MEMORIAS

Memorias

Localización

- CPU
- Interna
- Externa

Capacidad

- Tamaño de palabra
- Cantidad de bytes

MEMORIA VIRTUAL

- ✓ Analogía con el espacio en un armario. Cuanto mas tenemos mas necesitamos
- ✓ Una forma económica es aumentar ese espacio con memoria en disco que tiene menos costo por bit.
- ✓ Nada es gratis, a mayor memoria virtual mayor posibilidad degradar el sistema en tiempo de respuesta

Proyecciones de Moore

- La velocidad de procesamiento de las Pc se ha **duplicado cada 18 meses** manteniendo el precio
- Las memorias **cuadriplican su tamaño cada 26 meses** sin variar su precio
- La velocidad de las memorias aumenta el 10% anual

Memorias
FORMAS DE ESCRITURA

BIG-ENDIAN: El bit más significativo en la dirección mas baja

LITTLE-ENDIAN: El bit más significativo en la dirección mas alta

Little-endian: INTEL

Big-Endian: MOTOROLA, MAINFRAMES IBM, RISC

Se debe tener en cuenta en palabras mayores de 1 byte

Memorias

Unidades de transferencia

- Interna
  - Usualmente gobernada por el tamaño del bus de datos
- Externa
  - Usualmente por bloques mucho mayores que un carácter
- Unidad de dirección
  - Menor locación que puede ser unívocamente asignada

METODOS DE ACCESO A MEMORIA

- **Secuencial**
  - Comienza por el principio y lee en orden
  - El tiempo de acceso dependen de la locación a leer y la previa
    - e.j. Cinta magnética
- **Directa**
  - Acceso saltando a las cercanías y haciendo una búsqueda secuencial
  - El tiempo de acceso dependen de la locación a leer y la previa
    - e.j. Disco
- **Aleatoria**
  - Se identifican las locaciones individuales con exactitud
  - El tiempo de acceso es independiente de la locación a leer y la previa
    - e.j. RAM
- **Asociativa**
  - Los datos se localizan por comparación con el contenido de una parte de lo almacenado
  - El tiempo de acceso es independiente de la locación previamente accesada
    - e.j. Cache

Estructura jerárquica de memorias

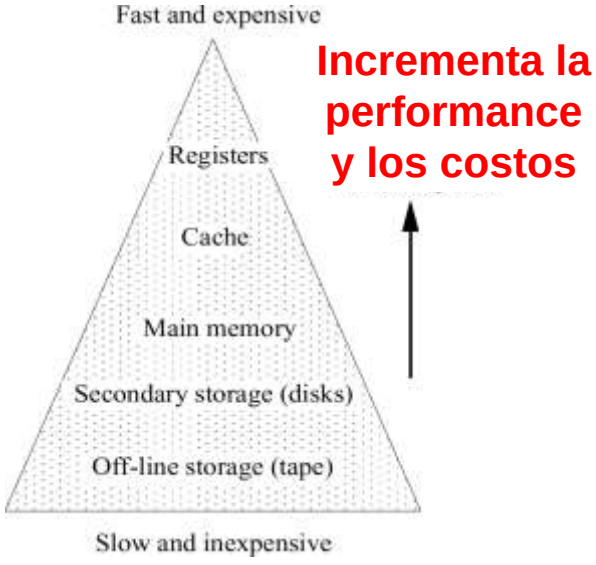
Una estructura jerárquica es una memoria que tenga diferentes niveles, con diferentes velocidades por nivel y diferentes tamaños

Los datos se copian entre niveles adyacentes por vez

Memorias - Jerarquías

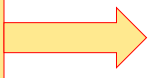
- Registros
  - En la CPU
- Interna o memoria principal
  - Puede incluir uno o mas niveles de cache
  - RAM
- Externa
  - Almacenamientos masivos

Memorias - Jerarquías



Incrementa la performance y los costos

¿QUIÉN SE OCUPA DE LA COPIA ENTRE MEMORIAS?



