UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

CENTRO UNIVERSITARIO UAEM TEXCOCO

ANALISIS DOCUMENTAL SOBRE MEMORIAS CACHE

TESINA

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTA:

ALMA DELIA NAPOLES SOLIS

DIRECTOR DE TESINA

DR. JOEL AYALA DE LA VEGA

REVISORES

M en I.S.C. IRENE AGUILAR JUÁREZ.

M. en C. LETICIA ARÉVALO CEDILLO.

TEXCOCO, MÉXICO NOVIEMBRE 2013.

Texcoco, México a 23 de Septiembre del 2013.

M. EN C. JUAN MANUEL MUÑOZ ARAUJO
SUBDIRECTOR ACADEMICO DEL
CENTRO UNIVERSITARIO UAEM TEXCOCO
PRESENTE:

AT'N M.EN P.P. ANTONIO INOUE CERVANTES
RESPONSIBLE DEL DEPARTAMENTO DE TITULACION

Con base en las revisiones efectuadas al trabajo escrito titulado "Análisis Documental sobre Memorias Caché" que para obtener el título de Licenciado en Ingeniería en Computación presenta la sustentante Alma Delia Nápoles Solis, con número de cuenta 0621522 respectivamente, se concluye que cumple con los requisitos teórico-metodológicos por lo que se le otorga el voto aprobatorio para su sustentación, pudiendo continuar con la etapa de digitalización del trabajo escrito.

A T E N T A M E N T E

REVISORA
M. en I.S.C. Irene Aguilar Juárez

REVISORA
M. en C. Leticia Arévalo Cedillo.

DR. JOEL AYALA DE LA VEGA
DIRECTOR DEL TRABAJO ESCRITO

c.c.p. ALMA DELIA NAPOLES SOLIS.
c.c.p. DR. JOEL AYALA DE LA VEGA.
c.c.p. M.EN.P.P. ANTONIO INOUE CERVANTES.

TITULACION
RECIBIDO
Por
Texcoco, Méx., a 23 de 9 del 2013

RESUMEN.

La memoria es uno de los componentes fundamentales de las computadoras, sin ella no tendrían un medio de almacenamiento temporal para la ejecución de programas. La memoria es el medio de almacenamiento temporal en el que la CPU (Microprocesador) puede escribir, leer o modificar información.

Las memorias de las computadoras presentan tal vez la más amplia diversidad de tipos, tecnología, estructura, prestaciones y costo, de entre todos los componentes de una computadora. Una computadora convencional está equipada con una jerarquía de subsistemas de memoria, algunos internos (directamente accesibles por el procesador), y otros externos (accesibles por el procesador mediante módulos de entrada/salida). (Stallings, México2007).

Ante la inmensa velocidad de los procesadores que a medida del tiempo se va incrementando, el límite es mayor entre la transferencia de información de la memoria principal (RAM) y el CPU; ante esto se plantearon soluciones, una incrementar la velocidad de la RAM y otra, quizá la más óptima, agregar un nuevo componente al PC: la memoria caché. (Bacalla, noviembre 2009).

Actualmente la informática a nivel usuario ha dejado de lado el uso de arquitecturas monoprocesadoras (salvo en los PCs y en los llamados Notebooks), pasando a utilizar arquitecturas con chips de al menos 2 núcleos.
Este cambio producido en los últimos años en el mundo PC ya se dio en la supercomputación hace más de dos décadas con el fin de alcanzar los Grandes Retos" de la ciencia. (Gargollo & Lorenzo, Mayo 2011).

Por lo anteriormente mencionado en esta tesina se presenta el estudio del elemento esencial de cualquier computadora moderna: **la memoria caché**.

Platicaremos de las técnicas para la combinación de una memoria pequeña y rápida combinada con una memoria grande y lenta, la memoria pequeña y rápida que se ha mencionado se le conoce como memoria caché (del francés cacher, que significa guardar).

AGRADECIMIENTOS.

A mi hermosa Universidad Autónoma del Estado de México la cual llevo en el corazón siempre, que me dio todo y abrió sus puertas del conocimiento que con mucho orgullo, amor, pasión y respeto representaré.

Por haberme dado cobijo y por las lecciones que aprendí en ella, asimismo, por haberme dado su voto de confianza y por todo el apoyo otorgado a mi persona.

A mi director de esta tesina al Dr. Joel Ayala de la Vega por motivarme a investigar y a dar mucho más de mí misma; y por mostrarme que el mundo es mucho más grande e interesante de lo que uno piensa y vale la pena conocerlo.

Gracias por haberme brindado su sabiduría, tiempo, paciencia, tolerancia, consejos y apoyo para poder concluir con este proyecto.

A mis revisoras; la M. en I.S.C. Irene Aguilar Juárez y a la M en C. Leticia Arévalo Cedillo por el tiempo que dedicaron para la asesoría y revisión del desarrollo de este trabajo

DEDICATORIA.

A mis padres.

Con todo mi amor y agradecimiento quiero dedicar este logro a ustedes por que representan el amor que siempre me impulso a realizar este sueño; siendo ustedes la inspiración para tomar nuevos retos en la vida.

Gracias por su paciencia y apoyo incondicional que en todo momento me brindaron. Para ustedes con todo mi amor

Finalmente:

Dedico a todas las personas que tuve la oportunidad de conocer en el Centro Universitario UAEM Texcoco; a mis compañeros, amigos y profesores; con quienes he tenido la oportunidad de compartir mis conocimientos y me han ayudado a reforzar y adquirir nuevas experiencias.

TABLA DE CONTENIDO.

ÍNDICE DE FIGURAS.

ÍNDICE DE TABLAS.

CAPÍTULO 1.- INTRODUCCIÓN.

En esta tesina se hace un análisis documental sobre memorias caché, ya que es un elemento esencial de cualquier computadora moderna y conoceremos las técnicas, las arquitecturas, su clasificación por niveles.

1.1 Antecedentes.

Las memorias caché empezaron a aparecer a principios de los años setenta, hoy en día su uso está muy extendido. (Cabello & Toledano, Octubre 1997). Una memoria caché es una memoria pequeña, especial y muy rápida (del francés cacher, que significa guardar o esconder) situada entre el procesador y la memoria principal, diseñada para acelerar el procesamiento de instrucciones del microprocesador, el cual contiene temporalmente la información que está siendo utilizada en un momento determinado. (S. Tanenbaum, 2000).

Desde las primeras generaciones de procesadores la latencia en los accesos a la memoria principal es la causa del bajo desempeño en la ejecución de las aplicaciones, ya que esta no responde a los pedidos de instrucciones y datos a la velocidad que le es necesaria al procesador.

Este factor se vuelve crítico en las arquitecturas multinúcleo o Chip Multi-Processors (CMP) actuales ya que se incrementa aún más esta diferencia de velocidad al tener múltiples núcleos de procesamiento realizando accesos en forma simultánea. (Carballal, Mayo2011).

La caché se utiliza de diversas formas para reducir el tiempo efectivo que requiere el procesador para acceder a las instrucciones o datos almacenados en la memoria principal. A veces, la caché se utiliza para almacenar instrucciones únicamente. En este caso, se aprovecha una ventaja que presentan las instrucciones frente a los datos, las instrucciones no se modifican.

El contenido de una caché de instrucciones no requiere ser escrito en memoria principal. Se puede ver el sistema jerárquico de tres niveles como dos sistemas independientes con dos niveles en jerarquía. El sistema de memoria principal-secundaria se puede organizar como un sistema de memoria virtual.

El sistema de memoria caché-principal se organiza de forma similar; este sistema presenta muchas de las propiedades de un sistema de memoria virtual.

Existen técnicas que permiten combinar una memoria pequeña y rápida con otra grande y lenta para conseguir un sistema de memoria con la misma capacidad que la memoria grande que trabaja a (casi) la velocidad de la memoria rápida a precio moderado. (Cabello & Toledano, Octubre 1997).

1.2 Justificación.

La memoria cache, en este momento, es una parte fundamental de una computadora. No sólo existe un tipo de memoria cache, sino más bien una jerarquía de ellas. No solo existe memoria cache entre la memoria principal y la CPU sino que existen memorias cache que apoyan a otros dispositivos que involucran entrada/salida. Por lo que es fundamental que un estudiante del área de cómputo conozca los tipos de memoria cache y su arquitectura.

1.3 Objetivos.

Realizar un análisis de la memoria caché utilizando diversos medios de información para mostrar la frontera del uso de esta herramienta y que sirva como una antología.

Mostrar la clasificación de las memorias cache basados en: tipo de acceso, lugar y arquitectura

1.4 Organización de tesina.

En este capitulado se ha presentado: una pequeña introducción a cerca del trabajo realizado, antecedentes, justificación y objetivo de este trabajo.

El resto de la tesina está organizado de la siguiente manera:

En el capítulo 2 se presentan los conceptos básicos de las memorias caché, así como la terminología, su funcionamiento, su importancia y su objetivo de la memoria caché.

El capítulo 3 contiene todo lo referente a la identificación de cachés desarrollo acceso RAM y CAM.

En el capítulo 4 se presenta la clasificación de las memorias caché desarrolladas dependiendo del lugar entre la CPU y la memoria principal.

En el capítulo 5 se muestra una arquitectura de las cachés en sus diferentes contenidos multiprocesador simétrico, memoria compartida distribuida, procesadores vectoriales

En el capítulo 6 se muestra otros tipos de memoria caché como la memoria caché de disco y la memoria caché en servidores.

Finalmente, se presentan 5 apéndices; el primer apéndice habla del principio de localidad, el segundo muestra paginación, el tercero habla de fallo de página, el cuarto menciona memoria virtual, el quinto contendrá a la memoria asociativa; todo esto con el fin de dar una información más detallada de ciertos conceptos que se engloban en esta tesina, con sus respectivas referencias y bibliografía de dicho trabajo.

CAPÍTULO 2 MEMORIA CACHÉ.

En este capítulo constará de tres secciones: en la primera se presentan los conceptos básicos relacionados a la memoria caché; en la segunda se incluye la importancia y función principal de dicha memoria; en la tercera sección hablaremos acerca del porqué de la caché.

2.1 Conceptos básicos de caché.

La memoria es un elemento sencillo que contiene los programas que se ejecutan en la computadora y los datos sobre los que trabajan dichos programas, sin embargo, presenta una gran diversidad de tipos, tecnologías, estructuras, prestaciones y costos.(García & Serra, Noviembre 2010).

La memoria caché es una memoria pequeña, rápida y especial de alta velocidad, se ubica entre la CPU y la memoria principal, diseñada para acelerar el procesamiento de instrucciones del microprocesador, el cual, puede acceder a los datos almacenados en la caché mucho más rápido que a aquellos datos almacenados en la memoria RAM. (Canto Q, Febrero 2004).

Las memorias cachés empezaron a aparecer a principios de los años setenta, hoy en día su uso está muy extendido. La caché se utiliza de diversas formas para reducir el tiempo efectivo que requiere el procesador para acceder a las instrucciones o datos almacenados en la memoria principal. A veces, la caché se utiliza para almacenar instrucciones únicamente. En este caso, se aprovecha una ventaja que presentan las instrucciones frente a los datos, las instrucciones no se modifican. El contenido de una caché de instrucciones no requiere ser escrito en memoria principal. (M.A de Miguel Cabello & T.Higuera Toledano, 1996).

Lo interesante es que se conozcan las técnicas para combinar una cantidad pequeña de memoria rápida con una cantidad grande de memoria lenta para obtener la velocidad de la memoria rápida y la capacidad de la memoria grande. (S. Tanenbaum, 2000).

El diseño de la memoria de un computador se pueden resumir en tres cuestiones: ¿Cuánta capacidad? ¿De qué velocidad? ¿De qué costo?

La cuestión del tamaño es un tema siempre abierto. Si se consigue hasta una cierta capacidad probablemente se desarrollarán aplicaciones que la utilicen. La cuestión es la rapidez es, en cierto sentido, fácil de responder. En un momento dado, la eficiencia del procesador depende del acceso de datos o instrucciones a memoria.

Es decir, cuando el procesador ejecuta instrucciones, no es deseable que tenga que detenerse a la espera de instrucciones o de operandos. La última de las cuestiones anteriores también debe tenerse en cuenta. En la práctica, el costo de la memoria debe ser razonable con relación a los otros componentes.

Como es de esperar, existe un compromiso entre las tres características clave que son: costo, capacidad y tiempo de acceso.

En un momento dado, se emplean diversas tecnologías para realizar los sistemas de memoria. En todo el espectro de posibles tecnologías se cumplen las siguientes relaciones:

- A menor tiempo de acceso, mayor coste por bit
- A mayor capacidad, menor coste por bit
- A mayor capacidad, mayor tiempo de acceso

El dilema con que se enfrenta el diseñador está claro. El diseñador desearía utilizar tecnologías de memoria que proporcionen gran capacidad, tanto porque esta es necesaria como por que el costo por bit es bajo.

Sin embargo, para satisfacer las prestaciones requeridas, el diseñador necesita utilizar memorias costosas, de capacidad relativamente baja y con tiempos de acceso reducidos.

La respuesta a este dilema es no contar con un solo componente de memoria, sino emplear una **jerarquía de memoria.** La Figura 2.1.1 ilustra una jerarquía típica de Memorias.

Cuando se desciende en la jerarquía ocurre:

a) Disminuye el coste por bit
b) Aumenta la capacidad
c) Aumenta el tiempo de acceso
d) Disminuye la frecuencia de accesos a la memoria por parte del procesador

Así pues, memorias más pequeñas, más costosas y más rápidas, se complementan con otras más grandes, más económicas y más lentas. La clave del éxito de esta organización está en el último *ítem* (d): la disminución de la frecuencia de acceso.

De acuerdo con lo anterior, es posible organizar los datos a través de la jerarquía de tal manera que el porcentaje de accesos un nivel inferior sea sustancialmente menor que el nivel anterior. (Stallings, México 2007).

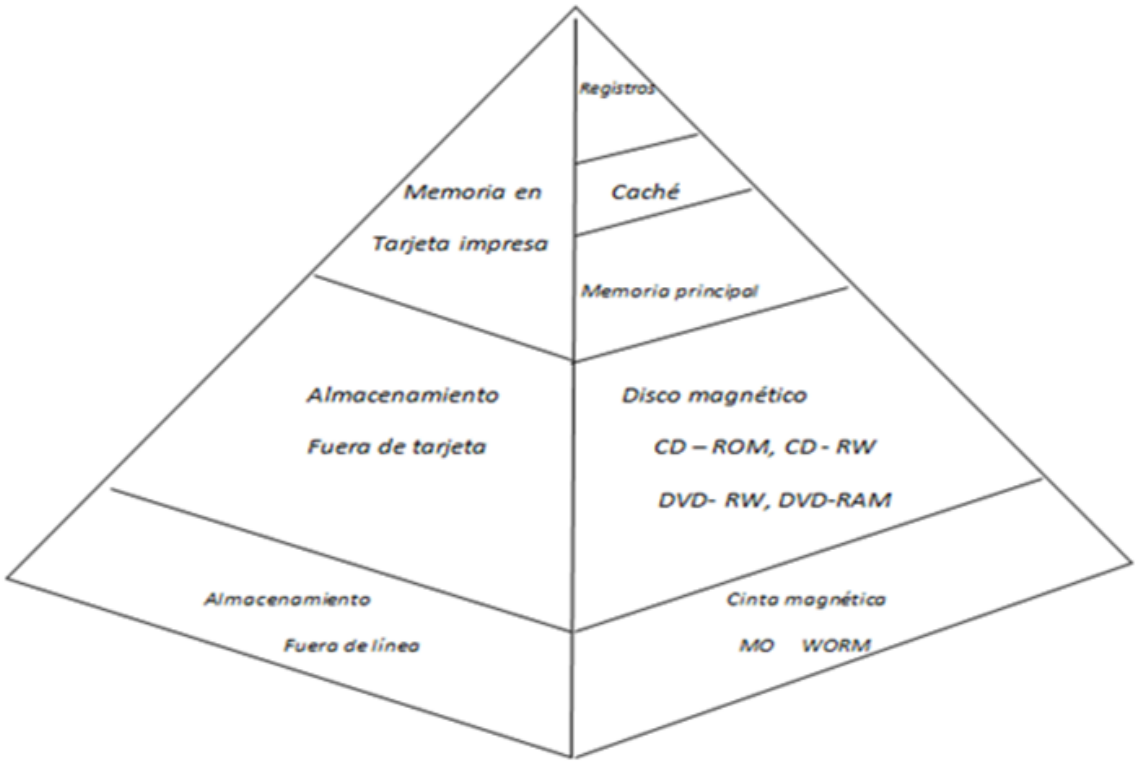


Figura 2.1.1.-. Jerarquía de Memorias.(Stallings, México 2007).

2.2 Porque de la memoria caché.

El análisis de un gran número de programas de computación ha mostrado que durante la ejecución del programa, la referencia a memoria tiende a ocurrir en patrones bien localizados. Considere por ejemplo, un bloque de memoria contiene un código continuo, entonces la ejecución se realizará en forma secuencial a través del bloque. Si el bloque representa un ciclo, la ejecución del bloque se ejecutará un número determinado de veces. Si el bloque representa un arreglo de datos, entonces será referenciada una determinada cantidad de veces.

Esta característica del programa se conoce como **principio de referencia de localidad** (Ver anexo A).

Cuando un programa es guardado en memoria principal, el tiempo de acceso en el ciclo de búsqueda **está en función del tamaño de la memoria** (entre más grande sea la memoria principal, más lento es su acceso, por lo que la velocidad de ejecución no depende de la velocidad de la CPU sino más bien de la velocidad de acceso a memoria) y no **en función del tamaño del bloque local.** Por lo que, para hacer uso del principio de localidad de referencia, puede ser usada una memoria pequeña de alta velocidad que guarde exclusivamente los bloques activos de código o datos.

El objetivo de la memoria caché es lograr que la velocidad de la memoria sea lo más rápida posible, consiguiendo al mismo tiempo un tamaño grande al precio de memorias semiconductoras menos costosas. El concepto se ilustra en la Figura 2.2.1 donde se muestra las Memorias caché y principal.

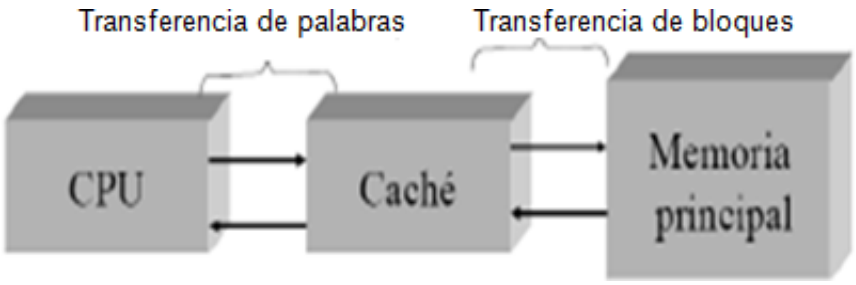


Figura 2.2.1.- Memorias caché y principal. (Stallings, México 2007).

Hay una memoria principal más grande y relativamente más lenta junto con una caché más pequeña y más rápida. Esta caché tiene una copia de partes de la memoria principal.

Cuando el procesador intenta leer una palabra de memoria, se hace una comprobación para determinar si la palabra está en la caché. Si es así, se entrega dicha palabra al procesador. Si no, un bloque de memoria principal, consistente en un cierto número de palabras, se transfiere a la caché y después la palabra es entregada al procesador. (Stallings, México 2007).

2.3 Función principal.

El funcionamiento de la memoria cache se basa en la transferencia de partes (bloques) de la memoria principal y la memoria cache y de la transferencia de palabras entre memoria cache y la CPU. (García & Serra, Noviembre 2010).

Desde el punto de vista memoria caché, una dirección consta de dos partes: la dirección del bloque y la palabra o byte dentro de él (desplazamiento).

Cuando el procesador manda una dirección a la caché, ésta compara la dirección del bloque, en paralelo, con todas las direcciones que tiene almacenadas. En esta búsqueda puede suceder:

- Que el bloque se encuentre en la caché, hay un éxito, y se selecciona la palabra deseada dentro del bloque correspondiente.

- Que el bloque no se encuentre en la memoria caché, hay un fallo. En este caso, se trae el bloque correspondiente de memoria principal y se actualiza el campo tag.
- Posiblemente sea necesario expulsar uno de los bloques que están en la caché.
- Por último, se localiza la palabra a la que se desea acceder.

En estos dos esquemas de acceder a la caché mediante direcciones reales, en cuyo caso se dice que la caché está localizada en el espacio de direcciones real.

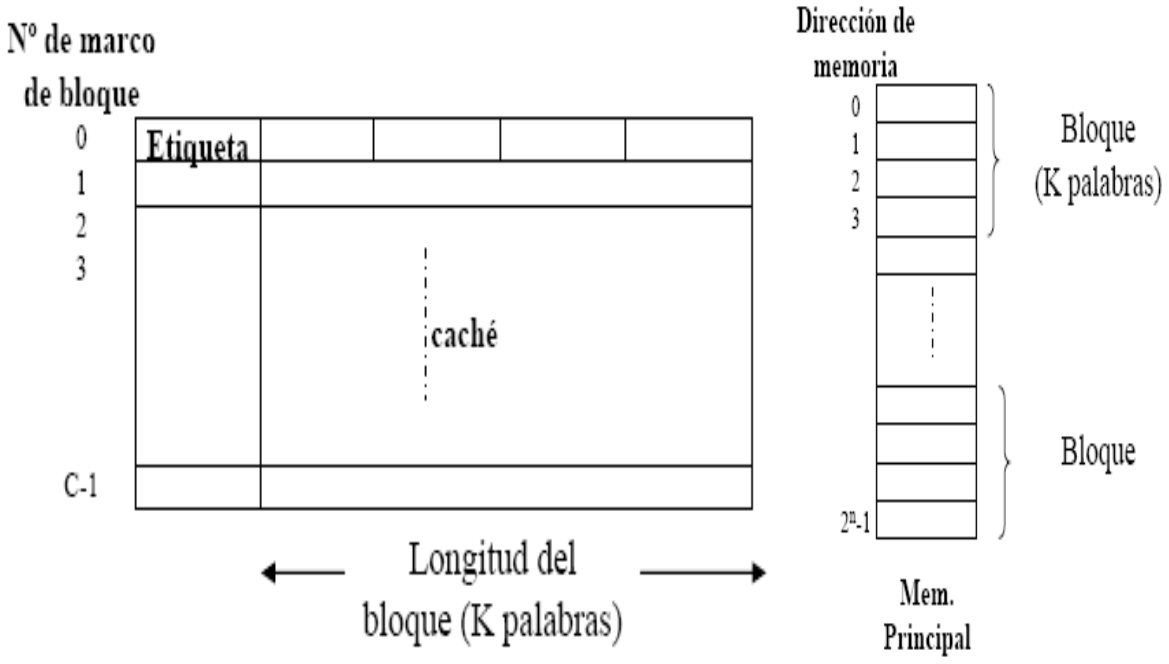
No se va a tratar el caso de memorias caché localizadas en espacio de memoria virtual, dado que habría que considerar ciertos problemas que se presentan en este tipo de implementación.

La Figura 2.3.1 describe la estructura de un sistema de memoria caché/principal. La memoria principal consta de hasta 2^n palabras direccionables, teniendo cada palabra una única dirección de de n bits. Esta memoria está organizada en M bloques de longitud fija (K palabras/bloque), siendo esta una organización por paginación. (Ver anexo B)

Donde $M=2^n/K$ bloques.

La memoria cache está dividida en C líneas o particiones de K palabras donde la memoria cache es mucho más chica que la memoria principal.

(C<<M).



En la figura 2.3.1 Estructura de memoria principal. (Stallings, México 2007).

La Figura 2.3.2. Ilustra una operación de lectura. El procesador genera la dirección, RA, de una palabra a leer. Si la palabra está en la caché, es entregada al procesador. Si no, el bloque que contiene dicha palabra se carga en la caché, y la palabra después es llevada al procesador.

La Figura2.3.2 indica como estas dos últimas operaciones se realizan en paralelo y refleja la organización mostrada en la Figura 2.3.3., que es típica en las organizaciones de caché actuales.

