.

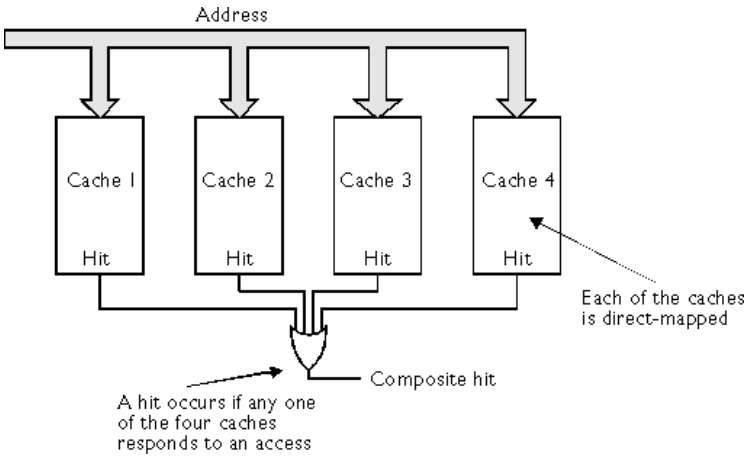

Figure 8: Caché asociativa por conjuntos de 4 vías.

5.3 Estrategias en ciclos de escritura

Además de seleccionar una organización y política de reemplazo para el sistema de caché el diseñador tiene que considerar cómo se deben tratar los ciclos de escritura.

escritura diferida (write-back)

Una tecnica común es modificar el dato en la entrada de la caché, sin modificar la

memoria principal. Este método es llamado estrategia de ***escritura diferida (write-back)***, y reduce el número de escrituras a memoria principal en muchas escrituras consecutivas de la misma entrada. Un bit extra para cada entrada está presente en la caché, y se utiliza para indicar si la línea en la caché fue modificada. El hardware es más complejo en este caso, ya que se debe verificar este bit para conocer cuándo se debe escribir a memoria principal. Cuando una entrada de la caché debe ser reemplazada debido a un fallo de caché (cache miss) el bit de modificación es verificado. Si los datos en la línea a ser reemplazada difiere de los datos en memoria principal (esto es, el bit indica que se encuentra modificada), un ciclo de escritura es realizado primero desde la caché a memoria principal, para luego poder sobreescribir esta línea de la caché con la nueva transferencia de datos desde la memoria principal.

escritura directa (write-through)

Otra estrategia mas sencilla es la ***escritura directa (write-through)***. Cuando se realiza un requisito de escritura el nuevo dato es escrito en ambas memorias, en la caché y en la memoria principal. De esta manera, tanto la caché como la memoria principal tienen siempre copias validas de todos los datos. Aunque existe un rendimiento mas bajo que la estrategia anterior, muchas veces el procesador puede continuar ejecutando instrucciones mientras la escritura a memoria principal procede.

5.4 Otras consideraciones en el diseño de cachés

Cuando se produce un fallo de caché, se obtiene una línea completa de datos desde la memoria principal. Si el procesador necesita leer un unico byte y, debido fallo de caché, se requiere leer una línea de 8 bytes, podemos imaginar que el costo de un fallo en un acceso a caché conlleva una penalización adicional. Afortunadamente, las tecnologías de memorias modernas y CPUs soportan un modo de completado llamado ***burst-fill mode***, que puede transferir un bloque/ráfaga (busrt) de elementos de datos consecutivos entre la memoria principal y la memoria caché.

coherencia de la caché

Otro aspecto de las memorias caché que debe tenerse en cuenta en los sistemas sofisticados es la ***coherencia de la caché***. Como sabemos, los datos en el caché son copias de los datos en memoria principal. Cuando el procesador modifica los datos, debe modificar tanto la copia en el caché como la copia en memoria principal (aunque no necesariamente al mismo tiempo). Hay circunstancias en las que la existencia de dos copias (que pueden ser diferentes) del mismo elemento de datos causan problemas. Por ejemplo, un controlador de E/S que utiliza DMA podría intentar mover una línea antigua de datos de la memoria principal al disco, sin conocer que el procesador acaba de actualizar la copia de los datos en el caché (y que aún no ha actualizado en memoria principal). La coherencia del caché también se conoce como consistencia de datos. En sistemas con varios procesadores es posible tener varias copias del mismo dato. Por ejemplo, una linea de datos de memoria principal podrías estar siendo utilizada por dos o más procesadores diferentes, los cuales cuentan con una cache individual cada uno. Si uno de los procesadores modifica una de las lineas de su caché, algún mecanismo debe estar presente para mantener la coherencia de las cachés.

bit de validez (valid bit)

La organización de la memoria caché suele utilizar un ***bit extra de validez (valid bit)*** para cada entrada en la caché. El bit de validez es utilizado para verificar si la línea presente es válida o no. Cuando un procesador comienza a utilizar la caché (y por lo tanto la caché está vacía), todos los bits de validez están desactivados (por ejemplo,

todos en cero), para indicar que todas las entradas son invalidas. A medida que se comienzan a transferir datos de memoria principal a la caché, el bit de validez de cada línea de la caché se activa, para indicar que los datos son válidos. Cada vez que la caché verifica si la dirección de entrada está o no en la caché, tambien verifica el bit de validez.