- **LA CPU**
- **EL SO**

LA COPIA SE DA SIEMPRE ENTRE CAPAS ADYACENTES

Memorias - Performance

- Tiempo de acceso
  - Tiempo entre que se presenta la dirección y se obtiene el dato valido
- Tiempo del ciclo de memoria
  - Tiempo que le toma a la memoria recuperarse antes del próximo acceso

El tiempo del ciclo es de acceso + recuperación

- Tasa de transferencia
  - Tasa a la que se pueden mover los datos

PROPIEDADES DE LAS DISTINTAS JERARQUIAS DE MEMORIA

TIPO DE MEMORIA	TIEMPO DE ACCESO	COSTO POR MBYTE	TAMAÑO TIPICO UTILIZADO	COSTO APROXIMADO
REGISTROS	1 ns	ALTO	1 Kb	-
CACHE	5 – 20 ns	80	1 Mb	80
MEMORIA PRINCIPAL	60 – 80 ns	0.7	1 Gb	70
DISCOS	10 ms	0.01	100 Gb	100

MEMORIAS INTERNAS

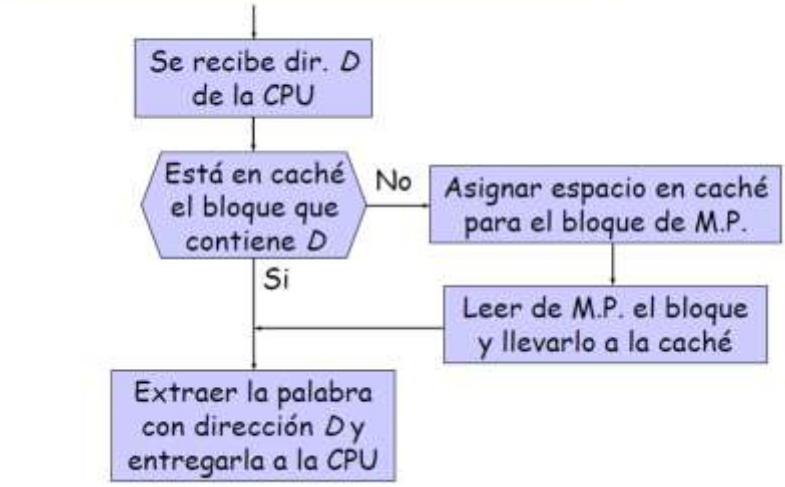
Cache

- Pequeña cantidad de memoria muy rápida
- Colocada entre la memoria principal y la CPU
- Habitualmente colocada dentro del chip del CPU

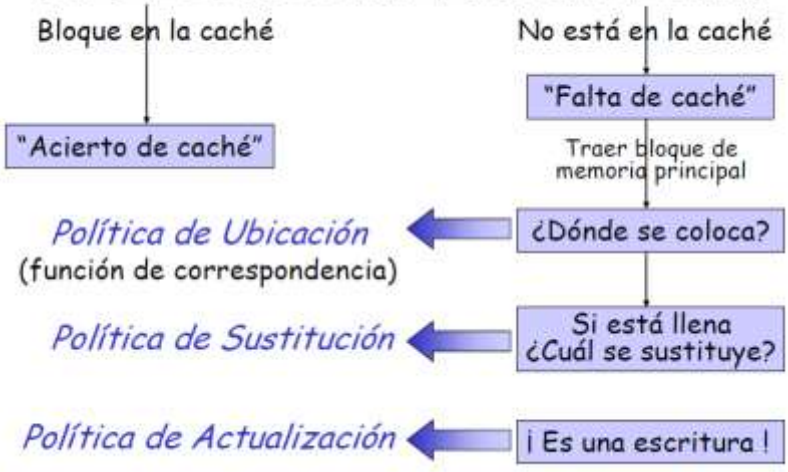
➢ FUNCIONAMIENTO
  - CPU requiere el contenido de una dirección de memoria
  - Verifica la existencia de este dato en el Cache
  - Si esta presente, la carga desde allí (rapidez)
  - Si no esta presente, la lee desde la memoria principal y la carga en el cache
  - La envía desde allí a la CPU
  - En el Cache se incluyen tags (ETIQUETAS) para identificar el bloque de la memoria principal desde donde fue tomado el dato