En ella, la caché conecta con el procesador mediante líneas de datos, de control y de direcciones. Las líneas de datos de direcciones conectan también con buffers de datos y de direcciones que las comunican con el bus del sistema a través de la cual se accede a la memoria principal.

Cuando ocurre un acierto de caché, los buffers de datos y de direcciones se inhabilitan, y la comunicación tiene lugar solo entre el procesador y caché, sin tráfico en el bus. Cuando ocurre un fallo de caché, la dirección deseada se carga en el bus del sistema y el dato es llevado, a través del buffer de datos, tanto a la caché como al procesador.

En otras formas de organización, la caché se interpone físicamente entre el procesador y la memoria principal para todas las líneas de datos, direcciones y control. En este caso frente a un fallo de caché, la palabra deseada es primero leída por la caché y después transferida desde esta al procesador.

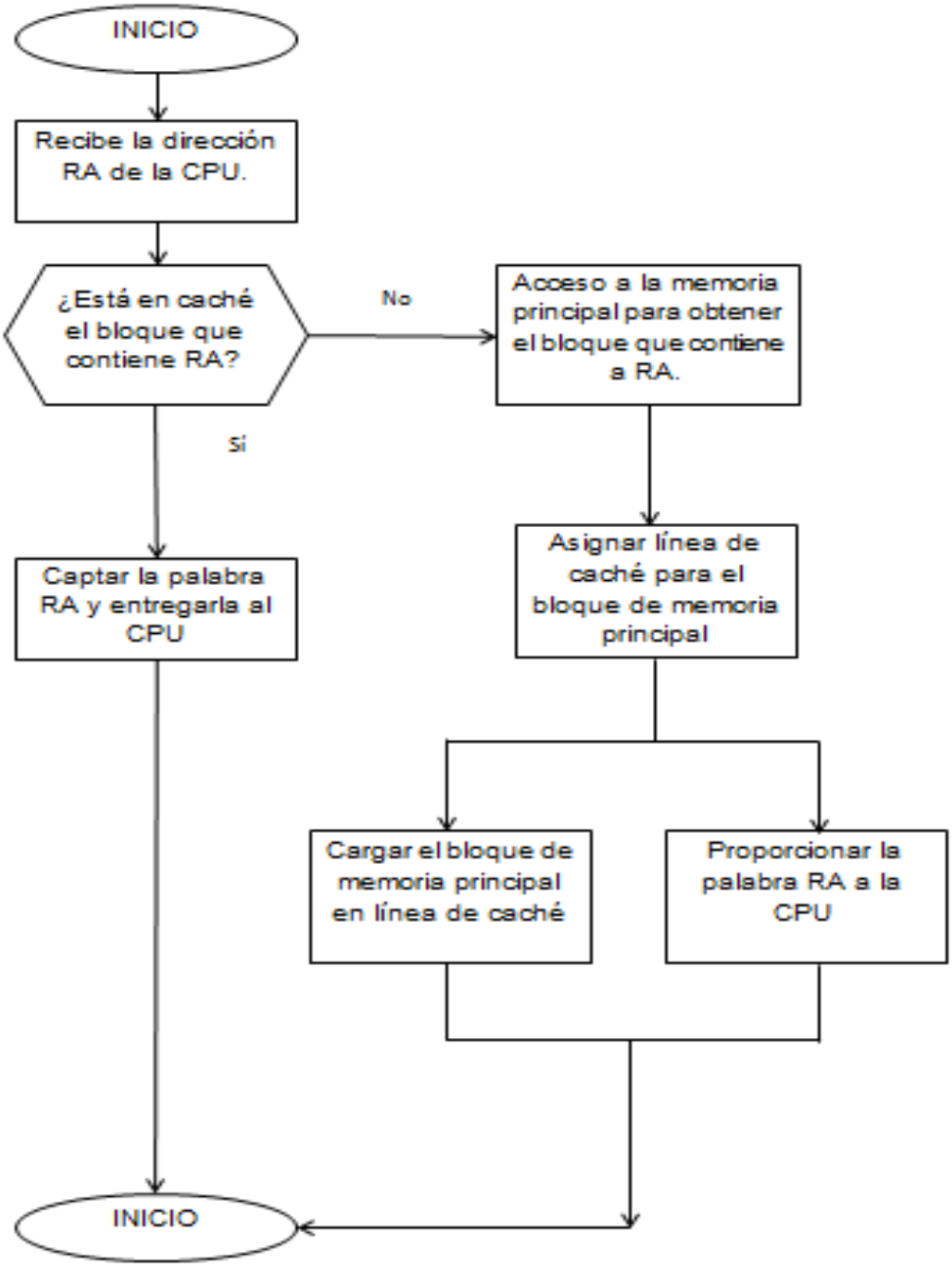


Figura 2.3.2. Operación de lectura de caché. (Stallings, México 2007).

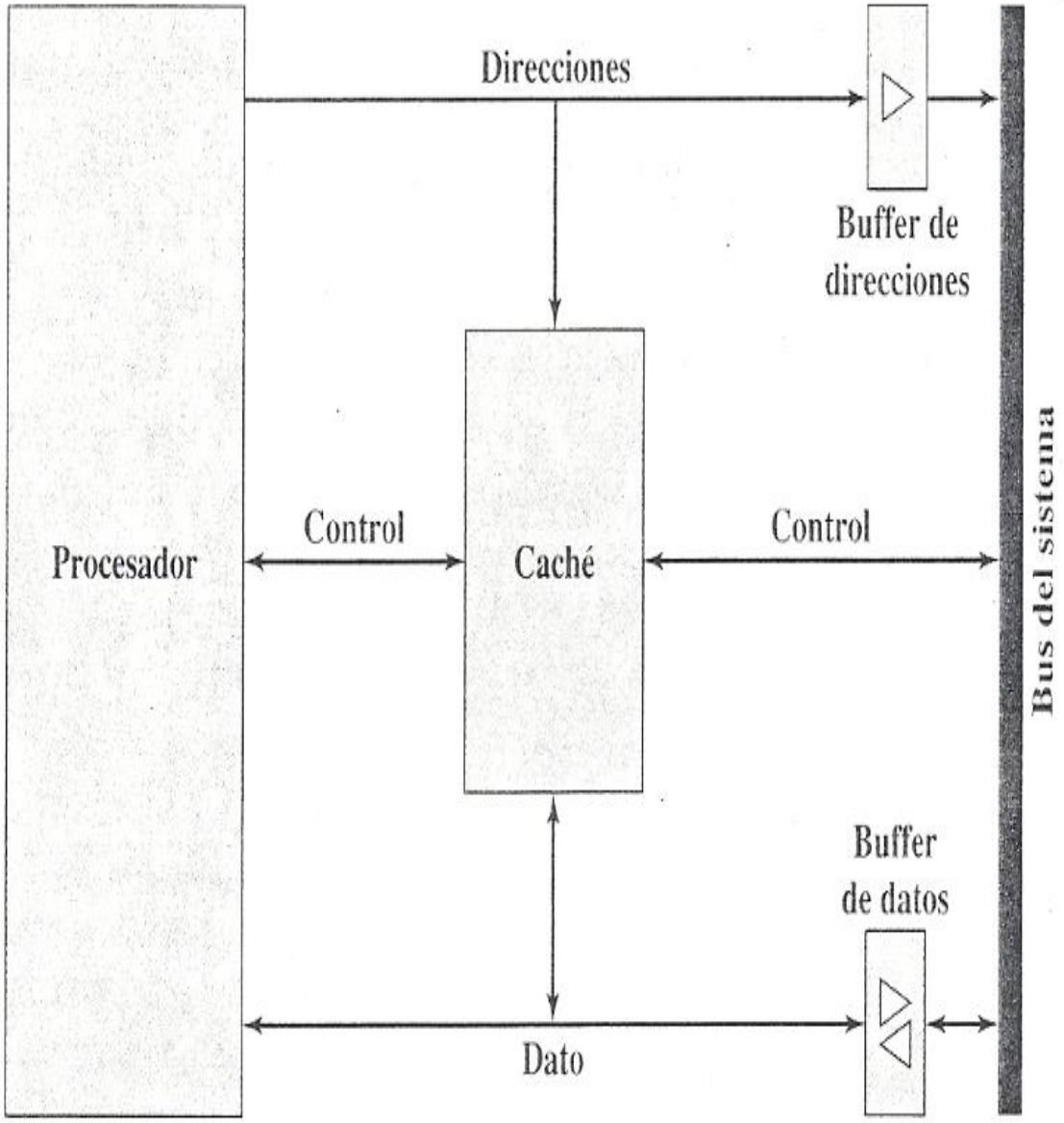


Figura 2.3.3. Organización típica de caché. (Stallings, México 2007).

2.3.1 Elementos de diseño de la caché.

Aunque existen diversas implementaciones de caché, existen unos cuantos criterios básicos de diseño que sirven para clasificar y diferenciar entre arquitecturas de caché. Los elementos de diseño de la caché son (Willians Stalling,2007)

2.3.1.1 Tamaño de caché.

El primer elemento, el tamaño de la caché, ya ha sido tratado. No nos gustaría que el tamaño fuera lo suficientemente pequeño como para que le coste total medio por bit se aproxime al de la memoria principal sola, y que fuera lo suficientemente grande como para que el tiempo de acceso medio total sea próximo al de la caché sola.

Hay otras muchas motivaciones para minimizar el tamaño de la caché. Cuanto más grande es, mayor es el número de puertas implicadas en direccionar la cachés.

El resultado es que cachés grandes tienden a ser ligeramente más lentas que las pequeñas (incluso estando fabricadas con la misma tecnología de circuito integrado y con la misma ubicación en el chip o en la tarjeta de circuito impreso). El tamaño de la caché está también limitado por las superficies disponibles de chip y tarjeta.

Como las prestaciones de la caché son muy sensibles al tipo de tarea, es imposible predecir un tamaño<<óptimo>>. La tabla 2.3.4 lista los tamaños de caché de diversos procesadores antiguos y moderno

Procesador	Tipo	Año de introducción	Caché L1	Caché L2	Caché L3
IBM 360/55	Gran computador	1968	16 a 32KB	-----	-----
PDD – 11/70	Minicomputador	1975	1 KB	-----	-----
VAX 11/780	Minicomputador	1978	16 KB	-----	-----
IBM 3033	Gran computador	1978	64 KB	-----	-----
IBM 3090	Gran computador	1985	128 a 26KB	-----	-----
Intel 80486	PC	1989	8 KB	-----	-----
Pentium	PC	1993	8KB / 8KB	256 A 512 KB	-----
Power PC 601	PC	1993	32 KB	-----	-----
Power PC 620	PC	1996	32 KB/ 32KB	-----	-----
Power PC 64	PC/servidor	1999	32 KB/ 32 KB	256 KM 1 MB	2MB
IBM S/390 G4	Gran computador	1997	32 KB	256 KB	2MB
IBM S/390 G6	Gran computador	1999	256 KB	8 MB	-----
Pentium 4	PC/servidor	2000	8 KB/ 8KB	256 KB	-----
IBM SP	Servidor de gama alta/Supercomputador	2000	64 KB/ 32KB	8 MB	-----
CRAY MTA[b]	Supercomputador	2000	8KB	2 MB	-----
Itanium	PC/servidor	2001	16 KB/ 16KB	96 KB	4 MB
SGI Origin 2001	Servidor de gama alta	2001	32 KB/ 32 KB	4MB	-----
Itanium 2	PC/ servidor	2002	32 KB	256 KB	6 MB
IBM POWER5	Servidor de gama alta	2003	64 KB	1,9MB	36 MB
CRAY XD-1	Supercomputador	2004	64 KB/ 64 KB	1 MB	-----

Tabla 2.3.4. Tamaños de caché de algunos procesadores. (Stallings, México 2007).

2.3.1.2 Función de correspondencia.

Ya que hay menos líneas de caché que bloques de memoria principal, se necesita un algoritmo que haga corresponder bloques de memoria principal a líneas de caché. Además, se requiere algún medio para determinar qué bloque de memoria principal ocupa actualmente una línea dada de caché.

La elección de la función de correspondencia determina cómo se organiza la caché. Pueden utilizarse tres técnicas: directa, asociativa, y asociativa por conjuntos.

2.3.1.2.1 Correspondencia Directa.

La técnica más sencilla, consiste en hacer corresponder cada bloque de memoria principal a solo una línea posible de caché. La Figura 2.3.5 ilustra el mecanismo general de está correspondencia.

La correspondencia se expresa como:

$$i = j \text{ módulo } m$$

Dónde:

i = número de línea de caché.

j = número de bloque de memoria principal

m = número de líneas en la caché

La función de correspondencia se implementa fácilmente utilizando la dirección. Desde el punto de vista del acceso a caché, cada dirección de memoria principal puede verse como dividida en tres campos.

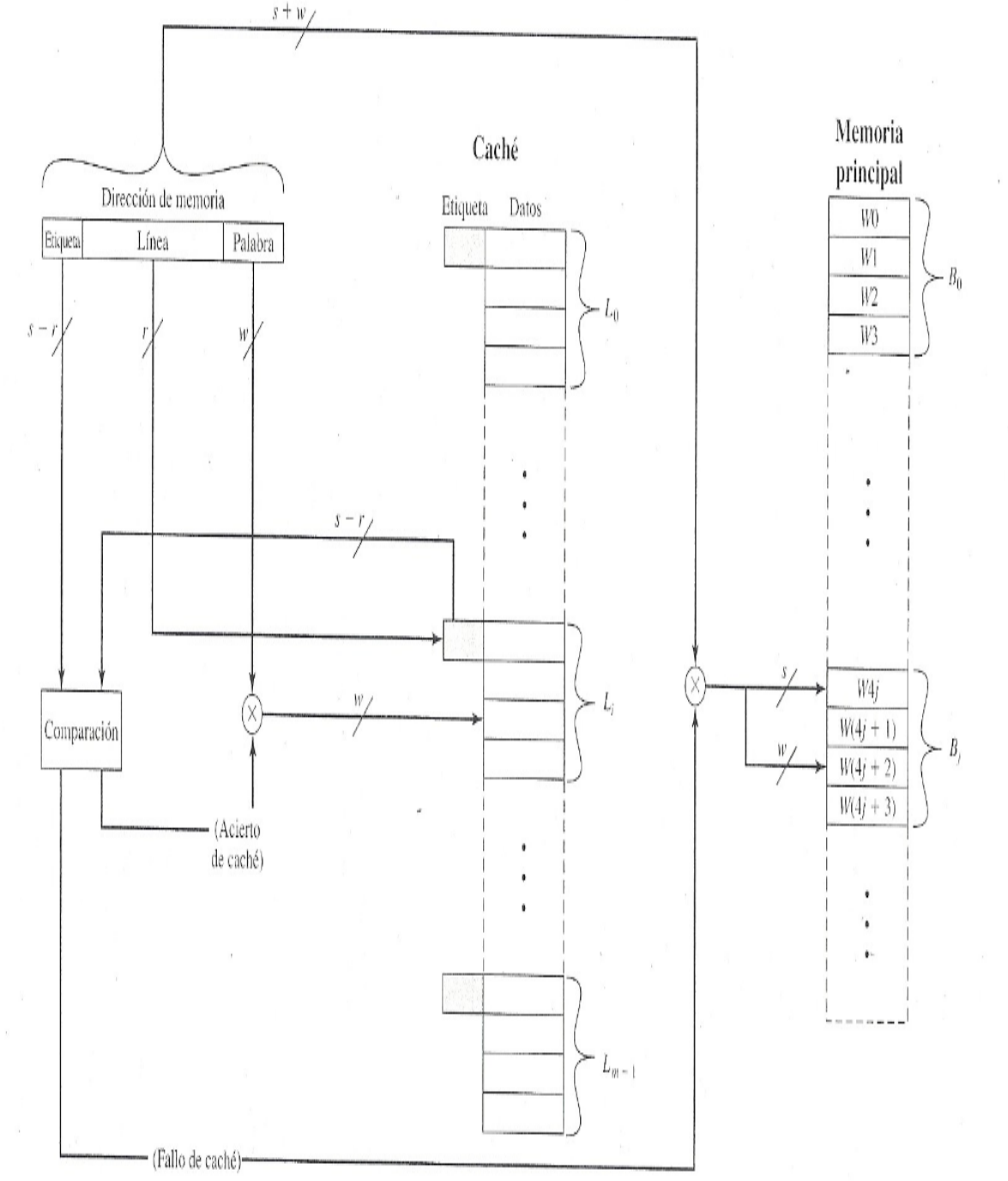


Figura 2.3.5. Organización de caché con correspondencia directa. (HWAN93).

(Stallings, México 2007).

Los *w* bits menos significativos identifican cada palabra dentro de un bloque de memoria principal: en la mayoría de las máquinas actuales, el direccionamiento es a nivel de bytes.

Los s bits restantes especifican uno de los 2^n bloques de la memoria principal. La lógica de la caché interpreta estos s bits como una etiqueta de s-r bits (parte más significativa) y un campo de línea de r bits.

Este último campo identifica una de las $m = 2^r$ líneas de la caché.

Resumiendo:

- Longitud de las direcciones = (s + w) bits.
- Número de unidades direccionables = 2^{s+w} palabras o bytes.
- Tamaño del bloque = tamaño de línea = 2^w palabras o bytes.
- Número de bloques en memoria principal = $\frac{2s+w}{2w} = 2^s$.
- Número de líneas en caché = $m = 2^r$.
- Tamaño de la etiqueta = (*s*-*r*) bits.

El resultado es que se hacen corresponder bloques de memoria principal a líneas de caché de la siguiente manera:

Línea de caché	Bloques de memoria principal asignados
0	0, m, 2m, …, $2^s - m$
1	1, m + 1, 2m + 1, …, $2^s - m + 1$
.	.
m -1	m – 1, 2m – 1, 3m-1, … , $2^s - 1$

Por lo tanto, el uso de una parte de la dirección como número de línea proporciona una correspondencia o asignación única de cada bloque de memoria principal en la caché.

Cuando un bloque es realmente escrito en la línea que tiene asignada, es necesario etiquetarlo para distinguirlo del resto de los bloques que pueden introducirse en dicha línea. Para ello se emplean los *s-r* bits más significativos.

La técnica de correspondencia directa es sencilla y poco costosa de implementar. Su principal desventaja es que hay una posición concreta de caché para cada bloque dado. Por ello, si un programa hace referencia repetidas veces a palabras de dos bloques diferentes asignados en la misma línea, dichos bloques se estarán intercambiando continuamente en la caché, y la tasa de aciertos seria baja.

2.3.1.2.2 Correspondencia Asociativa. La correspondencia asociativa supera la desventaja directa, permitiendo que cada bloque de memoria principal pueda cargarse en cualquier línea de la caché.

En este caso, la lógica de control de la caché interpreta una dirección de memoria simplemente como una etiqueta y un campo de palabra. El campo de etiqueta identifica unívocamente un bloque de memoria principal.

Para determinar si un bloque está en la caché, su lógica de control debe examinar simultáneamente todas las etiquetas de líneas para buscar una coincidencia. La Figura 2.3.6 muestra la lógica de esta correspondencia.

Observe que ningún campo de la dirección corresponde al número de línea, de manera que el número de líneas de la caché no está fijado por el formato de las direcciones.

En resumen:

- Longitud de las direcciones = (s + w) bits.
- Número de unidades direccionables = 2^{s+w} palabras o bytes.
- Tamaño de bloque = tamaño de línea = 2^{w} palabras o bytes.
- Número de bloques en memoria principal = $\frac{2^{s+w}}{2^{w}} = 2^{s}$.
- Número de líneas en caché = indeterminado.
- Tamaño de la etiqueta = s bits.

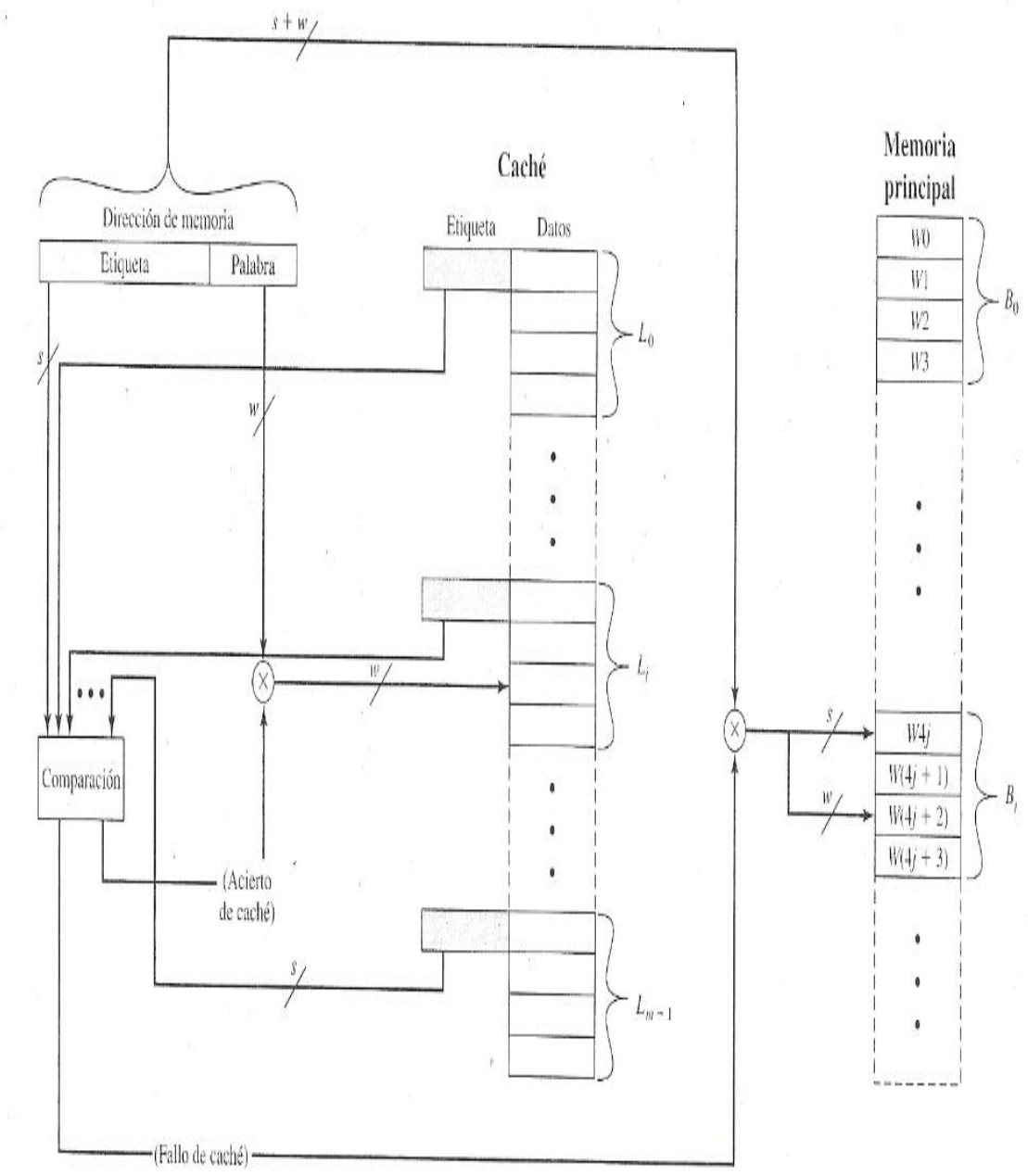


Figura. 2.3.6. Organización de caché totalmente asociativa (HWAN93). (Stallings, México 2007).

Con la correspondencia asociativa hay flexibilidad para que cualquier bloque sea reemplazado cuando se va a escribir uno nuevo en la caché. Los algoritmos de reemplazo o sustitución se diseñan para maximizar la tasa de aciertos.

La principal desventaja de la correspondencia asociativa es la compleja circuitería necesaria para examinar en paralelo las etiquetas de todas las líneas de caché.