En algunas situaciones, varias líneas de la caché (o todas) deben ser invalidadida. Por ejemplo, en una operación de E/S. Supongamos por un momento que un controlador DMA está modificando lineas de memoria principal con datos provenientes de un dispositivo de entrada. Si existen copias en la caché de estos datos, el bit de validez puede ser utilizado para "invalidar" esa línea en la caché. Otro ejemplo común es el cambio de contexto de un proceso (si la computadora cuenta con un sistema operativo, o multiproceso). Previo a que otro proceso esté activo en la CPU, los bit de validez pueden utilizarse para invalidar toda la caché, ya que todas las lineas presentes pertenecen al proceso activo anterior.

Múltiples memorias caché en paralelo (arquitectura Harvard) Aunque la mayoría de las computadoras en las últimas décadas son de arquitectura von Neumann la experiencia ha demostrado que es de gran ventaja separar las memorias caché utilizadas para instrucciones y para los datos. Intel ha utilizado memorias caché separadas para las intrcucciones y para los datos desde 1993, y nunca ha vuelto atrás. Las regiones de memoria necesarias para las instrucciones y los datos son generalmente independientes, por lo que tener cachés paralelas e independientes mejora aún mas el rendimiento (la CPU puede leer una instrucción al mismo tiempo que lee o escribe una dirección de memoria relacionada con los datos del programa).

POR HACER:

Politicas de reemplazo

Emprolijar la terminología.

Agregar los cuadros de notas importantes (obtenidos de los distintos libros)

6 Licencia, obras originales, permisos y bibliografía

Licencia de uso

Obras originales y permisos

Este apunte es un trabajo derivado (con permiso escrito) de las siguientes obras (ordenadas de la mas utilizada a la menos utilizada):

- Apuntes de cátedra del Profesor Alan Clements http://www.scm.tees.ac.uk/users/

a.clements/ Lamentablemente el sitio no está ya mas disponible, pero puede ser alcanzado utilizando http://www.archive.org.
- Libro "Computer Programming and Architecture the VAX-11", Henry Levy, Digital Press, 1980.

Alan Clements fue profesor de las materias "Sistemas de computadoras", "Organización de computadoras" y "Arquitectura de computadoras", en la Universidad de Teesside, Inglaterra (actualmente está retirado). Es también el autor de los siguientes libros:

-Microprocessor Systems Design: 68000 Family Hardware, Software and Interfacing. ISBN 978-0534948221. 1997
-Computer Organization & Architecture : Themes and Variations. ISBN 978-1111987046, 2012.

Permiso escrito:

```
From: Alan Clements
Date: Wed, 5 Jul 2017 16:48:40 +0100
Message-ID:
Subject: Re: About permission of notes and articles
To: Rafael Ignacio Zurita

--94eb2c072bc01bcd8c055393eff3
Content-Type: text/plain; charset="UTF-8"
Content-Transfer-Encoding: quoted-printable

Hola Rafael,

Thank you for writing to me.

The academic address was at Teesside university. I have now retired and,
sadly, can't use that address any more.

This is my main address and am perfectly happy for you to write to me at
this address.

Por supuesto, you can use my material from the web and translate it into
Spanish. I would be delighted for you to translate it into Spanish.

If there is anything I can do to help, please let me know.

[...]

Best wishes

Alan
```

Hank Levy trabajó en los años 70 y 80 en la arquitectura de computadora VAX, en Digital Equipment Corporation (DEC). Actualmente es profesor e investigador en la Universidad de Washington (https://www.cs.washington.edu/people/faculty/levy), y

es el autor del libro utilizado. .

Permiso escrito:

```
From: Hank Levy
Date: Sun, 14 May 2017 20:34:40 -0700
Message-ID:
Subject: RE: About rights of the Computer Programming and Architecture 2nd Edition The
Vax book
To: Rafael Ignacio Zurita
Content-Type: text/plain; charset="UTF-8"


Hi Rafael,

Wow -- that's very nice. The book is no longer in print and I don't have
any problem with you using it however you want. So....I hereby give
you permission to translate parts of the book to use for your class for
students.

Best of luck!

hank
```

Bibliografía extra

- Artículo "What Every Programmer Should Know About Memory", Ulrich Drepper, Red Hat, Inc, drepper@redhat.com, 2007.

7 Indice