- Durante la ejecución primero se verifica si está en el cache, y si no, se recurre a la memoria principal
- Los sistema actuales tienen varios niveles de cache: L1, L2 y hasta L3
- L1 y L2 esta incorporado en el circuito integrado de la CPU
- La cache es mucho más rápida y más cara

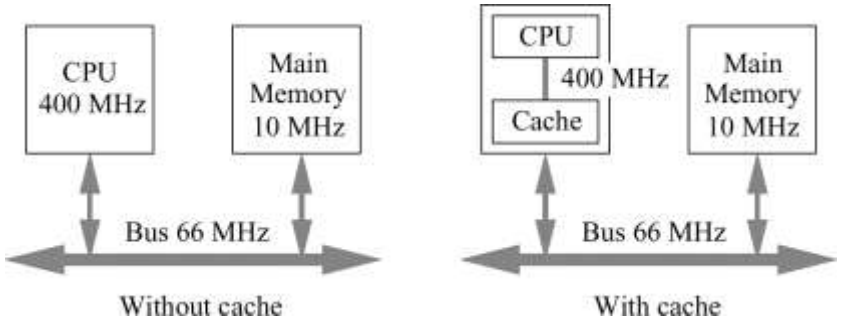
Operación de lectura en un sistema con caché



CPU → LECTURA / ESCRITURA



CPU con o sin cache



PRINCIPIO DE LOCALIDAD

En un programa, la mayor parte de las referencias de memoria se hacen con respecto a una pequeña cantidad de direcciones.

Cuando un programa hace referencia a un lugar en memoria, normalmente accede a ella en corto plazo: **LOCALIDAD TEMPORAL.**

De igual manera hay una **LOCALIDAD ESPACIAL,** los programas consumen mucho tiempo en interacciones dando vueltas sobre el mismo punto.

Los accesos a la memoria son mucho más lentos que la velocidad de procesamiento de las instrucciones, lo cual genera un **CUELLO DE BOTELLA**

PRINCIPIO DE LOCALIDAD

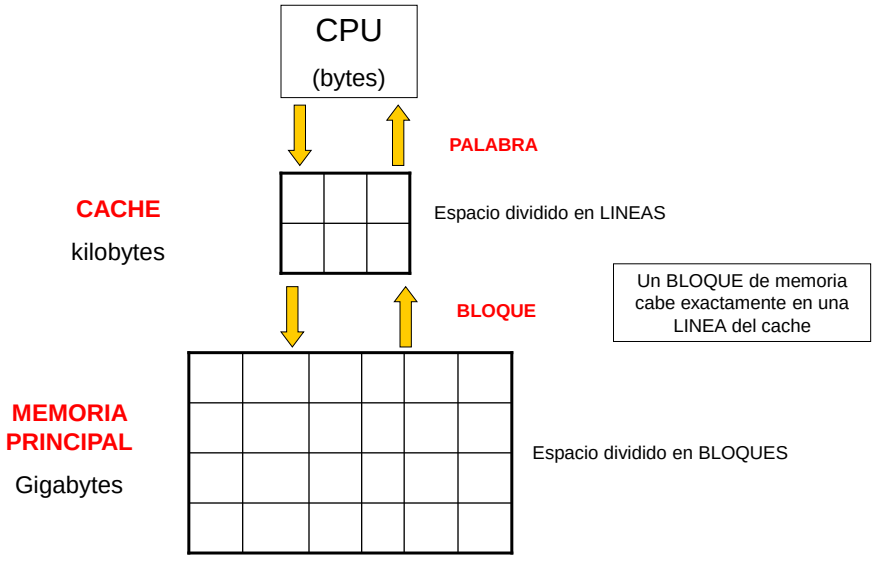
LOCALIDAD TEMPORAL

Probabilidad de repetir instrucciones (BUCLES)

LOCALIDAD ESPACIAL

Probabilidad de repetir instrucciones cercanas

Unidad de transferencia **BLOQUES**



EFECTIVIDAD DE LA CACHE

Tiempo medio de acceso a memoria

$T_{acceso} = T_C \cdot P_A + T_{MP} \cdot (1 - P_a)$

P_A : Probabilidad de acierto

T_C : Tiempo de acceso a caché

T_{MP} : Tiempo de acceso a M. P.