2.3.1.2.3 Correspondencia Asociativa por Conjuntos. La correspondencia asociativa por conjuntos es una solución de compromiso que recoge lo positivo de las correspondencias directa y asociativa, sin presentar sus desventajas.
En este caso, la caché se divide en v conjuntos, cada uno de k líneas. Las relaciones que se tienen son:

$$m = v \times k$$

$$i = j \text{ módulo } v$$

Dónde:

i = número de conjuntos de caché

j = número de bloque de memoria principal

m = número de líneas de la caché

En este caso se denomina correspondencia asociativa por conjuntos de k vías. Con la asignación asociativa por conjuntos, el bloque B_j puede asignarse en cualquiera de las k líneas del conjunto i. En este caso, la lógica de control de la caché interpreta una dirección de memoria como tres campos: etiqueta, conjunto y palabra.

Los d bits de conjunto especifiquen uno de entre $v=2^d$ conjuntos. Los s bits de los campos de etiqueta y de conjunto especifican uno de los 2^s bloques de memoria principal. La Figura 2.3.7 muestra la lógica de control de la caché. Con la correspondencia totalmente asociativa, la etiqueta en una dirección de memoria es bastante larga y debe comprarse con la etiqueta de cada línea en la caché.
Con la correspondencia asociativa por conjuntos de k vías, la etiqueta de una dirección de memoria es mucho más corta y se compara solo con las k etiquetas dentro de un mismo conjunto.

Resumiendo:

- Longitud de las direcciones = (s + w) bits
- Número de unidades direccionables = 2^{s+w} palabras o bytes
- Tamaño de bloque = tamaño de línea = 2^{w} palabras o bytes
- Número de bloques en memoria principal = $\frac{2s+w}{2w} = 2^{s}$
- Número de líneas en el conjunto = k
- Número de conjuntos = v =2^{d}
- Número de líneas en caché = k v = k X 2^{d}
- Tamaño de la etiqueta = (s-d) bits

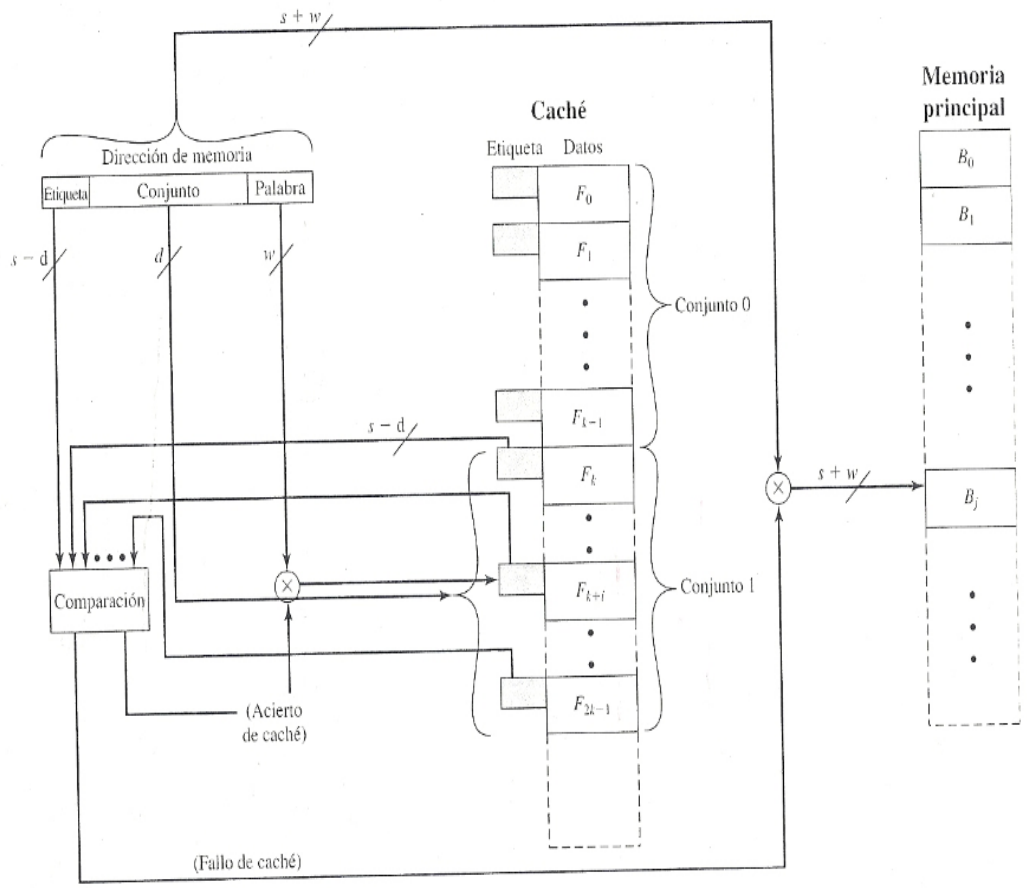


Figura 2.3.7. Estructura de caché asociativa por conjunto de K vías. (Stallings, México 2007).

En el caso extremo de v = m, k = 1, la técnica asociativa por conjuntos se reduce a la correspondencia directa, y para v = 1, k =m, se reduce a la totalmente asociativa. El uso de dos líneas por conjunto (v = m/2, k =2) es el caso más común, mejorando significativamente la tasa de aciertos respecto de la correspondencia directa. La asociativa por conjuntos de cuatro vías (v = m/4. K =4) produce una modesta mejora adicional con un coste añadido relativamente pequeño. Un incremento adicional en el número de líneas por conjunto tiene poco efecto.

2.3.1.3Algoritmo de sustitución.

Una vez que se ha llenado la caché, para introducir un nuevo bloque debe sustituirse uno de los bloques existentes. Para el caso de correspondencia directa, solo hay una posible línea para cada bloque particular y no hay elección posible. Para las técnicas asociativas se requieren algoritmos de sustitución.

Para conseguir alta velocidad, tales algoritmos deben implementarse en hardware. Se han probado diversos algoritmos; mencionaremos cuatro de los más comunes.

2.3.1.3.1 Utilizando menos recientemente (LRU).

Se sustituye el bloque que se ha mantenido en la caché por más tiempo sin haber sido referenciado. Esto es fácil de implementar para la asociativa por conjuntos de dos vías. Cada línea incluye un bit de USO. Cuando una línea es transferida se pone a 1 su bit USO y a 0 el de la otra línea del mismo conjunto. El LRU da la mejor tasa de aciertos.

2.3.1.3.2 Primero en entrar-primero en salir (FIFO).

Sustituye aquel bloque del conjunto que ha estado más tiempo en la caché. Esté algoritmo puede implementarse fácilmente mediante un buffer circular.

2.3.1.3.3 Utilizado menos frecuentemente (LFU).

Sustituye aquel bloque del conjunto que ha experimentado menos referencias. LFU se implementa asociando un contador a cada línea.

2.3.1.3.4 Aleatorio.

Está sustitución aleatoria proporciona unas prestaciones solo ligeramente inferiores a un algoritmo.

2.3.1.4 Políticas de escritura.

2.3.1.4.1 Escritura inmediata.

Utilizando esta técnica todas las operaciones de escritura se hacen tanto en caché como en memoria principal, asegurando que el contenido de la memoria principal siempre es válido. La principal desventaja de esta técnica es que genera un tráfico sustancial con la memoria que puede originar un cuello de botella.

2.3.1.4.2 Postescritura.

Con esta técnica alternativa, las actualizaciones se hacen solo en caché. Cuando tienen lugar una actualización se activa un bit de actualizar asociado a la línea. Después cuando el bloque es sustituido, es (Post)escrito en memoria principal si y solo si el bit ACTUALIZAR está activo. El problema de este esquema es que tiene proporciones de memoria principal que no son válidas, y los accesos por parte de los módulos de entrada y salida tendrán que hacerse a través de la caché. Esto complica la circuitería y genera un cuello de botella.

2.3.1.5 Tamaño de línea.

Otro elemento de diseño es el tamaño de línea. Cuando se recupera y ubica en caché un bloque de datos, se recuperan no sólo la palabra deseada sino además algunas palabras adyacentes.

A medida que aumenta el tamaño de bloque, la tasa de aciertos primero aumenta debido al principio de localidad, el cual establece que es probable que los datos en la vecindad de una palabra referenciada sean referenciados en un futuro próximo. Al aumentar el tamaño de bloque, más datos útiles son llevados a la caché. Sin embargo, la tasa de aciertos comenzará a decrecer cuando el tamaño del bloque se haga aún mayor y la probabilidad de utilizar la nueva información captada se haga menor que la de reutilizar la información que tiene que reemplazarse.

Dos efectos concretos entran en juego:

- Bloques más grandes reducen el número de bloques que caben en la caché. Dado que cada bloque captado se escribe sobre contenidos anteriores de la caché, un número reducido de bloques da lugar a que se sobrescriban datos poco después de haber sido captados.

- A medida que un bloque se hace más grande, cada palabra adicional está más lejos de la requerida y por lo tanto más improbable que sea necesaria a corto plazo.

La relación entre tamaño de bloque y tasa de aciertos es compleja, dependiendo de las características de localidad de cada programa particular, no habiéndose encontrado un valor optimo definitivo. Un tamaño entre 8 y 64 bytes parece estar razonablemente próximo al óptimo.

Parasistemas HPC es más frecuente usar tamaños de línea de caché de 64 y 128 bytes.

2.3.1.6 Número de cachés (ver capítulo 4).

- Uno o varios niveles

- Unificada o partida

CAPÍTULO 3 IDENTIFICACIÓN DE CACHÉS POR ACCESO.

En este capítulo identificaremos los tipos de caché mediante una pequeña clasificación por su acceso y el capítulo constara de dos secciones: ya que aquí mencionamos a la memoria RAM y a la memoria CAM

Celda de Memoria.

En una celda de memoria se guarda un bit, siendo su diagramación lógica la siguiente:

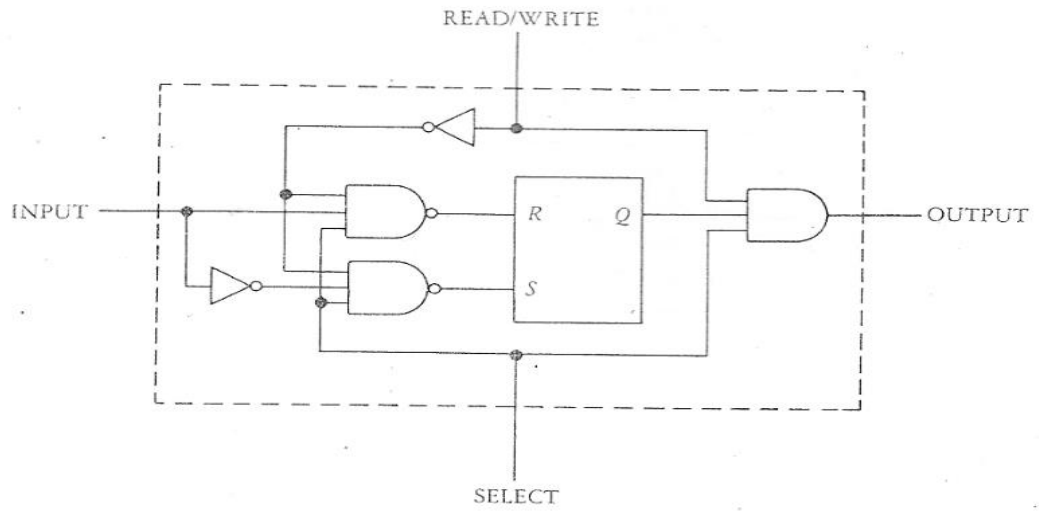


Figura 3.1.1.1 Celda de memoria que guarda un bit. A. K DEWDNEY, 1989

En esta celda se utiliza un Flip Flop SR para poder guardar un bit. Se tendrán dos líneas de control, una de ellas sirve para seleccionar a la celda de memoria y la otra indica la operación a seguir (lectura/escritura). Además, tiene dos líneas de datos para la entrada/salida.

3.1.1 RAM (Memoria de Acceso Aleatorio - Random Access Memory).

Es una memoria que tiene un mecanismo de acceso aleatorio, es decir, memorias en la que una dirección de una palabra se accede a la misma velocidad como el acceso a cualquier otra dirección, diremos que la RAM tiene su memoria organizada en palabras, y cada palabra tiene su propia dirección.

Por lo tanto, cualquier posición puede seleccionarse <<aleatoriamente>>, ser direccionada y accedida directamente. La dirección de memoria se localiza en el registro de acceso a memoria (Memory Address Register, MAR), y la selección se realiza por medio de un decodificador de N X 2^N donde N es el tamaño de palabra de la MAR pudiéndose escoger cualquiera de las 2^N palabras de la memoria RAM.

En la figura 3.1.1.2 se representa una memoria de 8 palabras y cada palabra tiene 4 bits (por lo que se tiene un decodificador 3X8). En la figura, cada cuadro nos representa un bit de almacenamiento (una celda de memoria).

La línea de selección depende del decodificador e indica cuál de las palabras será escogida para escribir/leer. Se tienen una línea de entrada de datos, otra línea para indicar el control de lectura/escritura y una línea de salida para cada uno de los bits. La línea de salida se conectará a un MBR (Memory Buffer Register)

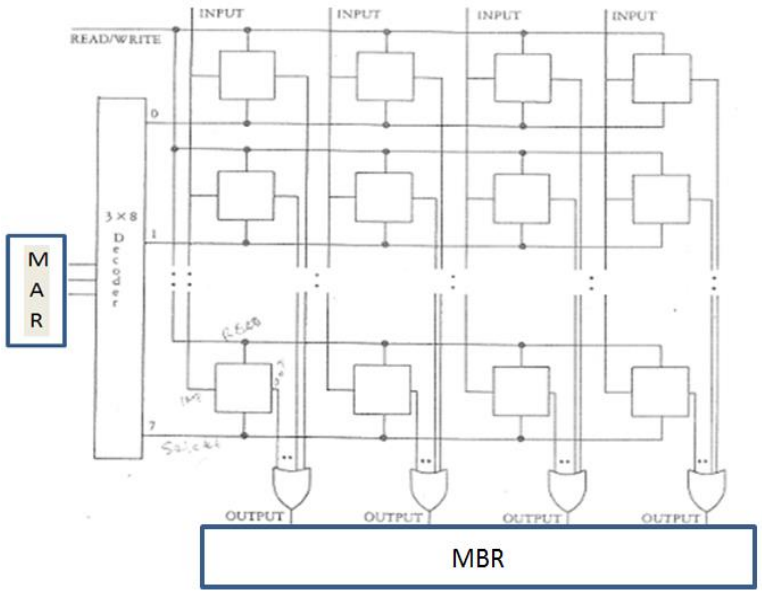


Figura. 3.1.1.2 Diseño de una memoria

A, K DWENDNEY, 1989

- Tiempo de acceso (latencia): Para memorias de acceso aleatorio, el tiempo de latencia es el tiempo que tarda en realizarse una operación de escritura/lectura, es decir, el tiempo que transcurre desde el instante en el que se presenta una dirección a la memoria hasta que el dato sea leído, o haya sido memorizado, o esté disponible para su uso. Por lo que el tiempo de latencia se incrementa conforme se incremente el tamaño de la memoria RAM. Se puede decir que el tiempo de latencia se incrementa en forma proporcional al tamaño de la memoria.

3.1.2 CAM = Memorias direccionables por contenido (CAM: Content Addresable Memory).

Dado un conjunto de datos (valor), se busca en la memoria para cualquier ubicación que contiene ese valor.

En una memoria asociativa, todas las posiciones de memoria se buscan simultáneamente y en paralelo sobre la base del contenido de los datos para ver si cualquiera contiene el valor deseado. Cualquier ubicación que tiene el valor deseado envía una señal para indicar que se ha encontrado el valor.

Las memorias asociativas realizan una búsqueda paralela por datos y es mucho más rápida que una memoria RAM para la búsqueda de una palabra. En ciertas aplicaciones, tales como operaciones en la tabla de búsqueda, puede ser una organización de la memoria alternativa muy útil. Sin embargo, como veremos en breve, las memorias asociativas son más caras debido a la circuitería lógica adicional necesaria para la comparación y selección.

Un diagrama de bloques inicial de una memoria asociativa se muestra en la Figura 3.1.2.1.La memoria consiste en las líneas de salida de entrada tradicional y, pero también incluye un registro de argumento (A) y el registro de comparación (M). El tamaño de A es la misma que la longitud de palabra de la memoria. El tamaño de M es igual al número de palabras en la memoria, con cada bit de M correspondiente a una palabra de memoria.

El patrón de datos para ser leído o escrito de la memoria, se almacena en el registro argumento. (Esto es análogo a la MAR de la memoria RAM). Si los datos en el registro argumento coinciden con los datos en cualquiera de las palabras de la memoria, se establece el bit correspondiente en el registro de comparación. Este registro se utiliza para determinar qué palabra (s) para leer o qué palabra (s) para escribir, como se verá en breve.

Sin embargo, por el momento, vamos a ignorar las líneas de control leer/escribir y las líneas de entrada /salida y nos concentramos en la lógica de la búsqueda paralela.

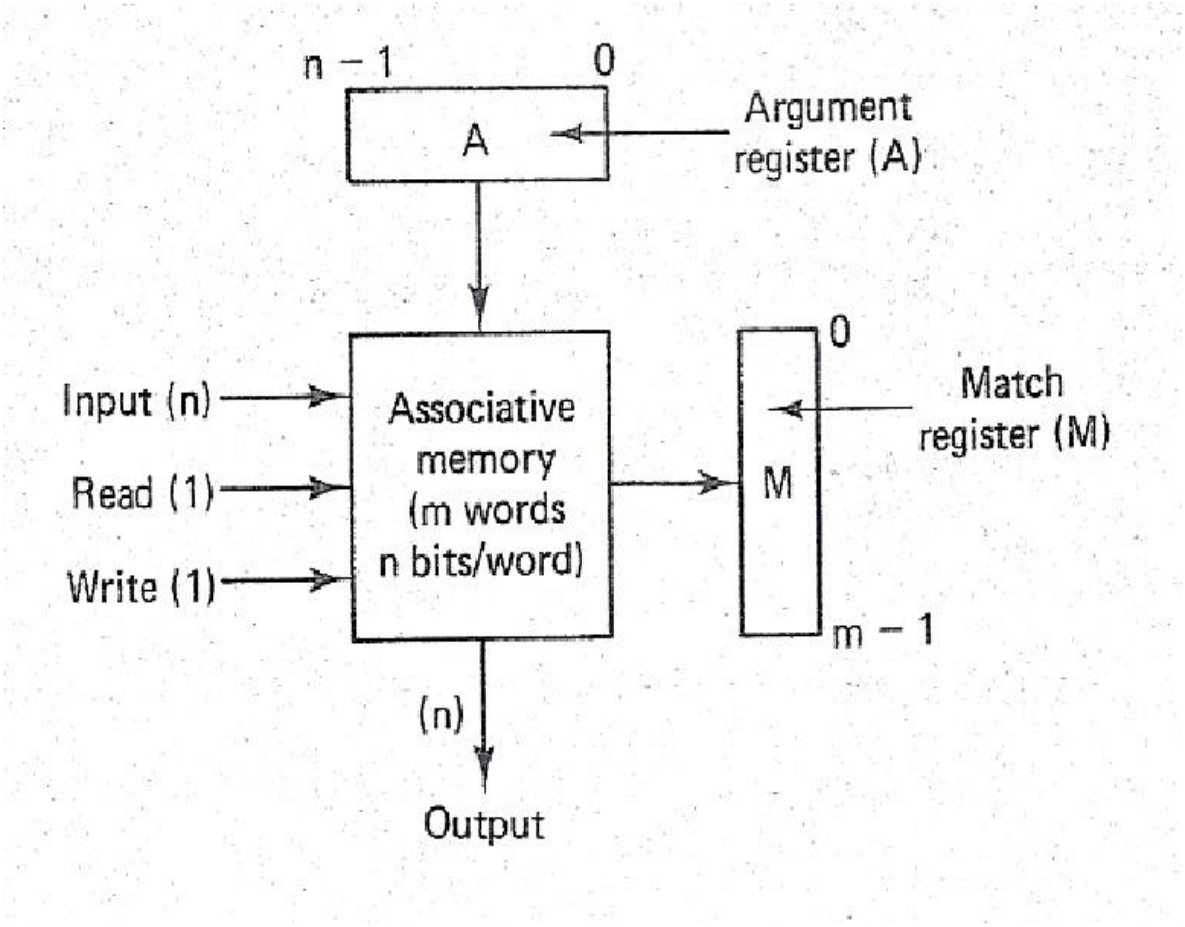


Figura3.1.2.1 Diagrama inicial de una memoria asociativa. Langholz, 1989

Figura3.1.2.2. Muestra la relación básica del registro argumento con cada palabra individual de la memoria. Cada palabra de memoria se compara con el argumento. Esta comparación se realiza en la figura en el cuadro ML donde cada ML se tiene la ecuación 3.1. Si se tiene una coincidencia en una palabra, entonces la i-ésima salida del circuito de ML establece el bit correspondiente en M. Como se observa, se requerirán m circuitos del tipo ML para una memoria de m-palabra. (Gideon & Francioni, New Jersey).

La ecuación 3.1 muestra la forma de comparar un bit del argumento con un bit de cada una de las palabras de la memoria, por lo que el subíndice j representa el bit del argumento a comparar con respecto al j-ésimo bit a comparar de la i-ésima palabra.

$$X_j = A_j W_{ij} + \overline{A}_j \overline{W}_{ij} \qquad (3.1)$$

Por lo tanto $x_j = 1$ si el par de bits en la posición j son iguales (es decir, si ambos son unos o ambos son ceros). Ahora, ML_i establece M sólo si todos los bits de A son iguales a su correspondiente bit de W_i. Eso es, $M_i=1$ (es decir, $A= W_i$) solo si $x_j=1$ para todo j, j = 0, 1,..., n-1, donde n es el número de bits de la palabra. Esta condición requiere una operación AND para todo X_j.

$$(A = W_i) = X_0X_1\ldots X_{n-1} \qquad (3.2)$$

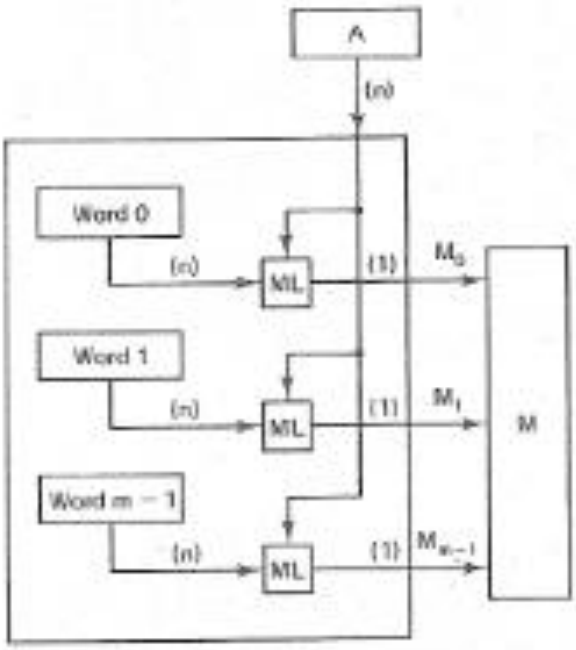


Figura: 3.1.2.2. Relación del argumento regístrese para palabras de memoria (ML=lógica de partido).

Si el valor de esta expresión es 1, entonces $M_i = 1$, de lo contrario, $M_i=0$. El diagrama de bloques lógicos correspondientes a las ecuaciones (3.1) y (3.2) se muestra en la Figura3.1.2.3.

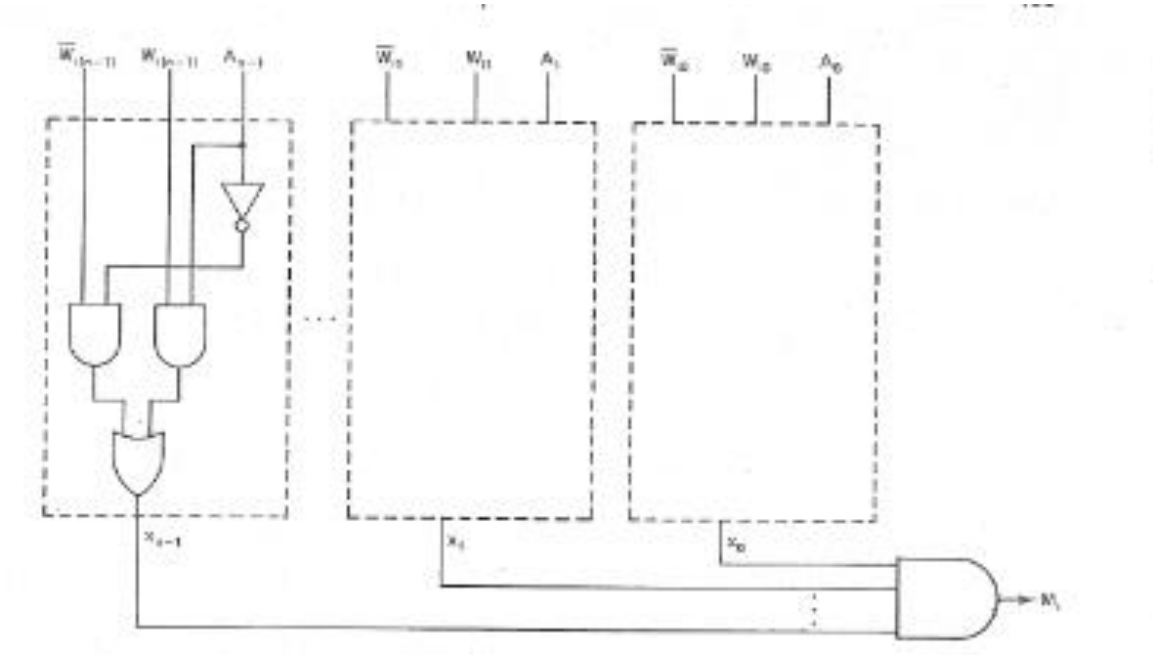


Figura: 3.1.2.3.Circuito lógico (ML). Langholz, 1989

Clave de registro

En general, queremos una memoria asociativa que funcione solo con un determinado campo de bits en lugar de toda la palabra. Para facilitar esto, la memoria incluye un registro de claves (K) que define qué bits de las palabras se deben comparar contra el argumento de registro (A). El contenido del registro de claves es interpretado de tal manera que $K_j = 1$ significa que coincida con j bit de cada palabra de j bits de A, y $K_j = 0$ significa que se debe considerar el j-ésimo bit de todas las palabras con el j-ésimo bit del argumento A. Figura A. 3.1.2.4. Muestra la versión modificada de la figura 3.1.2.1, después de incluir el registro de claves, y el siguiente ejemplo Ilustra este procedimiento.

Ejemplo.

Considerando las tres palabras en memoria:

$W_0 = 110010101$

$W_1 = 101010101$

$W_2 = 110011001$

Si K = 000011111 y A = 110010101, sólo los cinco bits menos significativos de cada palabra de memoria serán comparados con los bits correspondientes de A. por lo tanto W_0 y W_1 son iguales con A, pero W_2 no s igual, por lo que M es 011.

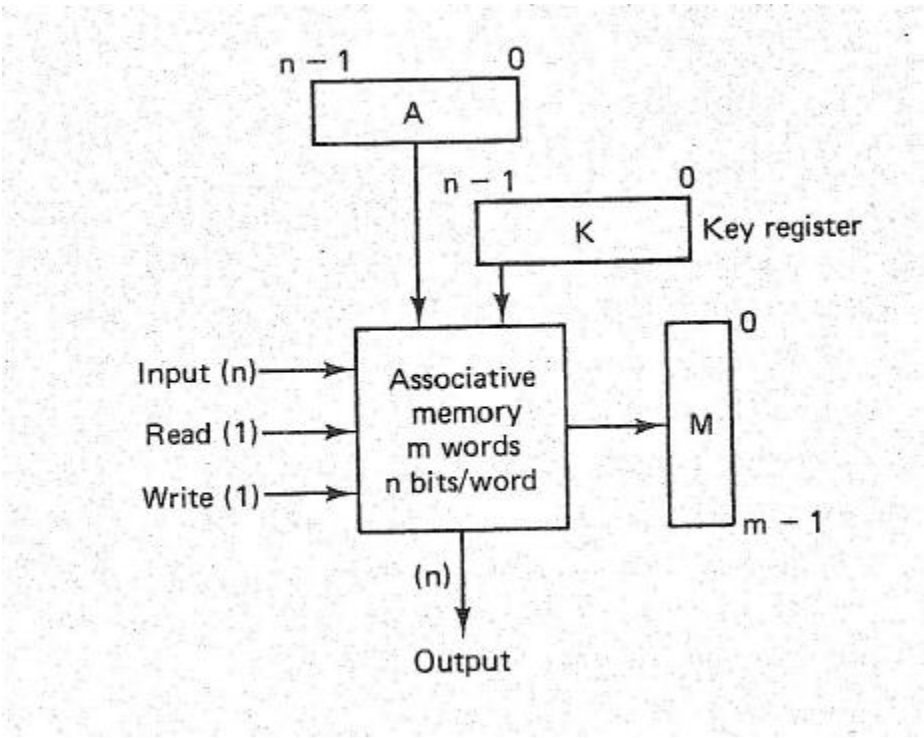


Figura 3.1.2.4. Diagrama de bloques completo de una memoria asociativa. Langholz, 1989

Para incluir el registro de claves en las expresiones lógicas que se obtuvieron antes, observe que no se requiere ninguna comparación entre W_{ij} y Aj cuando K_j =0. Por lo tanto, la ecuación (3.1) tiene que ser modificado para incluir esta condición:

$$X_j = A_j W_{ij} + \overline{A}_j \overline{W}_{ij} + \overline{K}_j \qquad (3.3)$$

En la figura 3.1.2.5 se muestra el diagrama del bloque lógico modificado de la figura 3.1.2.3 después de incluir la clave de registro.

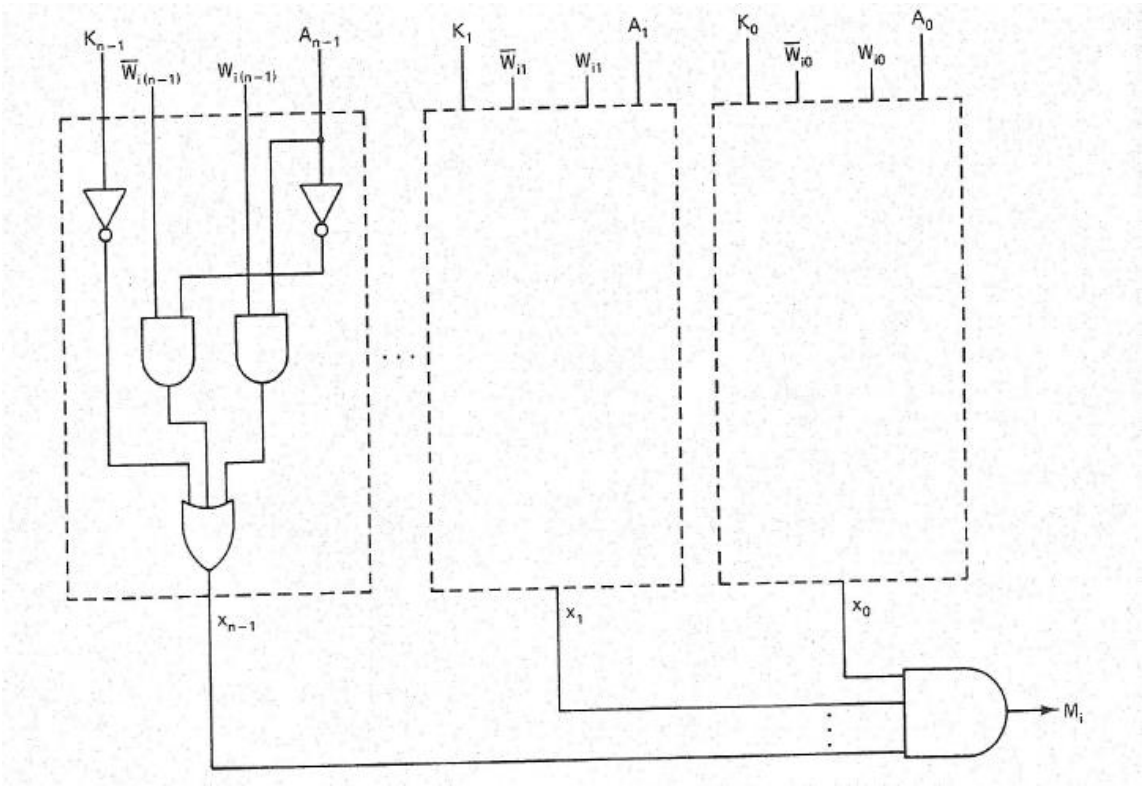


Figura 3.1.2.5. Coincidir con la lógica de clave. Langholz, 1989

Lectura y Escritura.

Considerando la posibilidad de leer y escribir en una memoria asociativa, echemos un vistazo a un bit de una palabra. El circuito lógico correspondiente se muestra en la figura 3.1.2.6 para el bit j de la palabra i.

Dependiendo de lo que se almacena en la memoria, son dos posibles resultados de una lectura. Si más de una palabra coincide con el argumento, es necesario implementar algún medio de la lectura de las palabras coincidentes en secuencia.

Esto se puede hacer mediante la comparación del registro, un bit a la vez, y el establecimiento de la línea de control de lectura. Si la palabra coincide el resultado de salida es un uno en M. En este caso, si una palabra no coincide es todo cero, una salida de todos cero puede implicar que no se encontró ninguna coincidencia.

(Gideon & Francioni, New Jersey).

Al escribir en una memoria asociativa, los bits de comparación se pueden conectar a las líneas de control de escritura para llenar la memoria inicialmente. Muy a menudo, un registro de etiquetas es requerido en adición a otros registros. Contiene un bit por cada palabra de la memoria y es usado para marcar las palabras libres o palabras que están ocupadas. Por ejemplo, si la palabra es libre, se coloca en uno (o se establece, set) el bit correspondiente al registro etiqueta, si está ocupada, el bit correspondiente etiqueta se coloca en cero (o se reestablece, reset).

En general, memorias asociativas son muy rápido para ciertas aplicaciones. Sin embargo, se debe recordar que también son mucho más caras que las memorias RAM y, por lo tanto, aún no son viables como la unidad de memoria principal de una computadora.

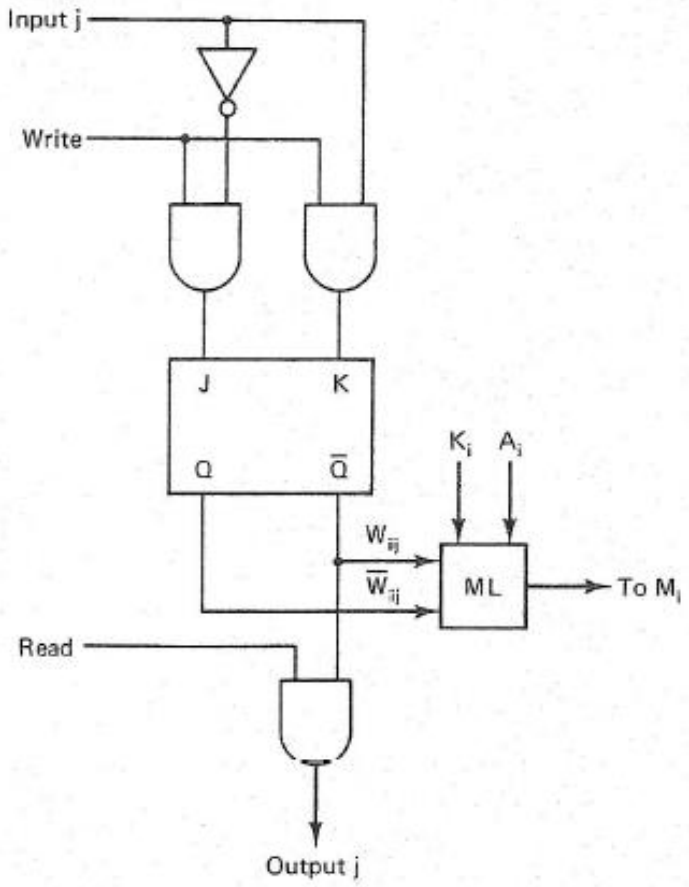


Figura: 3.1.2.6. Lógica de un bit de una palabra de una memoria asociativa.(Gideon & Francioni, New Jersey).

La memoria cache también puede ser vista como un buffer de la memoria principal. Aunque la cache usa información guardada en la memoria principal, no interfiere en forma directa con ella.

Existen cuatro caminos posibles para decremento el tiempo de acceso a memoria:

1. Una RAM pequeña. Con una RAM, el tiempo de acceso es función del número de palabras en la memoria.
   Implementando la cache como RAM comparada con la memoria principal será más rápida que la memoria principal.

2. Una CAM pequeña. Si la cache es implementada como una memoria asociativa, el tiempo de acceso es función del tamaño de la palabra. En general, el tamaño de palabra en la memoria principal es mucho menor que el número de palabras en la memoria.
   Por tal diferencia, la memoria asociativa es más rápida que una memoria principal RAM típica.

3. Semiconductores. Los semiconductores de las memorias usualmente se clasifican en dos tipos: bipolar y MOS (Metal-Oxide Semiconductor). En general, memorias MOS son usadas en la memoria principal por su bajo costo.
   Las memorias bipolares son más rápidas (y, por supuesto, más caras) y son con frecuencia utilizadas para memorias cache pequeñas.

4. La combinación de las tres anteriores. En casi todas las computadoras, las memorias cache son hechas con una combinación de las memorias explicadas en los puntos 1 a 3.

CAPÍTULO 4 CLASIFICACIÓN DE MEMORIAS POR LUGAR.

En esta sección del capitulado contaremos con una clasificación muy peculiar acerca de las caches ya que recientemente, se ha convertido en una norma del uso de múltiples cachés

4.1 Número de cachés. Cuando se introdujeron originalmente las cachés, un sistema tenía normalmente solo una caché. Más recientemente, se ha convertido en una norma el uso de múltiples cachés.

Hay dos aspectos de diseño relacionados con este tema que son el número de niveles de caché, y el uso de caché unificada frente al de cachés separadas.

***4.2 Cachés multinivel*.** Con el aumento de densidad de integración ha sido posible tener una caché en el mismo chip del procesador: caché *on-chip*. Comparada con la accesible a través de un bus externo, la caché *on-chip* reduce la actividad del bus externo del procesador y por tanto reduce los tiempos de ejecución e incrementa las prestaciones globales del sistema.

Cuando la instrucción o el dato requeridos se encuentran en la caché *on-chip*, se elimina el acceso al bus. Debido a que los caminos de datos internos al procesador son muy cortos en comparación con la longitud de los buses, los accesos a la caché *on-chip* se efectúan apreciablemente más rápidos que los ciclos de bus, incluso en ausencia de estados de espera. Además, durante este periodo el bus está libre para realizar otras transferencias. (Stallings, México 2007).

La búsqueda de información comienza por la caché L1, y se va subiendo nivel a nivel en caso de no encontrar lo que se busca en el nivel actual. Cuantas más capas se asciende, mayor es el tiempo de espera. Pero, a mayor cercanía a la CPU, la probabilidad de encontrar lo que se busca es mayor.

Esta forma de trabajo resulta una excelente relación de compromiso entre diversos factores, y consigue mejorar el rendimiento del ordenador de forma notable.

INFORMACIÓN DE MEMORIA CACHÉ Y VELOCIDAD DE ALGUNOS PROCESADORES INTEL			
MODELO	VELOCIDAD EN MHz	CACHÉ L1 EN KB	CACHÉ L2 EN KB
8088	8	0	0
8086	8	0	0
80c86	8	0	0
80186	16	0	0
80286	20	0	0
80386DX	40	0	0
80386SX	25	0	0
80486SLC	25	8	0
	33	8	0
80486DX	25	8	0
	33	8	0
	50	8	0
80486SX	20	8	0
	25	8	0
	33	8	0
80486DX2	40	8	0
	50	8	0
	66	8	0
80486DX4	75	16	0
	100	16	0
Pentium	75	16	0
	100	16	0
Pentium MMX	166	32	0
	233	32	0
Pentium Pro	150	16	256-512
	200	16	256-512
Pentium II	233	32	512
	450	32	512
Pentium II Xeon	400	32	512
	450	32	512
Pentium III	450	32	512
	1000	32	256
Pentium III Xeon	500	32	512
	800	32	256
Celeron	266	32	0
	600	32	128
Pentium 4	1400	32	256
	2000	32	512

(Bacalla, Noviembre 2009).

Tabla 4.1. Niveles de memoria cache en diferentes arquitecturas de CPU

4.3 Niveles de caché

Los diferentes tipos de caché se organizan por niveles, formando una jerarquía.
En general se cumple que, *a mayor cercanía a la CPU, se presenta mayor velocidad de acceso y menor capacidad de almacenamiento.*

A) Caché de memoria: De acuerdo a la ubicación física que tienen en el sistema se denominan o identifican por niveles:

4.3.1 Nivel 1 (L1): Conocido como caché interno, es el nivel más cercano a la CPU (está en el mismo núcleo) con lo que el acceso se produce a la velocidad de trabajo del procesador (la máxima velocidad). Presenta un tamaño muy reducido, en Intel (4 a 32 KB), en VIA/Cyrix (1 a 64 KB), en AMD (8 a 128 KB). (Bacalla, Noviembre 2009).

Por lo tanto la memoria caché L1 forma parte del procesador.

Se puede dividir en dos secciones:

- *L1 para datos*
- *L1 para instrucciones*

- L1 DC: "*Level 1 date cache*": se encarga de almacenar datos usados frecuentemente y cuando sea necesario volver a utilizarlos, como se utilizan en forma inmediata se agilizan los procesos.

- L1 IC: "*Level 1 instruction cache*": se encarga de almacenar instrucciones usadas frecuentemente y cuando sea necesario volver a utilizarlas, inmediatamente las recupera, por lo que se agilizan los procesos.(Canto Q, Febrero 2004).

4.3.2 Nivel 2 (L2): La memoria caché de segundo nivel (L2) es una memoria muy rápida llamada SRAM (RAM estática) que se coloca entre la memoria principal y la CPU y que almacena los últimos datos transferidos.

La mejora potencial del uso de una caché L2 depende de las tasas de aciertos de ambas cachés L1 y L2. Varios estudios han demostrado que, en general, el uso de un segundo nivel de caché mejora las prestaciones. No obstante, el uso de cachés multinivel complica todos los aspectos de diseño de la caché, incluyendo el tamaño, el algoritmo de sustitución y la política de escritura

A partir de los procesadores Pentium 4 viene incorporada en el procesador (no precisamente en el núcleo). El nivel L2 apareció con el procesador Pentium Pro, es una memoria más lenta que L1, pero de mayor capacidad. Los tamaños típicos de la memoria caché L2 oscilan en la actualidad entre 256 KB y 4 MB. (Bacalla, Noviembre 2009).

El procesador primero consulta a dicha memoria intermedia para ver si la información que busca está allí, en caso afirmativo podemos trabajar con ella sin tener que esperar a la memoria principal.

No hay que confundir nunca la memoria de segundo nivel con la de primer nivel (L1) ya que esta suele ir integrada dentro del procesador, y suele ser de menor capacidad, aunque evidentemente dispone de un acceso mucho más rápido por parte de la CPU.(Canto Q, Febrero 2004).

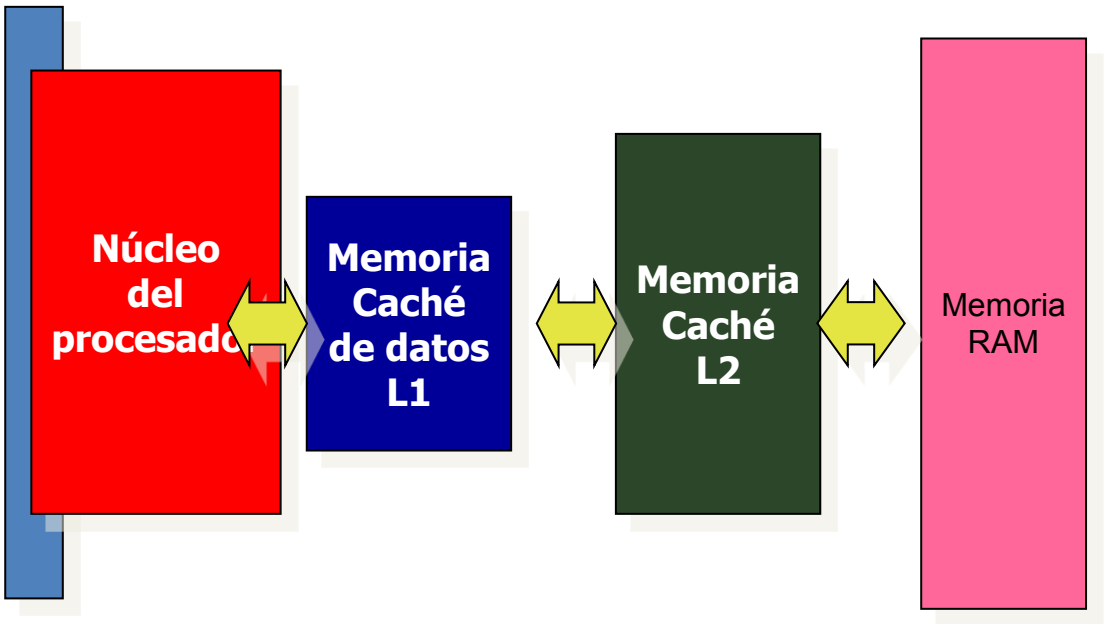


Figura 4.2 Niveles uno y dos de memoria cache.

4.3.3 Nivel 3 (L3): Se encuentra en algunas placas base, procesadores y tarjetas de interfaz. El procesador de Intel Itanium trae contenida en su cartucho al nivel L3 que soporta un tamaño hasta de 4 MB, y el Itanium 2 tolera hasta 6 MB de caché L3. (Bacalla, Noviembre 2009).

Esta memoria es un tercer nivel que soportan principalmente los procesadores de la firma AMD®. Con este nivel de memoria se agiliza el acceso a datos e instrucciones que no fueron localizadas en L1 ó L2. Si no se encuentra el dato en ninguna de las 3, entonces se accederá a buscarlo en la memoria RAM. (Canto Q, Febrero 2004).

La memoria caché L3 recientemente se ha incorporado on-chip. En cualquiera de los casos, añadir un tercer nivel de caché parece suponer una mejora en las prestaciones.(William Stallings 2006)

4.4 Caché en disco duro: Utilizadas por los navegadores Web y algunos periféricos. (Bacalla, Noviembre 2009).

4.5 Caché unificada frente a cachés separadas. Cuando hicieron su aparición las cachés *on-chip*, muchos de los diseños contenían una sola caché para almacenar las referencias tanto a datos como a instrucciones. Más recientemente, se han hecho normal separar la caché en dos: una dedicada a instrucciones y otra a datos.

Una caché unificada tiene varias ventajas potenciales:

- Para un tamaño dado de caché, una unificada tiene una tasa de aciertos mayor que una caché partida, ya que nivela automáticamente la carga entre captación de instrucciones y de datos. Es decir, si un patrón de ejecución implica muchas más captaciones de instrucciones que de datos, la caché tendrá a llenarse con instrucciones, y si el patrón de ejecución involucra relativamente más captaciones de datos, ocurrirá lo contrario.

- Solo se necesita diseñar e implementar una caché.

A pesar de estas ventajas, la tendencia es hacia cachés separadas, particularmente para maquinas súper-escalares tales como el Pentium y el Power

PC, en las que se enfatiza la ejecución paralela de instrucciones y la pre captación de instrucciones futuras previstas.

La ventaja clave del diseño de una caché partida es que elimina la competición por la caché entre el procesador de instrucciones y la unidad de ejecución. Esto es importante en diseños que cuentan con segmentación de cause (pipelining) de instrucciones. Normalmente el procesador captará instrucciones anticipadamente y llenará un buffer con las instrucciones que van a ejecutarse.

Supongamos ahora que se tiene una caché unificada de instrucciones/datos. Cuando la unidad de ejecución realiza un acceso a memoria para cargar y almacenar datos, se envía la petición a la caché unificada.