EJEMPLO

$T_{MP} = 500\ \eta s$

$T_C = 50\ \eta s$

$P_A = 0{,}99$

$T_{acceso} = 50 \cdot 0{,}99 + 500 \cdot 0{,}01 = 54{,}5\ \eta s$

¿ Merece la pena la caché ?

$$\text{Índice de mejora} = \frac{T_{sin\ caché}}{T_{con\ caché}} = \frac{500}{54{,}4} = 9{,}19$$

Organización típica de un cache

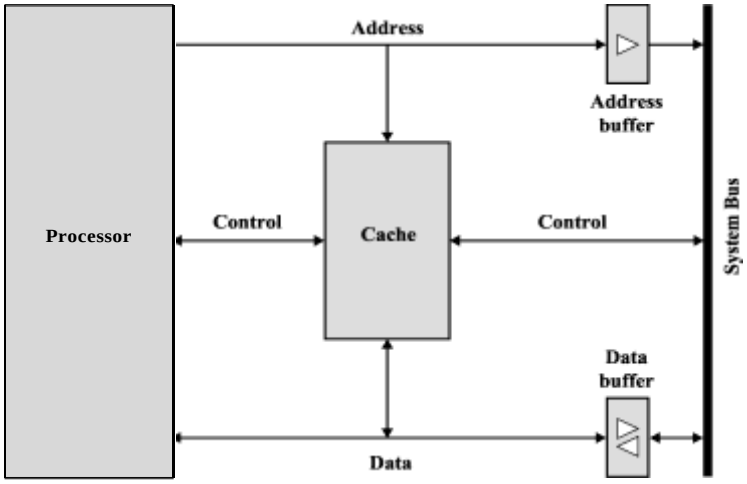


Figure 4.16 Typical Cache Organization

MEMORIA CACHE DE ASIGNACION DIRECTA

✓La memoria se divide en 2^{27} bloques de $2^5 = 32$ palabras por bloque

✓La memoria cache consiste en 2^{14} líneas es decir que $2^{27}/2^{14} : 2^{13}$ bloques de memoria principal a cada línea de memoria cache.

✓Para mantener el control de cual de los 2^{13} bloques se encuentran en cada línea, se agrega un campo de etiqueta de 13 bits

✓Cada línea de memoria cache se corresponde con un conjunto explicito de bloques de memoria principal.

✓Cada línea puede recibir mas de un bloque por lo que se le agregan 14 bits a la etiqueta para definir la línea

✓Es simple de implementar pero puede conducir a errores

Etiqueta	Línea	Palabra
13 bits	14 bits	5 bits

PUEDE PRESENTAR PROBLEMAS DE COLISIONES

MEMORIA CACHE DE ASIGNACION DIRECTA

Es una estructura de cache en el que cada locación de memoria es mapeada a exactamente una locación en el cache

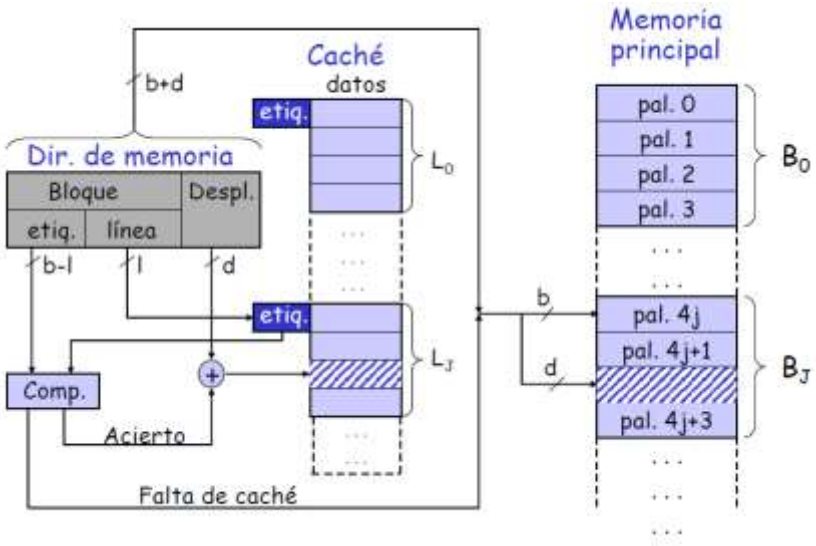
Como las diferencias de tamaño son muy grandes utiliza los bits menos significativos.

Ej 00001_2 y 11101_2 van a la locación 01_2

Como cada locación de memoria cache puede contener un numero diferentes de locaciones de la memoria principal, agrega etiquetas (TAGS) a cada palabra compuestas por los bits mas significativos de la dirección.

En el cache puede haber info valida, o estar vacío, por lo que se coloca una FLAG para indicar si una locación tiene información valida o no. Por ejemplo cuando la palabra en la dirección 18 ($100\mathbf{10}_2$) se coloca en el bloque 2 (010_2) del cache, la palabra de la dirección 26 ($110\mathbf{10}_2$) debe ser reemplazada por el nuevo dato.

MEMORIA CACHE DE ASIGNACION DIRECTA



MEMORIA CACHE DE ASIGNACION DIRECTA

Este algoritmo se implementa fácilmente a partir de las direcciones que genera la CPU. En lo que concierne a la caché, cada dirección de memoria consta de tres campos:

Desplazamiento: Los *d* bits menos significativos identifican una única celda de memoria dentro de un bloque de memoria principal. Es decir, es el desplazamiento de la celda dentro de su bloque.

Los restantes *b* bits de la dirección indican uno de los 2^b bloques de memoria principal. Como todos los bloques de memoria principal no caben en las *L* líneas de la caché, ésta interpreta estos *b* bits como una composición de dos campos: la línea y la etiqueta.

Línea: Este campo indica la línea en la que debe ubicarse o localizarse un bloque de memoria principal. Está formado por los *l* bits menos significativos de los *b* bits de mayor peso de la dirección, e indica una de las *L* líneas de la caché, pues $2^l = L$.

Pero claro, va a haber muchos bloques a los que les corresponda la misma línea de caché. Concretamente, los restantes *b-l* bits, los de mayor peso de la dirección, indican a cuántos bloques les corresponde la misma línea en la caché.

En la figura de la diapositiva anterior se puede ver que en una caché con *L* líneas, a la línea *0* le corresponden los bloques número *0*, *L*, *2L*, *3L*, ...; a la línea *1* le corresponden los bloques *1*, *L+1*, *2L+1*, *3L+1*, ...; y a la línea *L-1* le corresponden los bloques *L-1*, *2L-1*, *3L-1*, ...

Etiqueta: Lo que va a diferenciar a todos los bloques a los que les corresponda la misma línea son los *b-l* bits de mayor peso, esto es, el campo de "etiqueta". Obsérvese que aunque una línea de caché puede corresponder a varios bloques, todos esos bloques tendrán una etiqueta distinta.

Cuando la CPU realiza una lectura, la dirección se divide en estos tres campos. Tomando los *b* bits del campo de bloque se obtiene el número de bloque de memoria principal. Con el algoritmo arriba indicado se obtiene la línea que le corresponde al bloque. Si la línea no está ocupada, se trae el bloque desde memoria principal a esa línea, y con el desplazamiento que indican los *d* bits de menor peso de la dirección se obtiene la celda dentro del bloque. Los *b-l* bits de mayor peso de la dirección deben ponerse en el campo de etiqueta de la línea correspondiente de la caché.