Si, al mismo tiempo, el pre captador de instrucciones emite una petición de lectura de una instrucción a la caché, dicha petición será bloqueada temporalmente para que la caché pueda servir primero a la unidad de ejecución permitiéndole complementar la ejecución de la instrucción en curso.

Esta disputa por la caché puede degradar las prestaciones, interfiriendo con el uso eficiente del cauce segmentado de instrucciones. La caché partida supera esta dificultad.

4.6 ORGANIZACIÓN DE CACHÉ EN EL PENTIUM 4.

La evolución de la organización de la caché se observa claramente en la evolución de los microprocesadores de Intel (Tabla 4.3). El 80386 no incluía caché on-chip.

Problema	Solución	Primer procesador en incluirla
Memoria externa más lenta que el bus del sistema	Añadir caché externa usando tecnología de memoria más rápida.	386
El aumento de velocidad de los procesadores hace que el bus se convierta en un cuello de botella para el acceso a caché.	Trasladar la caché externa al mismo chip (caché on-chip), funcionando a la misma velocidad que el procesador.	486
La caché es pequeña debido a la disponibilidad de superficie on-chip limitada.	Añadir una caché L2 externa con tecnología más rápida que la empleada en la memoria principal.	486
Se produce contención cuando las unidades de precaptación de instrucciones y de ejecución necesitan acceder simultáneamente a la caché. En este caso, la unidad de precaptación se bloquea mientras la unidad de ejecución accede a los datos.	Crear cachés separadas de datos y de instrucciones.	Pentium
El incremento de velocidad del procesador hace que el bus externo sea un cuello de botella para el acceso a la caché L2.	Crear un bus externo específico o <<puerta trasera>> (BSB, black-side bus) que funcione a más velocidad que el bus principal. El BSB se dedica a la caché L2.	Pentium Pro
	Llevar la caché L2 al chip del procesador.	Pentium II
Algunas aplicaciones que trabajan con grandes bases de datos deben tener acceso rápido a cantidades de datos. Las cachés on-chip son demasiado pequeñas.	Añadir una caché L3 externa.	Pentium III
	Integrar la caché L3 on-chip.	Pentium 4

Tabla 4.3. Evolución de la caché en los Intel.

El 80486 incluye una sola caché on-chip de 8 KB, utilizando un tamaño de línea de 16bytes y una organización asociativa por conjuntos de cuatro vías.

Todos los procesadores Pentium incluyen dos cachés L1 on-chip, una para datos y otra para instrucciones. Para el Pentium 4, la caché de datos es de 8KB, utilizando un tamaño de línea de 64bytes y una organización asociativa por conjunto de cuatro vías. Posteriormente describiremos la caché de instrucciones del Pentium 4.

El Pentium II incluye también una caché L2 que alimenta a las dos cachés L1. La caché L2 es asociativa por conjuntos de ocho vías, con una capacidad de 256Kb y un tamaño de línea de 128bytes. En el Pentium III se añadió una caché L3 que pasó a ser on-chip en las versiones avanzadas del Pentium 4.

La Figura 4.4 muestra un esquema simplificado de la estructura del Pentium 4, resaltando la ubicación de las tres cachés. El núcleo del procesador consta de cuatro componentes principales:

- Unidad de captación/decodificación: capta instrucciones en orden de la caché L2, las decodifica en una serie de micro-operaciones, y memoriza los resultados en la cache L1de instrucciones.

- Lógica de ejecución fuera-de-orden: planifica la ejecución de micro-operación teniendo en cuenta las dependencias de datos y los recursos disponibles; de forma que puede planificarse la ejecución de micro-operaciones en un orden diferente del que fueron captadas de la secuencia de instrucciones. Si el tiempo lo permite, esta unidad planifica la ejecución especulativa de micro-operaciones que puedan necesitarse en el futuro.

- Unidades de ejecución: estas unidades ejecutan la micro-operaciones, captando los datos necesarios de la caché de datos L1, y almacenado los resultados temporalmente en registros.

- Subsistema de memoria: esta unidad incluye las cachés L2 y L3, y el bus del sistema, que se usa para acceder a la memoria principal cuando en las cachés L1 y L2 tiene lugar un fallo de caché, asi como para acceder a los recursos de E/S del sistema.

A diferencia de la organización de los modelos Pentium anteriores, así como de la mayoría de los demás procesadores, la caché de instrucciones del Pentium 4 está situada entre la lógica de decodificación de instrucciones y el núcleo de ejecución.

El uso de micro-operaciones, sencillas, de longitud fija, posibilita la utilización de segmentación superescalar y técnicas de planificación que mejoran las prestaciones.

Sin embargo, las instrucciones máquina del Pentium son difíciles de decodificar, ya que tienen un número variable de bytes y muchas opciones diferentes.

Se consigue mejorar las prestaciones si dicha decodificación se realiza independientemente de la lógica de planificación y de segmentación.

La caché de datos emplea una política de postescritura: los datos se escriben en memoria principal solo cuando, habiendo sido actualizados, se eliminan de la caché. El procesador Pentium 4 puede configurarse dinámicamente para utilizar la política de escritura inmediata.

La caché L1 de datos controla por dos bits de uno de los registros de control (véase la Tabla 4.5), rotulados CD (Caché Disable: inhabilitar caché) y NW (Not Write Through: no escritura inmediata).

Hay también dos instrucciones del Pentium 4 que pueden utilizarse para controlar la caché: INVD inválida la memoria caché interna e indica que es inválida la caché externa (si la hay); WBINVD postescribe e inválida la caché interna y entonces postescribe e inválida la caché externa.

Las dos cachés, L2 y L3, son asociativas por conjuntos de ocho vías, con un tamaño de línea de 128 bytes.

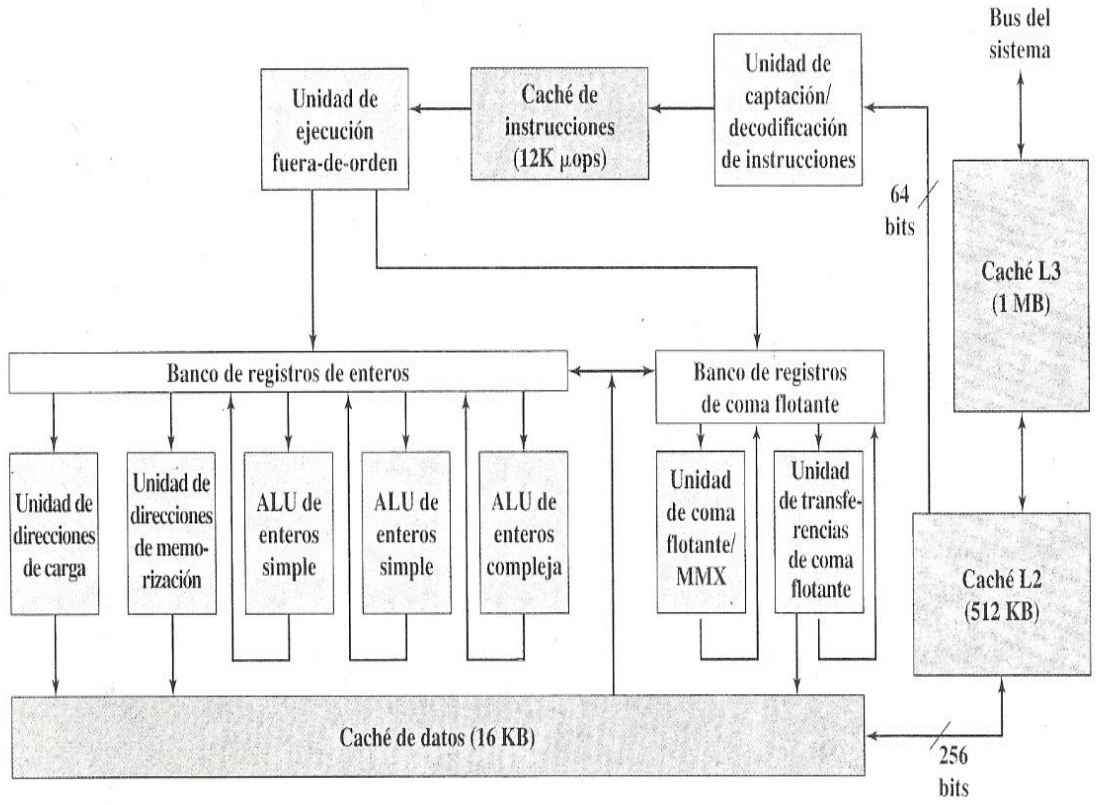


Figura 4.4. Diagrama de bloques del Pentium 4.

Tabla 4.5. Modos de funcionamiento de la caché en el Pentium 4.

Bits de control		Modo de operación		
CD	NW	Llenado de cache	Escrituras inmediatas	Invalidaciones
0	0	Habilitado	Habilitadas	Habilitadas
1	0	Inhabilitado	Habilitadas	Habilitadas
1	1	Inhabilitado	Inhabilitadas	Inhabilitadas

NOTA: CD= 0; NW =1 es una combinación no válida

4.7 Organización de caché en el PowerPC

La organización de caché del PowerPC ha ido evolucionando paralelamente a la arquitectura global de la familia PowerPC, reflejando la búsqueda continua de mejores prestaciones que es el motor de todos los diseñadores de microprocesadores.

La tabla 4.6 muestra esta evolución. El modelo original, el 601, incluye una sola caché de código/datos de 32KB, que es asociativa por conjuntos de ocho vías. El 603 emplea un diseño RISC más sofisticado pero tiene una caché más pequeña: 16KB divididos en cachés separadas de datos de instrucciones, ambas con una organización asociativa por conjuntos de dos vías.

Como resultado, el 603 tiene aproximadamente las mismas prestaciones que el 601 pero un costo menor. En cada modelo posterior, el 604 y el 620, se va duplicando el tamaño de las cachés respecto de su predecesor. Los modelos G3 y G4 tienen el mismo tamaño de cachés L1 que el 620. El G5 proporciona 32KB de caché de instrucciones y 64KB para cachés de datos.

La figura 4.7 es un esquema simplificado de la organización del PowerPC G5, resaltando la ubicación de las dos cachés. Las unidades de ejecución fundamentales son dos unidades aritméticas y lógicas de enteros que pueden ejecutar en paralelo, y dos unidades de coma flotante con sus propios registros y componentes de multiplicación, suma y división. La caché de instrucciones, que se dé solo lectura, alimenta a la unidad de instrucciones.

Las cachés L1 son asociativas por conjuntos de ocho vías. La caché L2 es asociativa por conjuntos de dos vías, con capacidades de 256K, 512K, o 1MB. El G5 admite una caché L3 externa de hasta 1MB, y está previsto incorporar una caché L3 on-chip en implementaciones avanzadas del G5.

Tabla 4.6. Cachés L1 internas en la familia PowerPC.

Modelo	Tamaño	Bytes/línea	Organización
PowerPC 601	1 32-KB	32	Asociativa por conjuntos de 8 vías
PowerPC 603	2 8-KB	32	Asociativa por conjuntos de 2 vías
PowerPC 604	2 16-KB	32	Asociativa por conjuntos de 4 vías
PowerPC 620	2 32-KB	64	Asociativa por conjuntos de 8 vías
PowerPC G3	2 32-KB	64	Asociativa por conjuntos de 8 vías
PowerPC G4	2 32-KB	32	Asociativa por conjuntos de 8 vías
PowerPC G5	1 32-KB 1 64-KB	32	Asociativa por conjuntos de 8 vías

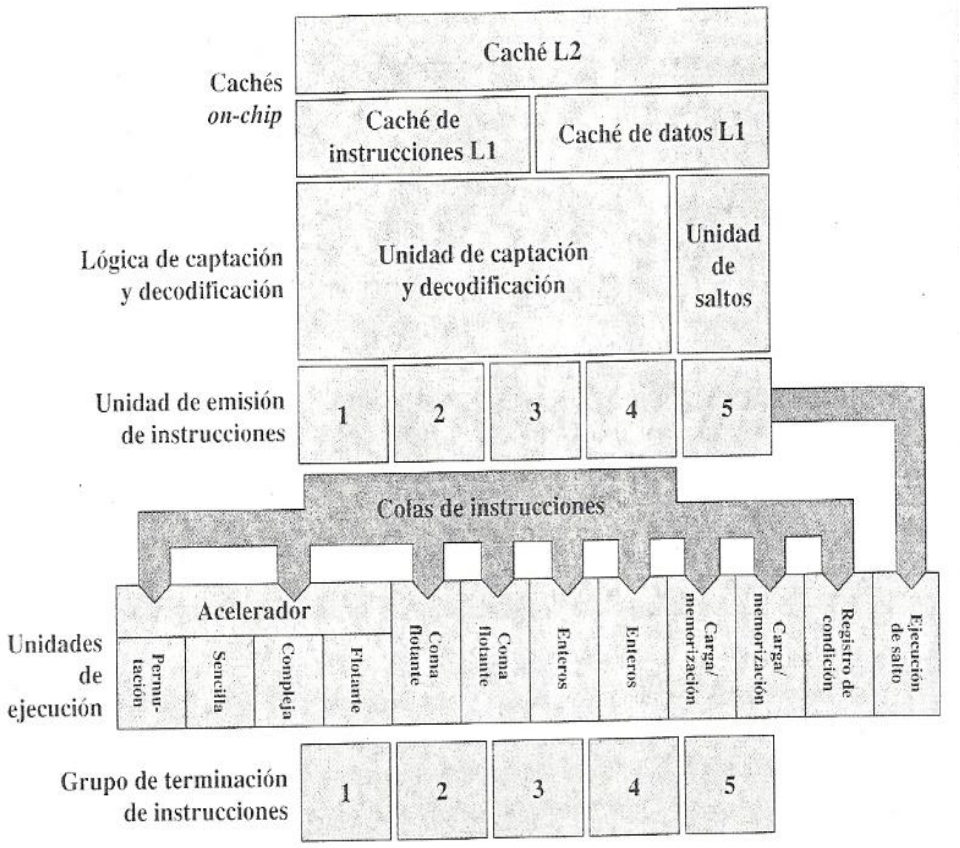


Figura 4.7. Diagrama de bloques del PowerPC G5.

CAPÍTULO 5 ARQUITECTURA MULTIPROCESADORES

La clasificación más conocida para procesamiento se conoce como “clasificación de Flynn". Esta taxonomía de las arquitecturas está basada en la clasificación atendiendo al flujo de datos e instrucciones en un sistema. Un flujo de instrucciones es el conjunto de instrucciones secuenciales que son ejecutadas por un único procesador, y un flujo de datos es el flujo secuencial de datos requeridos por el flujo de instrucciones.

Con estas consideraciones, Flynn clasifica los sistemas en cuatro categorías:

5.1 Clasificación de Flynn.

5.1.1 SISD (*Single Instruction Stream, Single Data Stream*). Flujo único de instrucciones y flujo único de datos. Este el concepto de arquitectura serie de Von Neumann donde, en cualquier momento, sólo se está ejecutando una ´única instrucción.

A menudo a los SISD se les conoce como computadores serie escalares. Todas las máquinas SISD poseen un registro simple que se llama *contador de programa* que asegura la ejecución en serie del programa. Conforme se van leyendo las instrucciones de la memoria, el contador de programa se actualiza para que apunte a la siguiente instrucción a procesar en serie.

Prácticamente ningún computador puramente SISD se fabrica hoy en día ya que la mayoría de procesadores modernos incorporan algún grado de paralización como es la segmentación de instrucciones o la posibilidad de lanzar dos instrucciones a un tiempo (superescalares).

5.1.2 MISD (*Multiple Instruction Stream, Single Data Stream*) Flujo múltiple de instrucciones y ´único flujo de datos. Esto significa que varias instrucciones actúan sobre el mismo y ´único trozo de datos. Este tipo de máquinas se pueden interpretar de dos maneras.

Una es considerar la clase de máquinas que requerirían que unidades de procesamiento diferentes recibieran instrucciones distintas operando sobre los mismos datos. Esta clase de arquitectura ha sido clasificada por numerosos arquitectos de computadores como impracticable o imposible, y en estos momentos no existen ejemplos que funcionen siguiendo este modelo.

Otra forma de interpretar los MISD es como una clase de máquinas donde un mismo flujo de datos fluye a través de numerosas unidades procesadoras. Arquitecturas altamente segmentadas, son clasificadas a menudo bajo este tipo de máquinas.

Las arquitecturas segmentadas, o encauzadas, realizan el procesamiento vectorial a través de una serie de etapas, cada una ejecutando una función particular produciendo un resultado intermedio.

La razón por la cual dichas arquitecturas son clasificadas como MISD es que los elementos de un vector pueden ser considerados como pertenecientes al mismo dato, y todas las etapas del cauce representan múltiples instrucciones que son aplicadas sobre ese vector.

5.1.3 SIMD (*Single Instruction stream, Multiple Data stream).* Flujo de instrucción simple y flujo de datos múltiple. Esto significa que una única instrucción es aplicada sobre diferentes datos al mismo tiempo. En las máquinas de este tipo, varias unidades de procesado diferentes son invocadas por una única unidad de control.

Al igual que las MISD, las SIMD soportan procesamiento vectorial (matricial) asignando cada elemento del vector a una unidad funcional diferente para procesamiento concurrente.

Por ejemplo, el cálculo de la paga para cada trabajador en una empresa, es repetir la misma operación sencilla para cada trabajador; si se dispone de un arquitectura SIMD esto se puede calcular en paralelo para cada trabajador. Por esta facilidad en la paralización de vectores de datos (los trabajadores formarían un vector) se les llama también *procesadores matriciales*.

5.1.4 MIMD (*Multiple Instruction stream, Multiple Data stream*). Flujo de instrucciones múltiple y flujo de datos múltiple. Son máquinas que poseen varias unidades procesadoras en las cuales se pueden realizar múltiples instrucciones sobre datos diferentes de forma simultánea.

Las MIMD son las más complejas, pero son también las que potencialmente ofrecen una mayor eficiencia en la ejecución concurrente o paralela.

Aquí la concurrencia implica que no sólo hay varios procesadores operando simultáneamente, sino que además hay varios programas (procesos) ejecutándose también al mismo tiempo.

La figura 5.1 muestra los esquemas de estos cuatro tipos de máquinas clasificadas por los flujos de instrucciones y datos.

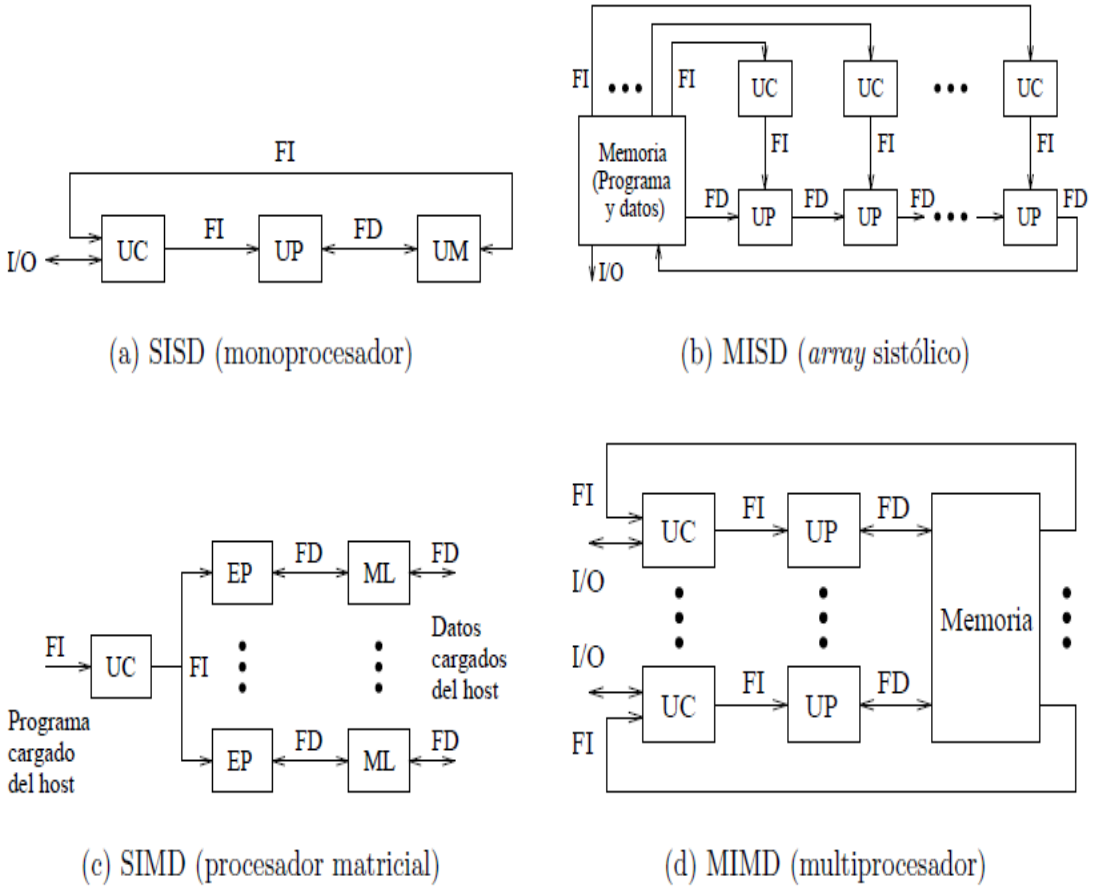


Figura 5.1: Clasificación de Flynn de las arquitecturas de computadores. (UC=Unidad de Control, UP=Unidad de Procesamiento, UM=Unidad de Memoria, EP=Elemento de Proceso, ML=Memoria Local, FI=Flujo de Instrucciones, FD=Flujo de datos.)

En la actualidad, algunas arquitecturas de procesadores cuentan con múltiples niveles de memoria caché. Por simplicidad y desempeño, algunos niveles se mantienen como privados para cada núcleo de un procesador, mientras que otras arquitecturas exploran compartir el último nivel de memoria caché entre diferentes núcleos del mismo procesador.

Los multiprocesadores se pueden dividir en dos grandes tipos:

- Multiprocesadores de Memoria Compartida (MMC).
- Multiprocesadores de Memoria Distribuida (MMD).

Indicando sus características generales, así como los tres tipos de multiprocesadores más representativos (Multiprocesador simétrico (SMP), Procesador Vectorial Paralelo (PVP) y Memoria Compartida Distribuida (DSM)), detallando para cada uno de ellos sus características y daremos un ejemplo de su uso en un supercomputador.

5.2 ¿Qué es un multiprocesador?

Se denomina multiprocesador a un sistema que cuenta con más de un procesador, funcionando de modo paralelo e independiente del resto, para la ejecución de una o varias tareas, bajo el control de un único sistema operativo.
En los cuales “varias unidades funcionales realizan diferentes operaciones sobre diferentes datos".

5.3 Tipos de Multiprocesadores

5.3.1 Multiprocesadores con Memoria Compartida (MMC).

En los MMC, la memoria se organiza en uno o varios módulos, compartidos por todos los procesadores a través de distintos tipos de interconexión (tratados más adelante), con un acceso constante. A este tipo de arquitectura se le conoce como memoria de acceso uniforme (UMA). El acceso a los módulos por parte de los procesadores se realiza en paralelo, pero cada módulo solamente puede atender una petición en cada instante de tiempo.

UMA (*Uniform Memory Access*) En un modelo de *Memoria de Acceso Uniforme*, la memoria física ésta uniformemente compartida por todos los procesadores. Esto quiere decir que todos los procesadores tienen el mismo tiempo de acceso a todas las palabras de memoria.

Cada procesador puede tener su cache privada, y los periféricos son también compartidos de alguna manera.
A estas computadoras se les suele llamar *sistemas fuertemente acoplados* dado el alto grado de compartición de los recursos.

Cuando todos los procesadores tienen el mismo acceso a todos los periféricos, el sistema se llama multiprocesador *simétrico*. En este caso, todos los procesadores tienen la misma capacidad para ejecutar programas, tal como el Kernel o las rutinas de servicio de I/O. En un multiprocesador *asimétrico*, sólo un subconjunto de los procesadores puede ejecutar programas.

A los que pueden, o al que puede ya que muchas veces es sólo un procesador que se le llama *maestro*. La Figura 5.1.2.1. Muestra el modelo UMA de un multiprocesador.

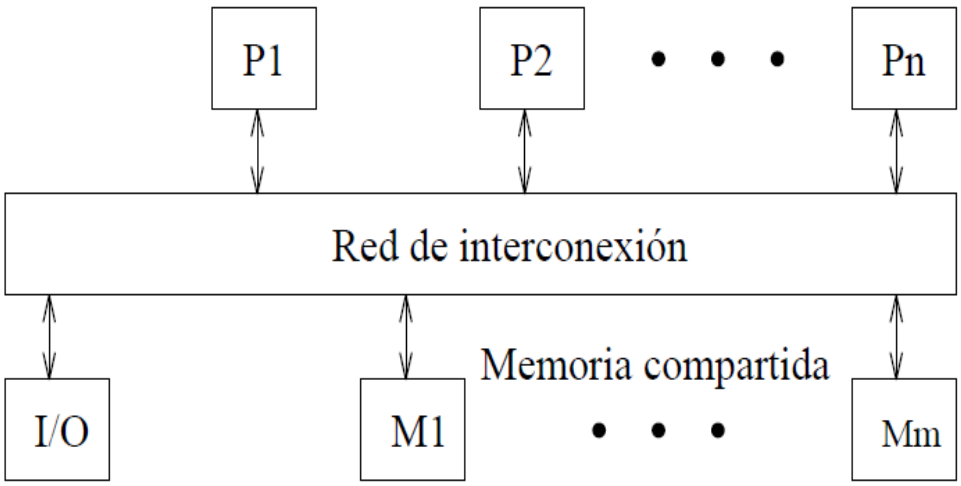


Figura. 5.1.2.1. El modelo UMA de multiprocesador.

5.3.2Multiprocesadores con Memoria Distribuida (MMD).

En los MMD, los multiprocesadores distribuyen la memoria de manera que dentro de cada procesador posee uno o varios modelos de memoria propia y está conectado mediante una red de interconexión al resto de procesadores.

De esta manera, cada procesador podrá acceder tanto a su memoria local, como a la memoria remota de cualquiera del resto de procesadores. Este tipo de arquitectura se denomina NUMA.

NUMA Un multiprocesador de tipo NUMA es un sistema de memoria compartida donde el tiempo de acceso varía según el lugar donde se encuentre localizado el acceso. La Figura. 5.1.2.2. Muestra una posible configuración de tipo NUMA, donde toda la memoria es compartida pero local a cada módulo procesador.

La ventaja de estos sistemas es que el acceso a la memoria local es más rápido que en los UMA aunque un acceso a memoria no local es más lento. Lo que se intenta es que la memoria utilizada por los procesos que ejecuta cada procesador, se encuentre en la memoria de dicho procesador para que los accesos sean lo más locales posible. Aparte de esto, se puede añadir al sistema una memoria de acceso global.

En este caso se dan tres posibles patrones de acceso. El más rápido es el acceso a memoria local. Le sigue el acceso a memoria global. El más lento es el acceso a la memoria del resto de módulos.

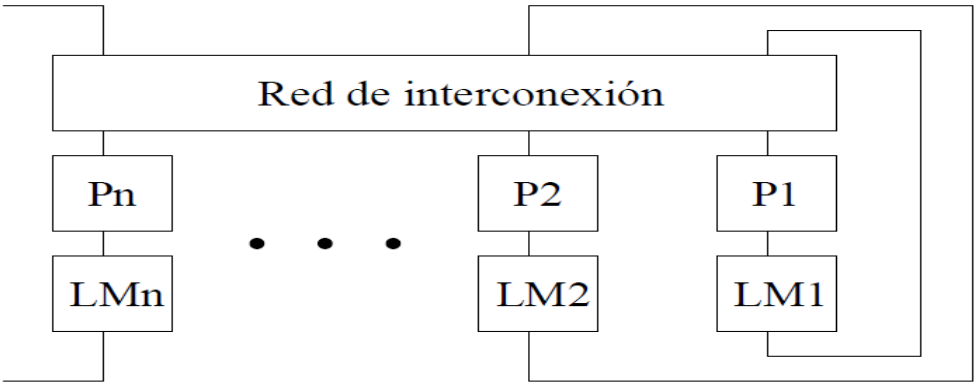


Figura. 5.1.2.2. El modelo NUMA de multiprocesador.

5.4 SMP (Multiprocesador simétrico).

Los sistemas SMP utilizan una modalidad de procesamiento en paralelo en la que “todos los procesadores son tratados como iguales". Los SMP están basados en el modelo de memoria de acceso uniforme, en donde la memoria física está uniformemente compartida por todos los procesadores, lo cual implica que todos ellos posean los mismos tiempos de acceso a todas las palabras de memoria.

Estos sistemas poseen una red de interconexión entre los distintos procesadores y la memoria, habitualmente en forma de bus, aunque también pueden tener otro tipo de sistema de interconexión. Cabe destacar que en los sistemas SMP todos los procesadores tienen el mismo acceso a los periféricos, lo cual implica que todos los procesadores tienen la misma capacidad para ejecutar programas tal como el Kernel o las rutinas de entrada y salida.

Otra característica importante es que todo el sistema está controlado por un mismo sistema operativo que posibilita la cooperación entre los procesadores y sus programas, debido a este alto nivel de cooperación se clasifican como sistemas fuertemente acoplados. La arquitectura de estos multiprocesadores sigue un esquema como el mostrado en la Figura 5.2.1

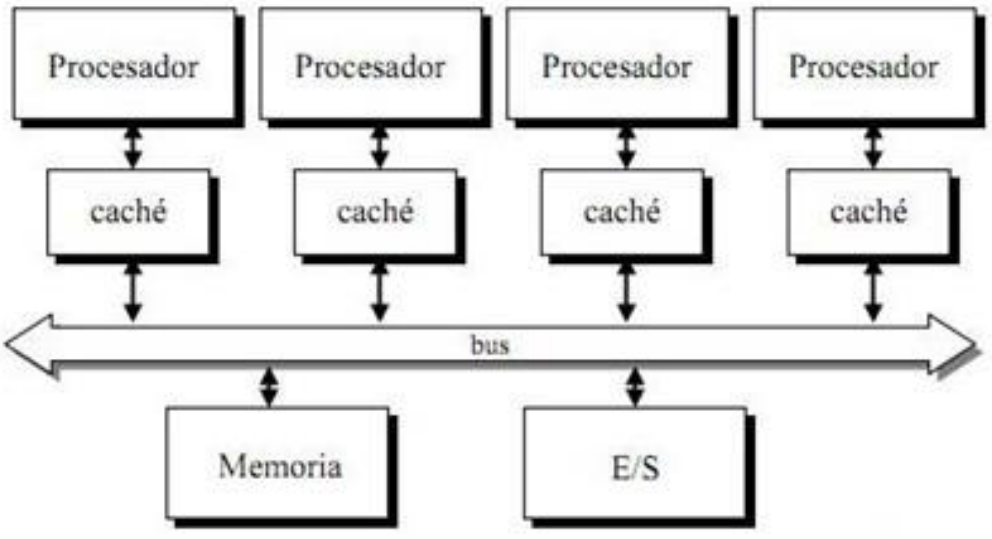


Figura 5.2.1 Arquitectura de una memoria SMP

El principal problema que se presenta en los SMP en cuanto a su arquitectura de memoria es la coherencia de caché, ya que los distintos procesadores pueden mantener su propia caché, por lo tanto si uno de ellos modifica un dato de la memoria compartida otros pueden mantener copias en su propia cache incoherentes.

Para solucionar este problema hay diferentes estrategias hardware y software, aunque este aspecto merecería un estudio en profundidad queda fuera del alcance de este documento.

Los SMP no sólo se encuentran en entornos de supercomputación, si no que los PCs con procesadores multinúcleo aplican esta arquitectura, tratando a cada núcleo como un procesador independiente.

5.4.1. Ejemplos

Existen actualmente muchos ejemplos de multiprocesadores simétricos de memoria compartida en el mercado. En la tabla 5.2.1, se muestran algunas de las características y precios de algunos de modelos reales del mercado:

Procesador	Núcleos	Frecuencia	Caché	Memoria	Precio	Año
SPARC T3	16	1.65 GHz	6 MB L2	128 GB DDR3	18600$	2010
Intel Xeon W3690	6	3.46 GHz	12 MB L3	24 GB DDR3	999$	2011
MIPS32 1074k	4	1.5 GHz	32+32KB L1	-	-	2010
ARM Cortex A15	4	2 GHz	-	-	-	-
AMD Opteron 6100	12	2.5GHz	12MB L3	-	1500$	2010

Cuadro 5.2.1. Tabla comparativa de modelos SMP en el mercado. Nótese que el precio del SPARC T3 incluye el rack con todos los accesorios, discos duros, memoria, etc.

5.4.1 Arquitectura de ejemplo: Supercomputador Cray Jaguar XT5-HE.

El Centro Nacional de Ciencias de la Computación (NCCS) del Laboratorio Nacional de Oak Ridge (Tennessee, Estados Unidos) posee uno de los supercomputadores incluidos en la lista Top500, en la que figuran los computadores de mayores capacidades del mundo.

Se trata del Cray Jaguar XT5-HE, que se puso en funcionamiento en 2009 y estuvo situado en el primer puesto de la lista Top500 durante el segundo semestre de 2009 y el primero de 2010.

Actualmente se encuentra en el segundo puesto de la lista. El Jaguar XT5-HE es un supercomputador que está compuesto por dos particiones, la XT4 y la XT5.

La partición XT5 se compone de 18688 núcleos de computación con 2 procesadores AMD Opteron de 6 núcleos (2,6GHz) cada uno y 16GB de memoria, lo que hace un total de 224256 cores, con una memoria total de 300TB (DDR2 800) y un rendimiento que se sitúa en los 2,3 petaflops.

La partición XT4 está formada por 7832 nodos de computación, cada uno de ellos con un procesador AMD Opteron quad core (2,1GHz) y 8GB de memoria, lo que hace un total de 31238 cores y 62TB de memoria RAM.

El Jaguar XT5-HE está disponible para su utilización (las dos particiones) en el campo de la investigación por parte de cualquier organización, previo registro en el NCCS y solicitud de uso. Además, el Oak Ridge National Laboratory ofrece visitas a sus instalaciones.

Procesador utilizado: **AMD Opteron** un ejemplo de sistemas multiprocesador de memoria compartida que siguen el esquema SMP son los procesadores de la serie Opteron de AMD. En la figura 5.2.2 se presenta, a modo de ejemplo, un esquema de la arquitectura de un procesador AMD Opteron Quad Core:

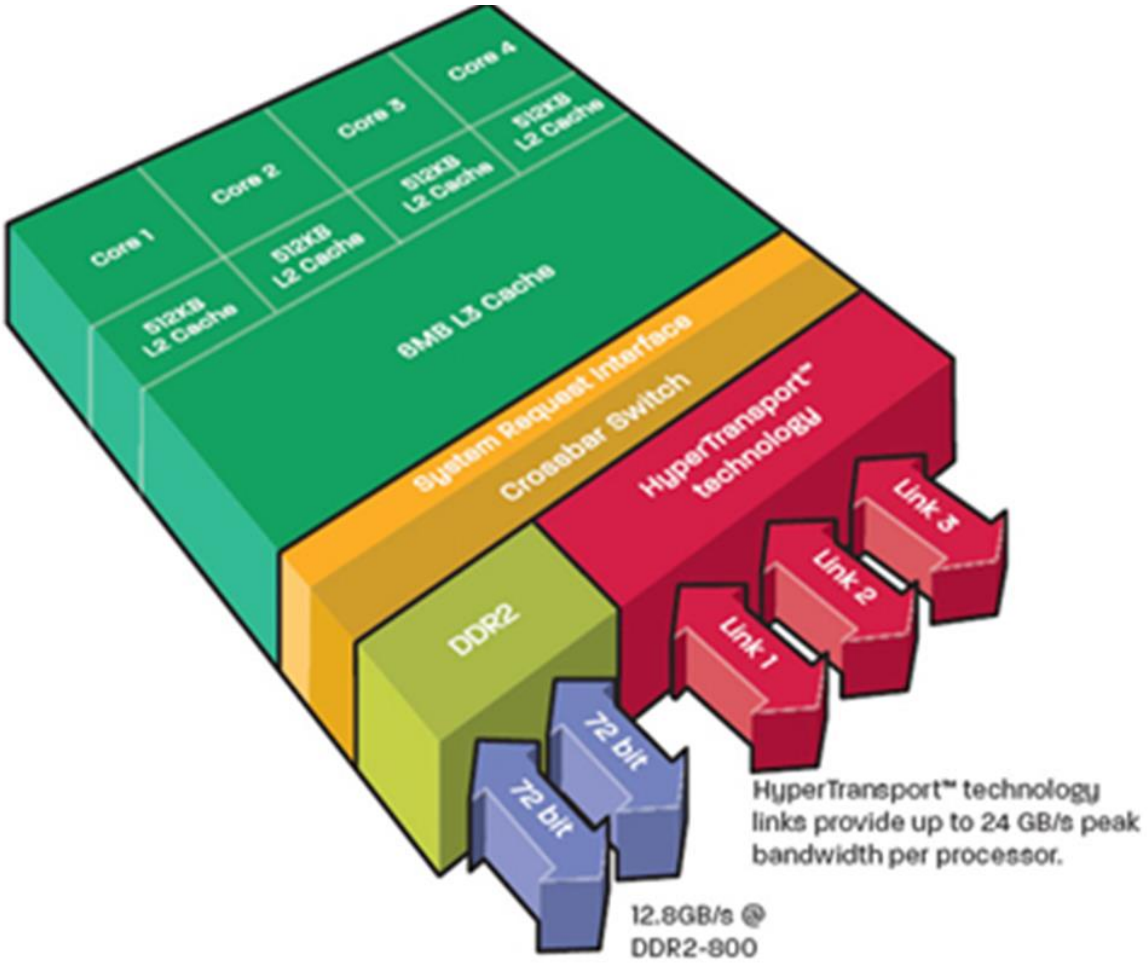


Figura 5.2.2. Arquitectura del procesador AMD Opteron Quad Core

Las características principales a nivel de arquitectura de estos procesadores son:

- 4 núcleos independientes (con frecuencia variable e independiente en cada núcleo).
- Caché L1 + L2 (512 KB) independientes en cada núcleo.
- Caché L3 de 6 MB compartida entre los 4 núcleos.
- Controladora de memoria DDR2 independiente del bus principal para un acceso más rápido a la memoria principal.
- Sistema Hyper Transport que sustituye a los buses antiguos y permite la conexión de hasta 3 dispositivos con un mayor ancho de banda (hasta 8GB/s).

Se puede observar que aunque la memoria caché de nivel 1 y 2 son independientes en cada procesador, la memoria caché de nivel 3 ya aparece como compartida, como una manera de reducir la latencia de la memoria compartida principal.

5.5 PVP: Procesador Vectorial Paralelo.

Los procesadores vectoriales cuentan con un micro-arquitectura orientado al procesamiento de vectores, además de un repertorio de instrucciones máquina que implementan operaciones en donde tanto los operandos como el resultado son vectores.

Esto permite que los procesadores vectoriales apliquen una instrucción sobre un conjunto de datos vector, en vez de aplicarla sobre un único dato, por lo tanto una única instrucción puede representar una importante carga de trabajo, lo que reduce el requisito de ancho de banda para instrucciones.

En procesadores no vectoriales la búsqueda y decodificación de instrucciones representa a menudo un cuello de botella, denominado cuello de botella de Flynn.

El paralelismo viene de la independencia de los elementos de las matrices entre sí, aunque esto no ocurre siempre si es habitual, por lo que las operaciones de unas matrices con otras pueden realizarse en paralelo, o al menos en el mismo cauce de instrucciones sin que se produzca un conflicto entre los datos.

No todas las tareas son susceptibles de este tipo de procesamiento, pero en general, esta arquitectura resulta especialmente indicada entre otros para problemas físicos y matemáticos, los cuales en su mayoría se pueden expresar fácilmente mediante matrices.

Las máquinas multiprocesadoras que utilizan procesadores vectoriales pueden ser vistas como un caso especial de las máquinas SMP, especialmente en cuanto al sistema de memoria compartida.

Esta combinación de procesadores vectoriales, de gran potencia y hechos a medida, con arquitecturas SMP, hacen que los sistemas PVP consigan grandes prestaciones para aplicaciones científicas y matemáticas.

Un problema de los sistemas PVP es la escalabilidad, debido a las colisiones en la red de interconexión, el número de procesadores que pueden componer el sistema está limitado. Como solución para poder escalar estos sistemas se suelen usar dos niveles, un nivel más profundo de multiprocesadores vectoriales, con memoria compartida, y un nivel más alto que representa una estructura de multicomputadores, lo cual implica memoria distribuida. (Gargollo & Lorenzo, Mayo 2011).

5.5.1. Ejemplo El IBM 3090.

El proceso vectorial de esta máquina es una opción llamada vector factible. En la Figura 5.5.1.1 se observa un diagrama de bloques. En el diseño de la arquitectura no se determina de forma concreta la longitud de registros vectoriales que se puede oscilar entre 8 y 512.

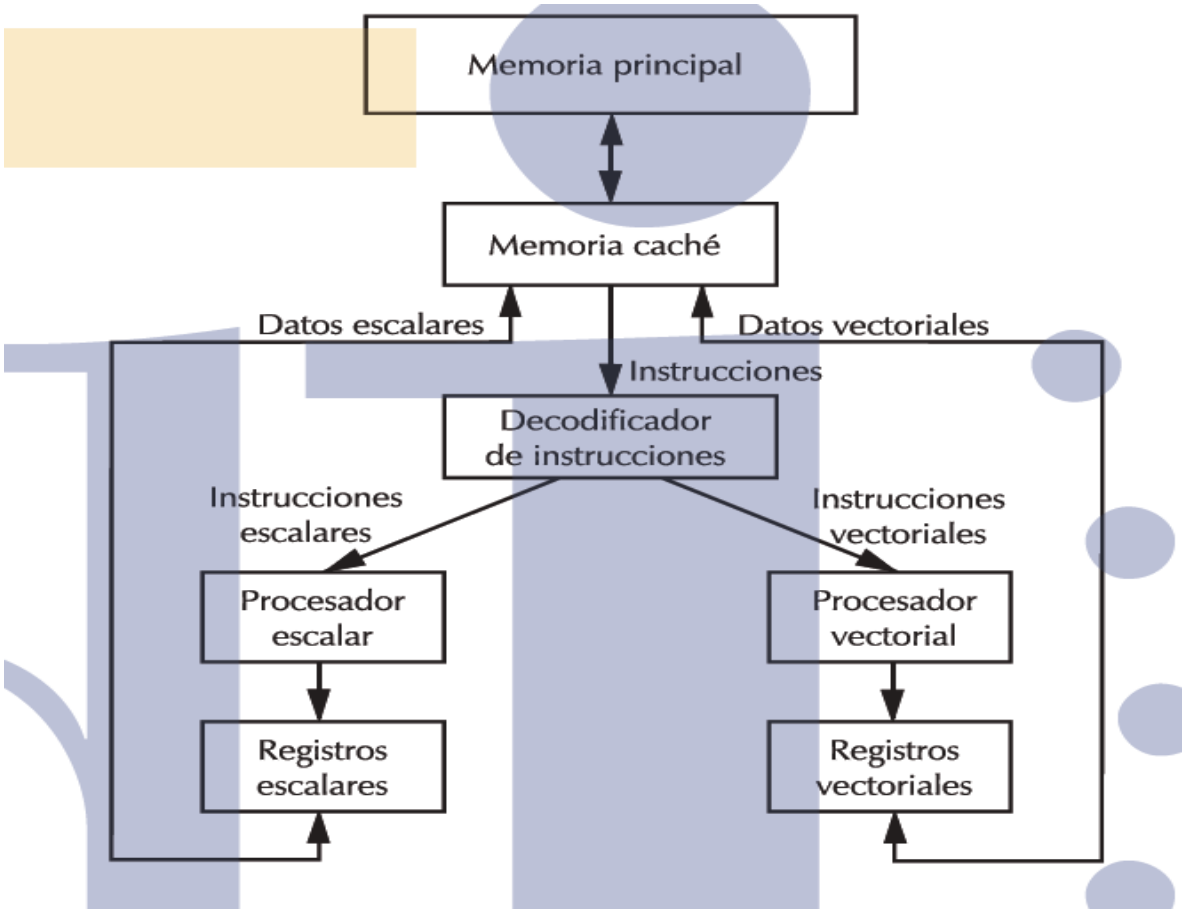


Figura 5.5.1. Diagrama de bloques del IBM-3090 con la opción de proceso vectorial.

5.5.2. Arquitectura de ejemplo: Earth Simulator

El Earth Simulador es un supercomputador de la Agencia Japonesa para la Tecnología y Ciencia Marina y Terrestre. Es un computador NEC SX-9/E/1280M160, que utiliza 5112 procesadores NEC a 3200Mhz organizados mediante la técnica NEC Vector de multiprocesamiento paralelo. El esquema de conexión de los distintos procesadores es el mostrado en la Figura 5.3.2.1.

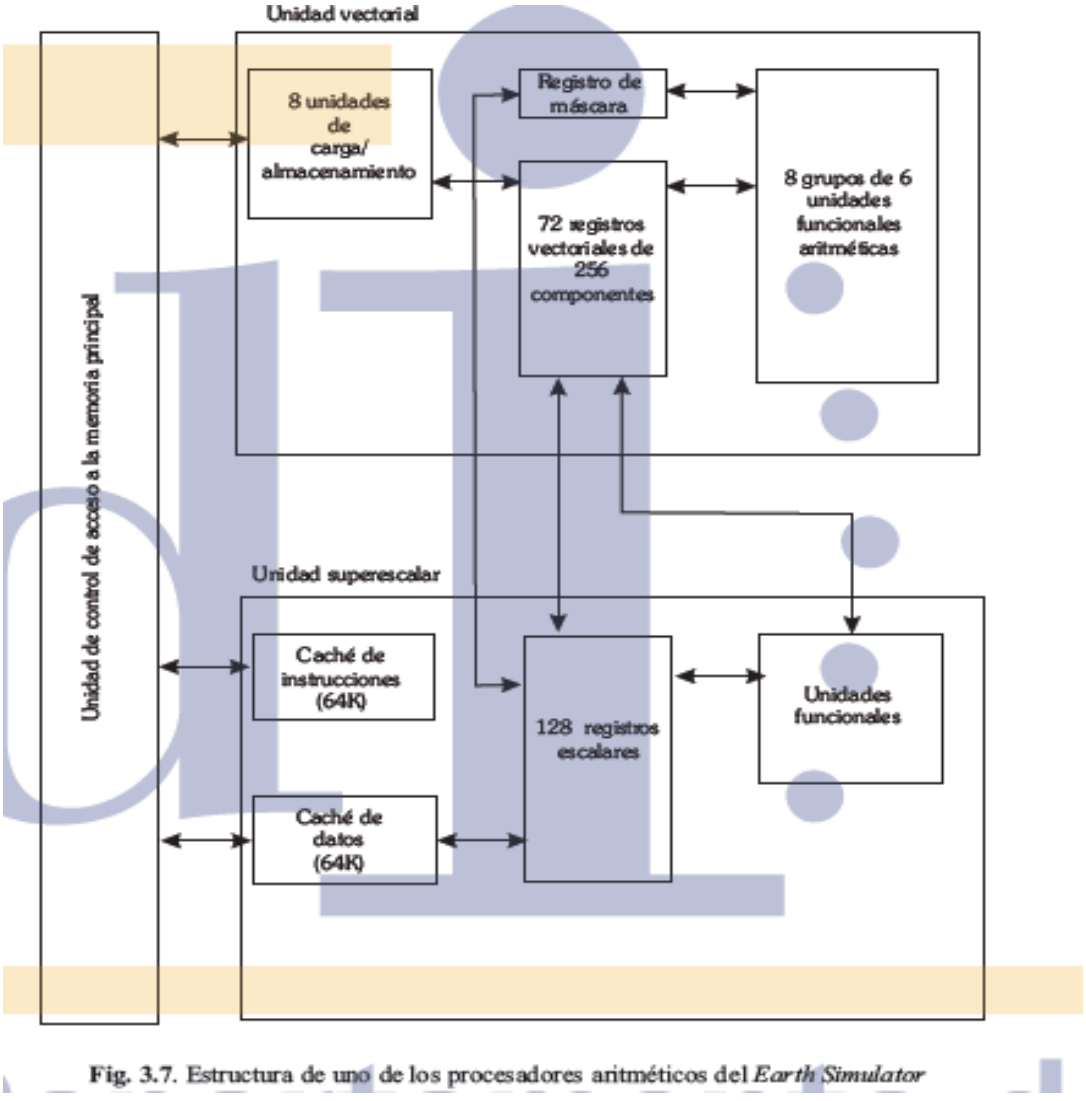


Figura 5.5.2. Arquitectura del supercomputador Earth Simulator.

Los procesadores se organizan en nodos de procesamiento, conteniendo cada uno 8 procesadores vectoriales que comparten 16Gb de memoria.
Los nodos de procesamiento están unidos mediante una red de interconexión conmutada cruzada.

Las características principales del computador, así como las tasas de rendimiento de los procesadores y los nodos son los mostrados en la tabla 5.3.2.2

Rendimiento de pico del procesador aritmético	8Gflops	**Número total de procesadores aritmétivos**	5120
Rendimiento de pico del nodo de procesamiento	64Gflops	**Número total de nodos de procesamiento**	640
Memoria compartida de cada nodo de procesamiento	16Gb	**Rendimiento de pico total**	40Tflops
		Memoria principal total	10TB

Tabla 5.3.2.2. Tabla de características del supercomputador Earth Simulator

El área de trabajo del supercomputador es la simulación del sistema terrestre completo mediante el procesamiento de modelos de datos de enorme tamaño sobre el presente y el pasado, para ayudar a predecir situaciones futuras en la tierra. Algunas aplicaciones realizadas son estudios sobre el cambio climático, el análisis de coexistencia entre el humano y el resto del medio ambiente y la simulación de condiciones atmosféricas. El supercomputador es utilizado tanto por el gobierno, como por distintas empresas que pagan por su tiempo de uso.

También dedica tiempo de uso para otros proyectos de investigación que deben ser anteriormente aprobados por la comisión del Earth Simulator. Estos últimos proyectos constituyen una gran parte del trabajo de la computadora, y se componen de varias decenas de nuevos proyectos anuales. (Gargollo & Lorenzo, Mayo 2011).

5.6 DSM: Memoria Compartida Distribuida.

Como hemos visto anteriormente, los sistemas de memoria compartida tradicional SMP, utilizan un mismo espacio de memoria compartido entre todos los procesadores. La comunicación entre la memoria y los procesadores generalmente se realiza mediante un bus, el cual puede llegar a suponer un cuello de botella en el acceso a memoria si el número de procesadores es suficientemente alto.

Las arquitecturas NUMA (Non-Uniform Memory Architecture) intentan aliviar este cuello de botella, acercando parte de la memoria a cada procesador, aunque esto deriva en que el acceso a la memoria remota es más lento que el acceso a la memoria local.

Desafortunadamente, tanto los sistemas SMP como NUMA tienen un precio elevado, lo que motiva el uso de los sistemas distribuidos, relativamente más baratos. No obstante, todas las comunicaciones y sincronizaciones deben hacerse mediante el paso de mensajes, ya que cada sistema tiene su propia memoria local, separada del resto.

En general resulta más fácil la programación para sistemas con un solo procesador o sistemas multiprocesadores con un bloque de memoria compartida, que mediante el paso de mensajes, de ahí el nacimiento de los sistemas DSM (Distributed Shared Memory), que no es más que una técnica para simular un espacio común de direcciones entre multicomputadores.

Existen varias alternativas para conseguir simular este espacio común de direcciones a partir de la arquitectura de memoria distribuida, por ejemplo mediante el uso de caches, más rápido y caro, mediante el uso de memoria virtual con modificaciones en el software, más lento y barato, o también mediante soluciones hibridas entre hardware y software.

Como es de esperar los sistemas DSM también plantean varios problemas que hay que tener en cuenta y para los cuales se proponen distintas soluciones, una vez más el problema de la coherencia es uno de los principales focos de conflicto, aunque su estudio queda fuera del alcance de este documento.

5.6.1 Arquitectura de ejemplo: Pleiades

El Pleiades es una supercomputadora de la División de Supercomputación Avanzada de la NASA el cual se encuentra situado en el campus de investigación Ames, en California. Pleiades, actualmente en la posición 11 del TOP500, soporta una variedad de proyectos científicos y de ingeniería, incluyendo modelos para evaluar la predicción del clima decenal para el Grupo Intergubernamental de Expertos sobre el Cambio Climático; el diseño de futuros vehículos espaciales; modelos detallados de los halos de materia oscura y la evolución de las galaxias.

Este supercomputador se trata de un SGI Altix ICE 8200EX, que utiliza 21504 procesadores Intel EM64T Xeon organizados en 10752 nodos de 2 procesadores por nodo.

De estos 10752, 3584 son nodos con procesadores Xeon X5670 (Westmere) de 6 núcleos y 1280 con procesadores Xeon X5570 (Nehalem) de 4 núcleos; en ambos casos trabajan a 2.93GHz. El esquema de los nodos \Westmere" es el mostrado en la Figura 5.4.2 (Para los \Nehalem" el esquema de conexión es el mismo pero con 4 cores).

Los procesadores se comunican mediante la Tecnología de Interconexión de Intel llamada QuickPath. Esta tecnología permite comunicar procesadores con controladores de memoria integrados.

Así mismo, los nodos se comunican entre sí utilizando para ello InfiniBand (bus de comunicaciones serie de alta velocidad, diseñado tanto para conexiones internas como externas) dado que todos los nodos están conectados siguiendo una topología de hipercubo de 11 dimensiones parciales.

Computador	Número de procesadores	Tamaño máximo de memoria compartida	Cache	Rendimiento de pico teórico del procesador	Rendimiento de pico teórico del computador
SGI ORIGIN 3800	16 a 512	2GB a 1TB	Cada procesador: 8MB cache L2, 32 KB cache L1 de datos, 32 KB cache L1 de instrucciones	800 Megaflops	400 Gigaflops
SGI Altix 4700	4864*2cores	39 TB	32 KB de cache L1 , 1 MB of L2 instruction cache, 256 KB de cache L2 de datos y 9 MB de cache L3 por cada núcleo	12.8 Gflop/s	62.3 TFlops
Cray X1[4]	4096	16GB por nodo	2MB "Ecache" compartidos entre cada cuatro procesadores	12.8 Gflop/s	-
ASURA[5]	1024 procesadores en 128 clusters	256Mb por nodo, 32GB en el sistema completo	Cache compartida, dividida en varios niveles jerárquicos (no especificados)	9.6 Gflop/s	-

Cuadro 5.4.1. Tabla comparativa de prestaciones de distintos supercomputadores DSM.(Gargollo & Lorenzo, Mayo 2011).

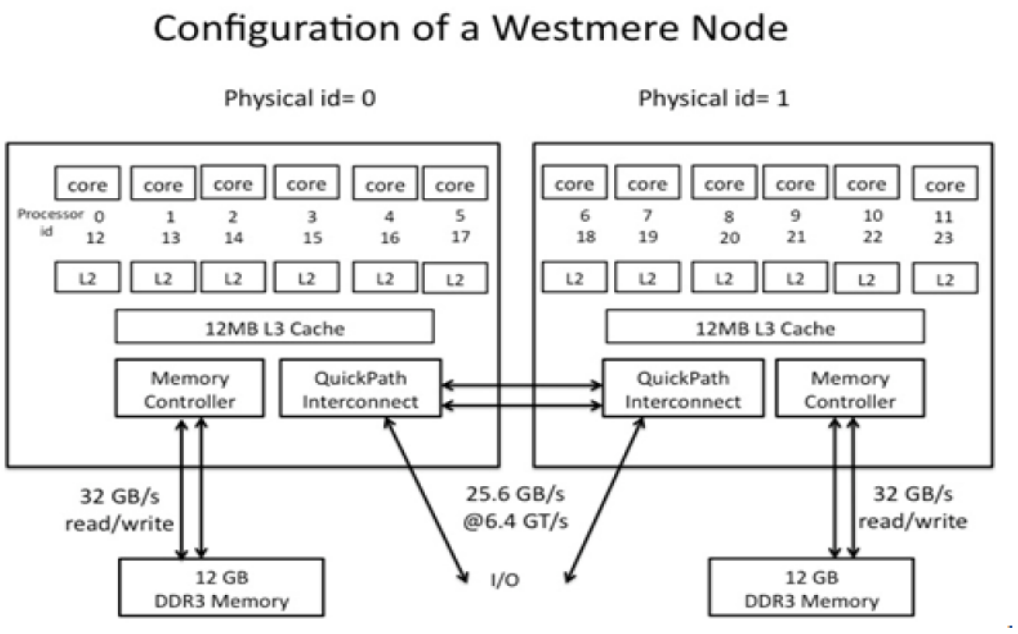


Figura 5.4.2. Arquitectura de Westmere.

En cuanto a las características principales del Pleiades, así como las tasas de rendimiento de los procesadores y los nodos son los mostrados en la tabla 5.4.3. (Gargollo & Lorenzo, Mayo 2011).

Numero de Racks (Bastidores)	168
Número Total de Nodos	10752
Número Total de Núcleos	100352
Memoria Principal	164 TB
Rendimiento máximo con High-Performance LINPACK	772.7 Tflops/s
Rendimiento máximo Teórico	973.291 Tflops/s

Cuadro 5.4.3. Tabla con las características de Pleiades.

Capítulo6. Otras Cachés.

6.1. Caché en Disco

¿Qué es la memoria caché de un disco y por qué es importante?.

Los expertos en tecnología suelen hablar de la "caché" cuando analizan las CPU, las unidades de disco y los sistemas informáticos. Pero, ¿qué es y por qué debería importarle?

La caché de un disco es RAM incorporada a su disco duro. Equilibra el flujo de datos entre la unidad de discos, que es relativamente lenta, y el resto del equipo al aceptar datos más de prisa de lo que pueden grabarse en el disco o leerse del mismo. Para ayudar a la caché a llevar a cabo su labor, el disco suele incluir una lógica para analizar el uso del disco y captar previamente los datos que probablemente necesitará pronto.

Mientras la caché no esté llena, la velocidad de transferencia de datos de la unidad de disco será mucho mayor que su velocidad real de lectura/grabación.

No confunda la caché con el tamaño de la unidad de disco ni con la RAM del sistema. La caché es una entidad diferenciada; de hecho, normalmente se trata de chips de memoria que están instalados en el receptáculo de la unidad de disco. A menudo, las especificaciones de las unidades de disco incluyen la cantidad de la caché montada.

Esta caché incorporada también es distinta del uso de una parte de la RAM del sistema como búfer de disco, aunque ocasionalmente también se hace referencia al mismo como "caché de disco".

La caché de disco es importante porque tiene una influencia en el rendimiento del sistema en aplicaciones de uso intensivo de datos, incluidos los juegos. Aunque generalmente pensamos en el rendimiento de disco en términos de velocidad de rotación y tiempo de búsqueda, la cantidad de caché que hay en el disco es, como mínimo, tan importante como su rendimiento. De hecho, se puede afirmar que en muchas aplicaciones, el tamaño de la caché es aún más importante que el tiempo de búsqueda. Esto es especialmente cierto en el caso de las aplicaciones que leen y grabar archivos muy grandes de manera secuencial.

La desventajas de la caché que solo dura hasta que se llena. Si el sistema intenta enviar o recibir datos más deprisa de lo que el disco puede leerlos o grabarlos, la caché acabará por llenarse y el rendimiento del disco bajará de manera significativa. Sin embargo, como la actividad de disco en los sistemas de sobremesa suele ser a ráfagas, es decir, se produce a ráfagas más que como una corriente continua de datos, en la mayor parte de los casos es posible poner suficiente memoria caché en un disco para adelantarse a las solicitudes de grabación.

El significado exacto de "la mayor parte de los casos" depende en gran medida de cuánta caché tiene el disco. Lo ideal es que la caché del disco sea tan grande que nunca se llene. Como la RAM es relativamente cara comparada con otros componentes de una unidad de disco (los precios varían en función del tipo de RAM que se utilice), esto no es muy práctico.

Existen grandes diferencias en el tamaño de la memoria caché de los discos. Hoy en día, el estándar es una memoria caché de 2 Mb como mínimo, aunque muchas unidades de "alto rendimiento" disponibles en las tiendas incluyen memorias caché de 8 Mb. Una unidad con 16 Mb producirá mejoras significativas en el rendimiento, pero son poco frecuentes y necesitan una unidad de disco más cara.

En igualdad de condiciones, el disco que tenga la mayor memoria caché le dará el mejor rendimiento. Pagará más, pero el sistema funcionará mejor. (Acronis,2004)

6.2Cache en servidores

6.2.1Domain Name System** o **DNS (en español: sistema de nombres de dominio) es un sistema de nomenclatura jerárquica para computadoras, servicios o cualquier recurso conectado a Internet o a una red privada. Este sistema asocia información variada con nombres de dominios asignado a cada uno de los participantes. Su función más importante, es traducir (resolver) nombres inteligibles para las personas en identificadores binarios asociados con los equipos conectados a la red, esto con el propósito de poder localizar y direccionar estos equipos mundialmente.

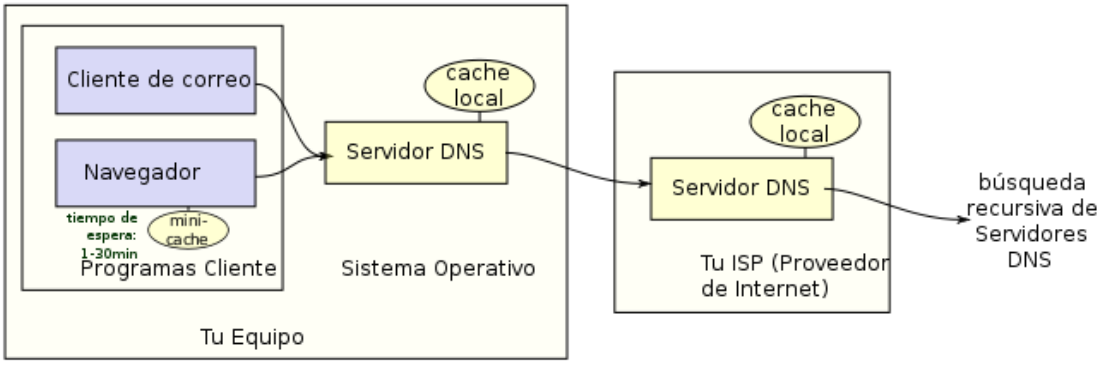
El servidor DNS utiliza una base de datos distribuida y jerárquica que almacena información asociada a nombres de dominio en redes como Internet. Aunque como base de datos el DNS es capaz de asociar diferentes tipos de información a

cada nombre, los usos más comunes son la asignación de nombres de dominio a direcciones IP y la localización de los servidores de correo electrónico de cada dominio.

La asignación de nombres a direcciones IP es ciertamente la función más conocida de los protocolos DNS. Por ejemplo, si la dirección IP del sitio FTP de prox.mx es 200.64.128.4, la mayoría de la gente llega a este equipo especificando ftp.prox.mx y no la dirección IP. Además de ser más fácil de recordar, el nombre es más fiable. La dirección numérica podría cambiar por muchas razones, sin que tenga que cambiar el nombre.

Los usuarios generalmente no se comunican directamente con el servidor DNS: la resolución de nombres se hace de forma transparente por las aplicaciones del cliente (por ejemplo, navegadores, clientes de correo y otras aplicaciones que usan Internet). Al realizar una petición que requiere una búsqueda de DNS, la petición se envía al servidor DNS local del sistema operativo. El sistema operativo, antes de establecer alguna comunicación, comprueba si la respuesta se encuentra en la memoria caché. En el caso de que no se encuentre, la petición se enviará a uno o más servidores DNS.

La mayoría de usuarios domésticos utilizan como servidor DNS el proporcionado por el proveedor de servicios de Internet. La dirección de estos servidores puede ser configurada de forma manual o automática mediante DHCP. En otros casos, los administradores de red tienen configurados sus propios servidores DNS.



En cualquier caso, los servidores DNS que reciben la petición, buscan en primer lugar si disponen de la respuesta en la memoria caché. Si es así, sirven la respuesta; en caso contrario, iniciarían la búsqueda de manera recursiva.

Una vez encontrada la respuesta, el servidor DNS guardará el resultado en su memoria caché para futuros usos y devuelve el resultado.

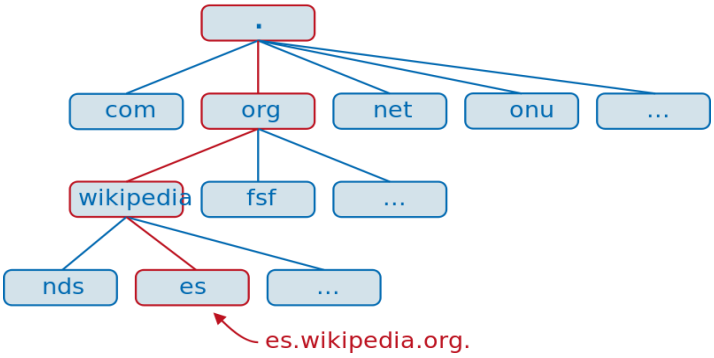
6.2.2Jerarquía DNS

El espacio de nombres de dominio tiene una estructura arborescente. Las hojas y los nodos del árbol se utilizan como etiquetas de los medios. Un nombre de dominio completo de un objeto consiste en la concatenación de todas las etiquetas de un camino. Las etiquetas son cadenas alfanuméricas (con '-' como único símbolo permitido), deben contar con al menos un carácter y un máximo de 63 caracteres de longitud, y deberá comenzar con una letra (y no con '-') (ver la RFC 1035, sección "2.3.1. Preferencia nombre de la sintaxis "). Las etiquetas individuales están separadas por puntos. Un nombre de dominio termina con un punto (aunque este último punto generalmente se omite, ya que es puramente formal). Un FQDN correcto (también llamado Fully Qualified Domain Name), es por ejemplo este: www.example.com. (Incluyendo el punto al final).

Un nombre de dominio debe incluir todos los puntos y tiene una longitud máxima de 255 caracteres.

Un nombre de dominio se escribe siempre de derecha a izquierda. El punto en el extremo derecho de un nombre de dominio separa la etiqueta de la raíz de la jerarquía (en inglés, root). Este primer nivel es también conocido como dominio de nivel superior (TLD - Top Level Domain).

Los objetos de un dominio DNS (por ejemplo, el nombre del equipo) se registran en un archivo de zona, ubicado en uno o más servidores de nombres.



6.2.3Tipos de servidores DNS

6.2.3.1 Primarios o maestros: Guardan los datos de un espacio de nombres en sus ficheros

6.2.3.2 Secundarios o esclavos: Obtienen los datos de los servidores primarios a través de una transferencia de zona.

6.2.3.3 Locales o caché: Funcionan con el mismo software, pero no contienen la base de datos para la resolución de nombres. Cuando se les realiza una consulta, estos a su vez consultan a los servidores DNS correspondientes, almacenando la respuesta en su base de datos para agilizar la repetición de estas peticiones en el futuro continuo o libre.

6.2.4 Consultas recursivas.

En las resoluciones recursivas, el servidor no tiene la información en sus caché, por lo que busca y se pone en contacto con un servidor DNS raíz, y en caso de ser necesario repite el mismo proceso básico (consultar a un servidor remoto y seguir a la siguiente referencia) hasta que obtiene la mejor respuesta a la pregunta. Cuando existe más de un servidor autoritario para una zona, Bind utiliza el menor valor en la métrica RTT para seleccionar el servidor. El RTT es una medida para determinar cuánto tarda un servidor en responder una consulta.

El proceso de resolución normal se da de la siguiente manera:

El servidor A recibe una consulta recursiva desde el cliente DNS.

El servidor A envía una consulta recursiva a B.

El servidor B refiere a A otro servidor de nombres, incluyendo a C.

El servidor A envía una consulta recursiva a C.

El servidor C refiere a A otro servidor de nombres, incluyendo a D.

El servidor A envía una consulta recursiva a D.

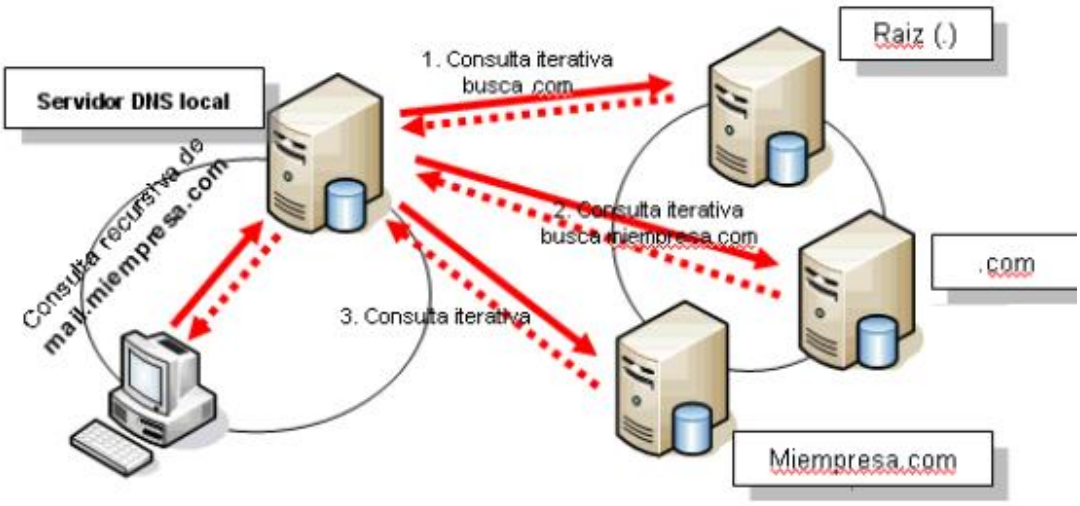
El servidor D responde.

El servidor A regresa la respuesta al resolver.

El resolver entrega la resolución al programa que solicitó la información.

6.2.5 Consultas iterativas.

Las resoluciones iterativas consisten en la respuesta completa que el servidor de nombres pueda dar. El servidor de nombres consulta sus datos locales (incluyendo su caché) buscando los datos solicitados. El servidor encargado de hacer la resolución realiza iterativamente preguntas a los diferentes DNS de la jerarquía asociada al nombre que se desea resolver, hasta descender en ella hasta la máquina que contiene la zona autoritativa para el nombre que se desea resolver.



6.2.6 Caché y TTL.

Los clientes y los servidores DNS se mantienen en memoria caché, si están configurados para ello, las respuestas a las preguntas que realizan a otros servidores. El tiempo que guardan las respuestas en caché se denomina TTL (Time to Live) y se define en los archivos de que zona de los servidores DNS preguntados.

Los clientes y servidores DNS almacenan en caché:

- Respuestas positivas. Registros de recursos de nombres resueltos.
- Respuestas negativas. Información de que no existen registros de recursos para un nombre consultado. Impiden la repetición de solicitudes adicionales para nombres que no existen.

En los ficheros de zona se define el tiempo que se guardarán en caché las respuestas positivas y el tiempo que se guardan en caché las respuestas negativas. Por lo tanto, se diferencia entre TTL de respuestas positivas y TTL de respuestas negativas.

La RFC 1912 recomienda que el valor TTL positivo sea de 1 día o mayor, y para los registros de recursos que cambien poco se usen valores de 2 o 3 semanas.

La RFC 2308 indica que el máximo valor del TTL negativo sea de 3 horas.

El administrador de un servidor DNS debe hacer un balance teniendo en cuenta la frecuencia con la que cambian los registros de recursos, la carga de los servidores DNS y el tráfico de red.

Los valores TTL pequeños ayudan a asegurar que la información acerca del dominio es más coherente en la red. En caso de que estos datos cambien a menudo, pero incrementan la carga en los servidores e incrementan el tráfico de red.

6.2.7 Recursividad y caché.

El recursivo es un servidor capaz de encontrar la respuesta a cualquier consulta DNS, y puede encontrar información acerca de casi todos los dominios, por ejemplo google.com, linux.org, gentoo.org .

Generalmente estos servidores están abiertos a cualquier IP, en otras palabras, pueden ser consultados por cualquier maquina en el mundo.

El problema es que muchas entidades hacen que el mismo servidor actúe tanto como autoritativo como recursivo. Una práctica sana es tenerlos separados, lo cual previene el problema denominado "envenenamiento de caché".

Si es imposible tenerlos separados, es recomendable que permitan recursión a solo un rango de IPs "confiables".

Posibles riesgos de tener un servidor recursivo en Internet .

Ser una víctima de ataques de envenenamiento de caché, la cual hace que el servidor afectado almacene información falsa.

Dicha información puede ser utilizada para comprometer la seguridad de los clientes que hacen consultas al servidor afectado, por ejemplo redireccionar google.com a un sitio con malware, o redireccionar sitios de bancos con la intención de obtener información confidencial del usuario.

El servidor podría ser ocupado para un ataque DoS distribuido el cual puede tener las siguientes consecuencias:

La gran cantidad de consultas DNS recibidas por el servidor y la gran cantidad de respuestas enviadas a la víctima pueden consumir una considerable cantidad de ancho de banda

Problemas legales ya que si, por ejemplo, un equipo de un ISP ataca a un cliente, seguramente el cliente lo demandará.

Bibliografías.

-Arquitectura de Computadoras Teoría y Ejercicios Resueltos, M.A de Miguel Cabello & T. Higuera Toledano, Editorial Computec ra-ma, Octubre 1997.

-Arquitectura de Computadoras, Bacalla Carlos, www.carlosbacalla.com/ cache.pdf, noviembre 2009.

-Arquitectura de Computadoras, Canto Q. Carlos, la memoria.ppt, Febrero 2004.

-Arquitecturas de Computadores, José García Rodríguez & José Antonio Serra Pérez, 2do Curso Ingeniería Técnica En Informática De Gestión Noviembre 2010.
-Electrónica C Memorias Semiconductoras, Raúl Rengel Estévez & María Jesús Martín Martínez, memorias semiconductoras.pdf, febrero 2011.

-Estudio, Análisis y Modelado de Memorias Caché Compartidas Bajo Administración Dinámica, Carballal Abarzúa Claudio, Tesis de Ingeniería en Universidad de Buenos Aires, Mayo2011.

-Elements of Computer Organizations, Gideon Langholz & Joan Francioni, Prentice Hall, Englewood Cliffs , New Jersey

-Memoria Caché, Departamento de Arquitectura de computadores, Universidad Nebrija, Noviembre 2007.

-Organización de Computadoras Un enfoque estructurado, Andrew S. Tanenbaum, Editorial Pearson Educación, Ámsterdam Países Bajos, 2000.

-Organización y Arquitectura de Computadores 7° edición, William Stallings, Editorial Prentice Hall, México 2007.

-Sistemas Multiprocesador de Memoria Compartida Comerciales, Florentino Eduardo Gargollo Acebrás, Pablo Lorenzo Fernández, Alejandro Alonso Pajares y Andrés Fernández Bermejo, Universidad de Oviedo, Asturias España, http://www.epigijon.uniovi.es/.

-Tipos de memoria, http://trestle.icarnegie.com/content/SSD/SSD2/4.4-Mx/normal/pg-hardware-sys/pg-proc-and-mem/pg-types-of-mem/pg-types-of-mem-ES.html, 2009.
The Turing Omnibus, 61 excursions in Computer Science, A. K . DEWDNEY, Computer Science Press, 1989.

Apéndice

A. Principio de localidad.

El principio de localidad establece que las direcciones que genera un programa durante su ejecución no son uniformes, sino que tienden a estar concentradas en pequeñas regiones del espacio de direcciones. La figura 5.1 muestra esta propiedad.Los programas tienden a reutilizar los datos e instrucciones que se han utilizado recientemente. En la mayoría de los programas, un pequeño porcentaje del código total es el responsable de un gran porcentaje del tiempo de ejecución. Es corriente que el 10% del programa ocupe el 90% del tiempo de ejecución. Este comportamiento es debido al flujo secuencial y a las estructuras de control de flujo (bucles), así como la agrupación de bloques, tanto de las instrucciones como de los datos dentro de los programas.

En la localidad de las referencias que genera un proceso coexisten tres componentes: tiempo, espacio y orden.

De esta forma, se puede hablar de tres tipos de localidad, cada una de las cuales viene marcada por una determinada tendencia:

Localidad temporal Tendencia del proceso a hacer referencia a elementos a los que se ha accedido recientemente. Si se hace referencia a un elemento, es muy probable que dicho elemento vuelva a ser referenciado pronto.

Localidad espacial Tendencia del proceso a efectuar referencias en la vecindad de la última referencia que se ha realizado. Si se hace referencia a un elemento, es probable que se realicen referencias a elementos próximos a él.

Localidad secuencial Tendencia del proceso a hacer referencia al siguiente elemento en secuencia al último referenciado. Si se hace referencia a un elemento, es muy probable que el próximo al que se haga referencia sea el siguiente en secuencia.

Una consecuencia del principio de localidad es que tomando como base el pasado reciente de un proceso se puede predecir, con una precisión razonable, qué instrucciones y datos utilizará dicho proceso en el futuro próximo. Esta característica se puede aprovechar implementando la memoria de un ordenador como una jerarquía de memoria.

B. Paginación.

Una de las principales técnicas que se utilizan en memoria virtual consiste en dividir, tanto el espacio de direcciones virtual como el espacio de direcciones real en bloques de igual tamaño.

Un bloque de un sistema paginado se denomina página y un fallo de memoria virtual se denomina falta de página. Las páginas se ubican en regiones fijas de memoria real llamadas marcos de página.

El tamaño de las páginas es normalmente una potencia de 2 y oscila entre 512 y 4096 palabras. Para ejecutar un programa en una maquina concreta, las direcciones virtuales deben estar ubicadas en el espacio de direcciones real de la memoria principal de dicha máquina. Hará falta una función para relacionar el espacio de direcciones de los programas y las posiciones de memoria real con que se cuenta.

La CPU produce una dirección virtual que debe ser traducida por una combinación de hardware y software en una dirección real para acceder a memoria principal. Cada dirección virtual consta de dos partes; un número de página y un desplazamiento que se utiliza para localizar la palabra dentro de la página.

Cada dirección real consta, a su vez, de un número de marco y un desplazamiento. La figura 5.3 muestra el formato de estas direcciones.

Figura 5.3: Dirección virtual y dirección real en un sistema paginado.

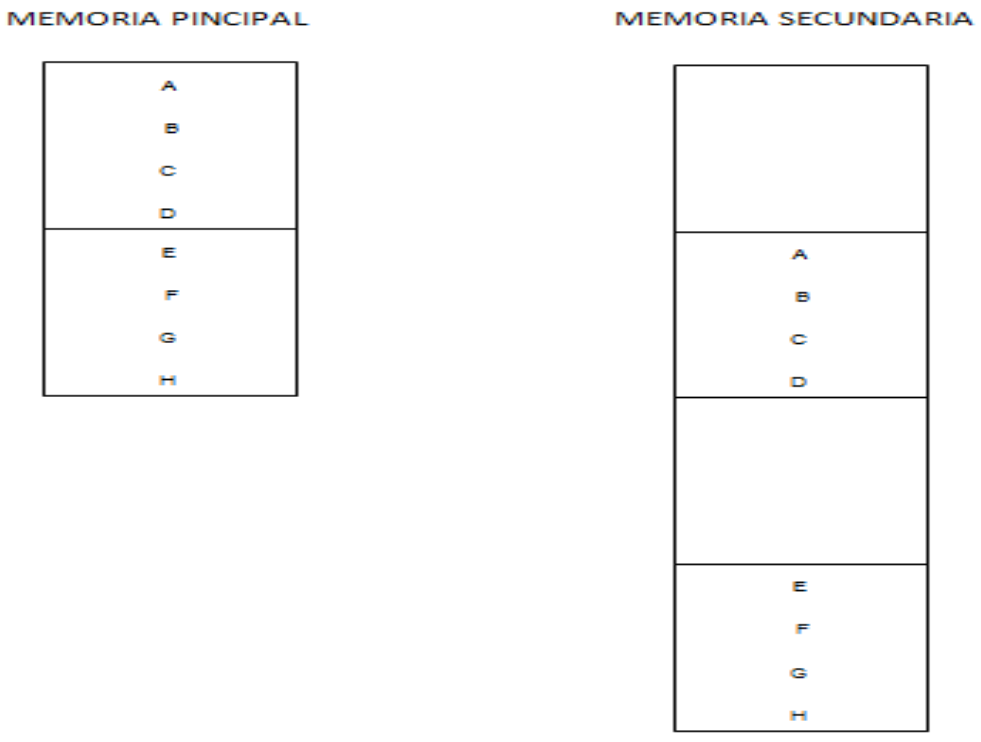
El número de página construye la parte superior de la dirección (bits de mayor peso) y determina el número de páginas direccionables.

El desplazamiento construye la parte inferior de la dirección (bits de menor peso) y determina el tamaño de la página. De igual manera, el número de marco construye la parte superior de la dirección real y determina el número de marcos direccionables de memoria real, mientras el desplazamiento, que no cambia, construye la parte inferior.

El número de páginas que se pueden direccionar con la dirección virtual puede ser mayor o igual que el número de marcos direccionables con la dirección real. En la traducción de una dirección virtual a una dirección real, se traduce el número de página en un número de marco y se concatena el desplazamiento.

La principal ventaja de la paginación es que la correspondencia de direcciones es muy sencilla, puesto que cualquier página puede ser asignada a cualquier marco de página que esté disponible.

Eso facilita la reubicación y simplifica el proceso de carga del programa en memoria principal.



C. Fallo de Página

Diagrama de Políticas de Sustitución.

Cuando se produce una falta de caché y hay que traer el bloque desde memoria principal, si no hay una línea libre para el bloque, habrá que sustituir alguno de los que están en la caché por el referenciado.

Si se utiliza correspondencia directa, no hay ninguna elección, hay que sustituir el bloque de la única línea en la que se puede ubicar el bloque referenciado. Si se trata de correspondencia asociativa de conjuntos, se puede elegir entre cualquiera de los bloques de conjunto que le corresponde al bloque referenciado.

Y si la función de correspondencia es la completamente asociativa, se puede elegir para sustituir cualquiera de los bloques que están en la caché. Así, tenemos que en los dos últimos tipos de correspondencias necesitamos una política de sustitución.

Esta cuestión es extremadamente importante, pues la política utilizada es un factor determinante para la tasa de aciertos del sistema.

Hay dos enfoques a la hora de elegir una política de sustitución: el que tiene en cuenta la estadística de utilización de los bloques (la historia de uso), y el que no lo tiene en cuenta. Entre los primeros se encuentran las políticas LRU y LFU, y como representantes del segundo enfoque esta la política random y la FIFO.

LRU (Least Recently Used). En general, el objetivo es mantener en la caché los bloques que tienen más probabilidades de ser accedidos en un futuro próximo, pero no resulta nada fácil saber esto.

No obstante, el principio de localidad nos dice que en ciertas áreas de memoria, y durante un periodo razonable de tiempo, hay una alta probabilidad de que los bloques que acaban de ser referenciados recientemente sean referenciados otra vez en un futuro próximo.

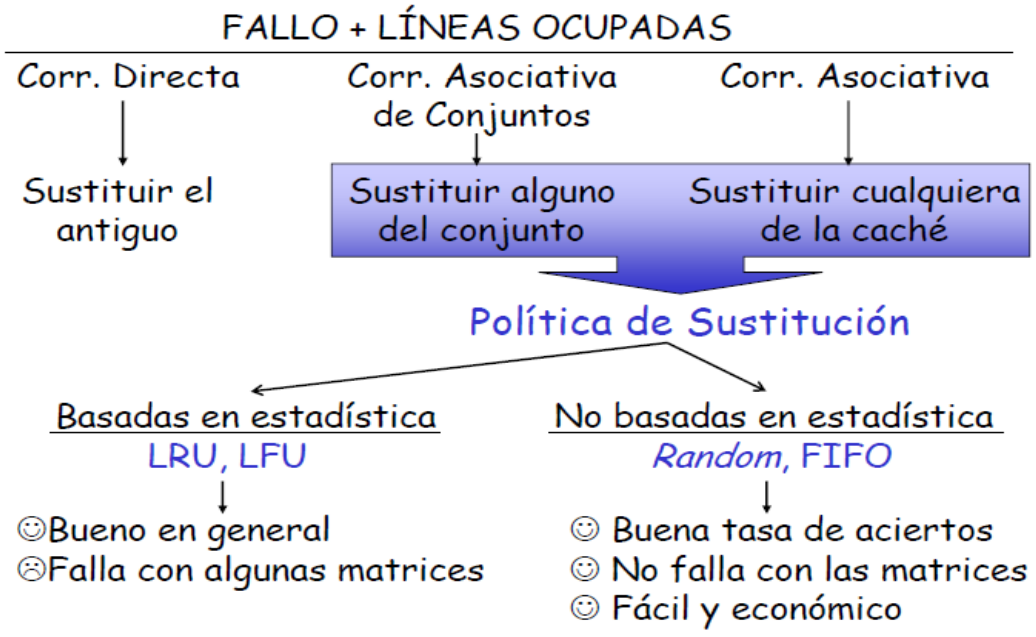
Por eso, cuando hay que seleccionar un bloque victima parece razonable elegir el que lleva más tiempo sin ser referenciado (el menos recientemente referenciados, the least recently used).

Para implementar este sistema se requiere añadir un contador a cada línea de la caché, de tal manera que cada vez que sea referencia un bloque su contador se pone a cero y los demás (del conjunto o de toda la memoria) se incrementan en uno. Cuando se trae un nuevo bloque, se elige como víctima el que tiene el contador más alto, y al nuevo bloque se le pone el contador cero.

Esta política de sustitución falla cuando se está accediendo de forma secuencial y cíclica a elementos de una matriz que no cabe completamente en la caché.
Una política parecida es la LFU (Least Frequently Used), en la cual sustituye el bloque menos referenciado. Esta también se implementa con ayuda de contadores. Como políticas no basadas en la estadística de uso, tenemos la FIFO (First In, First Out) que ofrece unos resultados ligeramente inferiores a la LRU, y que se implementa con ayuda de un buffer circular.
Los problemas que presenta se debe al hecho de que solamente tiene en cuenta el tiempo que lleva un bloque en la caché, y no cuales han sido las ultimas referencias; por esto también puede fallar con algunas matrices.

La política Random, que elige al azar el bloque a sustituir, ofrece una buena tasa de aciertos, no tiene el problema de la LRU con las matrices, y es facial y económico de implementar.

Diagrama de políticas de Escritura.



Ante una escritura...

Problema de la Coherencia de las Cachés

Políticas de Actualización

Escritura Inmediata (*write through*)	Escritura Diferida (*copy back*)
☺ No hay incoherencia	☹ Hay incoherencia
☹ Mucho tráfico	☺ Rápida, poco tráfico

Ante una escritura...

¡ Falta de Caché en Escritura ?

Dos Opciones

Escritura sin asignación	Escritura con asignación
Escribir directamente en la memoria	1º Traer el dato a caché 2º Escribir en la caché

Escritura diferida ⟶ Escritura con asignación

Escritura inmediata ⟶ Escritura sin asignación

D. Memoria Virtual

El termino memoria virtual hace referencia a sistemas en los que el usuario dispone de un espacio de direcciones virtual más grande que el número de posiciones de memoria real con que cuenta.

Esto es:

$$M \ll \sum Pi$$

Dónde:

M: Es el tamaño de memoria RAM.

P_i: Son los procesos que desean utilizar la CPU.

En las primeras implementaciones, el bus de direcciones de la CPU se conectaba directamente a la memoria principal y no tenía más que una única interpretación. Si un programa era muy grande y no cabía en memoria principal, era tarea del programación ajustarlo. El programador dividía los programas en partes o bloques, llamados recubrimientos (overlays).

Los recubrimientos se cargaban o descargaban en memoria durante la ejecución del programa. El mismo programa de usuario que se estaba ejecutando era el responsable de llevar los bloques a memoria a medida que se fueran necesitando, reemplazando los bloques que ya no fueran necesarios. Además, el programador debería asegurar que el programa no intentase acceder a un recubrimiento que no estuviese cargado y que los recubrimientos cargados no excedían del tamaño total de la memoria. Esta técnica crea gran carga al programador, que debe saber que partes de su programa necesitan estar residentes en memoria en un momento dado.

Aunque los recubrimientos se utilizaron durante muchos años, 1961 se propuso un método para efectuar de forma automática el proceso de los recubrimientos sin que el programador se diera cuenta de los que sucedía. La idea propuesta consistía en separar los conceptos de espacio de direcciones y posiciones de memoria.

Considerase el caso de un ordenador con 64 Kbytes de memoria y procesador MC68000; un programa que exceda de 65536 bytes (2^{16}) plantearía problemas al ejecutarle.

Sin embargo, un programa de ese ordenador puede direccionar 16 Mbytes (2^{24}) de memoria. La razón es que existen 16M direcciones de 24 bits.

El número de palabras direccionables depende sólo del número de bits de la dirección y no tiene ninguna relación con la cantidad de memoria con la que realmente se cuente para ejecutar el programa.

El conjunto de todas las direcciones que pueden ser referenciadas por un programa de usuario es lo que se llama espacio de direcciones virtual. Este espacio debe estar soportado en memoria secundaria.

En un sistema de memoria virtual se gestionan automáticamente los dos niveles de la jerarquía memoria principal- memoria secundaria, y se libera al usuario de la tediosa tarea de gestionar la transferencia de bloques de información, que pasa a ser responsabilidad del sistema operativo.

Además, los programas son independientes de la configuración y capacidad del sistema de memoria que se utilice para su ejecución, y se permite una compartición eficiente del espacio de memoria disponible entre varios usuarios en sistemas de tiempo compartido.

Para conseguir estos objetivos, se necesitan métodos de transferencia de información automáticos que hagan más eficiente la utilización del espacio de memoria disponible, y mecanismos de protección para evitar que los procesos alteren áreas de memoria que no deberían u otros posibles errores.

E. Memoria Asociativa vs Memoria de Acceso Aleatorio

MEMORIA ASOCIATIVA:

Una memoria asociativa se caracteriza por el hecho de que la posición de memoria a la que se desea acceder se realiza especificando su contenido o parte de él y no por su dirección. (García & Serra, Noviembre 2010).

POLÍTICAS UBICACIÓN “ASOCIATIVA”

- Permite que cada bloque de memoria principal pueda cargarse en cualquier línea de cache evitando la principal desventaja de la correspondencia directa.

- En este casi la lógica de control de la cache interpreta una dirección de memoria simplemente como una *etiqueta* y un *campo de palabra*.

Para determinar si el bloque está en cache, su lógica de control debe examinar simultáneamente todas las etiquetas de líneas para buscar una coincidencia. La etiqueta por tanto identifica unívocamente un bloque de memoria principal.

VENTAJAS:

Flexibilidad para que cualquier bloque pueda ser reemplazado por uno nuevo en la cache.

- Alta tasa de aciertos.

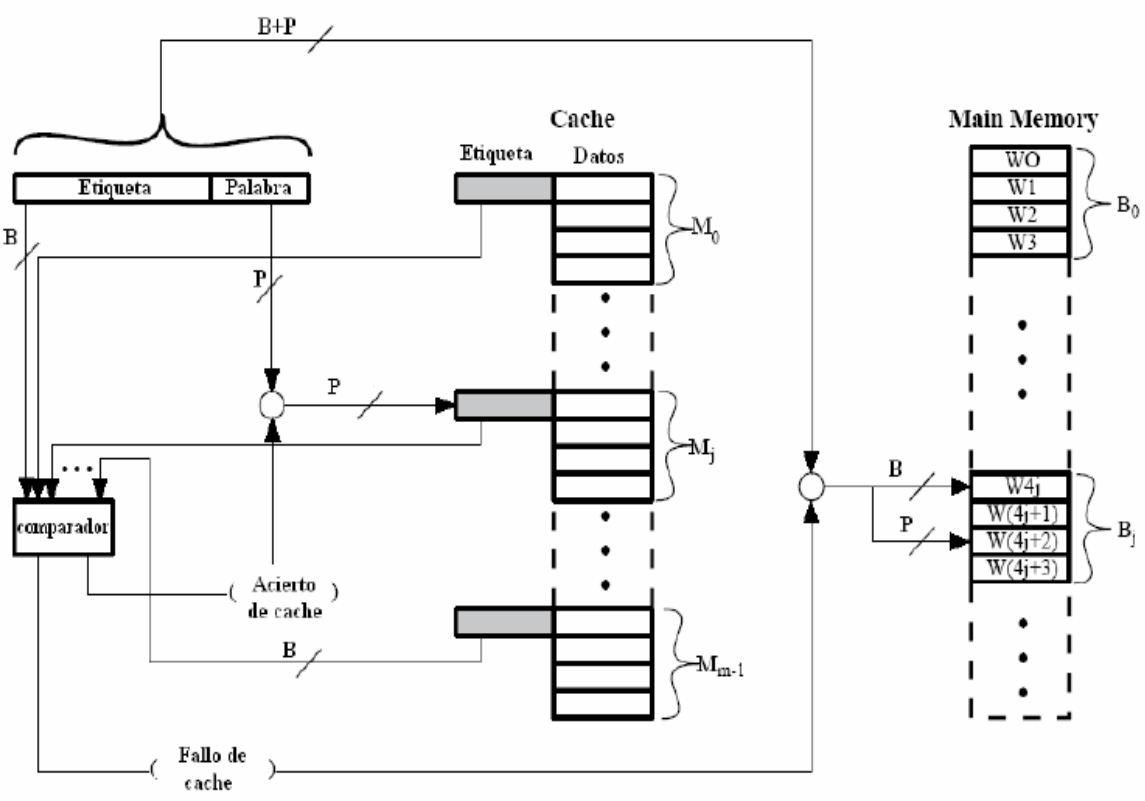
DESVENTAJAS:

Circuitería necesaria para examinar en paralelo las etiquetas de todas las líneas de caché es muy compleja.

- Proceso de identificación lento

- Alto coste

ORGANIZACIÓN DE CACHE ASOCIATIVA



(Universidad Nebrija, Noviembre 2007).

MEMORIA DE ACCESO ALEATORIO Random-Access-Memory
Memorias de acceso aleatorio, directo o selectivo
Su tiempo de acceso es independiente de la posición

(Rengel & Martin, Febrero 2011).

Acceso aleatorio (RAM). El tiempo de escritura/lectura es independiente de la localización de la información dentro de la memoria.
Dentro de este tipo de memorias podemos hacer la siguiente clasificación, basándonos en la permanencia de la información en ellas:

1. Memorias activas. Los tiempos de escritura/lectura (R/W) son del mismo orden. Según como se realice la operación de lectura/escritura tenemos:

a) Lectura/escritura no simultánea. En estas memorias sólo se puede seleccionar una posición de memoria simultáneamente y realizar en ella la operación de lectura o escritura.

b) Lectura/escritura simultánea. Se pueden realizar ambas operaciones a la vez, pero sobre posiciones de memoria distintas.

2. Memorias pasivas. En éstas los tiempos de escritura y lectura difieren considerablemente, siendo generalmente mucho mayor el de escritura. Los tipos más importantes son:

a) Totalmente pasivas (ROM). La escritura se realiza en el momento de la fabricación, siendo imposible escribir sobre ellas después (tiempo de escritura infinito).

b) Pasivas programables (PROM). El proceso de escritura sólo puede realizarse a través de aparatos especiales y es mucho mayor que el de lectura. Una vez almacenada la información, basta con lanzar un impulso eléctrico para que la información quede grabada para siempre.

c) Pasivas reprogramables (RPROM). Son iguales que las PROM, salvo que el método de escritura se puede usar para borrar y volver a escribir. El número de veces que se puede realizar esta operación no es ilimitado.