En una referencia posterior, cuando se compruebe si el bloque referenciado está en la caché, si la entrada correspondiente está ocupada, hay que comprobar si el bloque de esa entrada es el que corresponde a la dirección que se está referenciando. Para ello simplemente hay que comprobar que el campo de etiqueta de la dirección es igual a la etiqueta de la línea que corresponde a ese bloque. Si no es así, habrá que traer el bloque de memoria y sustituir al que estaba en esa línea. Esto puede representar un problema.

Ejemplo

Tamaño caché: 4 Kbytes
Tamaño bloque: 4 bytes
→ Caché de 1 Klíneas de 4 bytes

Memoria principal: 64 Kbytes (16 Kbloques de 4 bytes)

Dirección M.P.

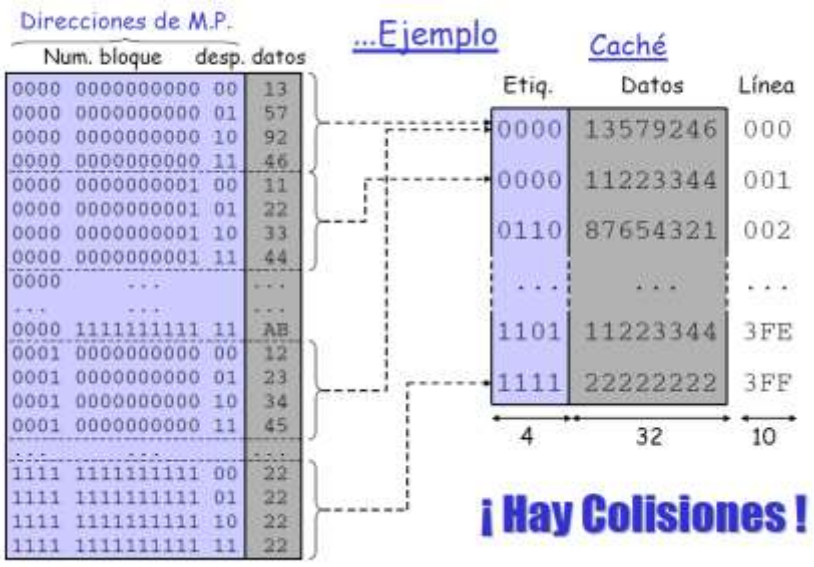
Línea caché	Bloques de memoria principal
0	0, 400, 800, C00, ..., 3C00
1	1, 401, 801, C01, ..., 3C01
. . .	. . .
3FF	3FF, 7FF, BFF, ..., 3FFF

Veamos un ejemplo de ubicación mediante correspondencia directa. Para ello, supondremos los siguientes valores:

1. El tamaño de la caché es de 4 Kbytes.
2. Los datos se transfieren entre la memoria principal y la caché en bloques de 4 bytes. Esto quiere decir que la caché está organizada en 1024 líneas de 4 bytes cada una.
3. La memoria principal es de 64 Kbytes, pudiendo direccionar a nivel de byte mediante direcciones de 16 bits. Esto quiere decir que, a efectos de la caché, la podemos considerar como una memoria de 16 Kbloques de 4 bytes cada uno.

Veamos el formato de las direcciones. Los 16 Kbloques de la memoria principal se referencian mediante los 14 bits de más peso de la dirección; los dos bits de menor peso constituyen el desplazamiento de la palabra dentro del bloque. En la caché, por su parte, el número de línea se expresa mediante 10 bits. La ejecución del algoritmo de la función de correspondencia para averiguar la línea que le corresponde a un bloque de memoria (dividir un número de 14 bits entre otro de 10 y tomar el resto) es lo mismo que tomar directamente los 10 bits de menor peso del dividendo, es decir, del número de bloque.

En la parte inferior de la transparencia podemos ver las correspondencias entre bloques y líneas para este ejemplo.



Las direcciones de memoria están descompuestas en número de bloque y desplazamiento. El número de bloque lo mostramos dividido en dos campos, para ver claramente la etiqueta de cada bloque y la línea que le corresponde.

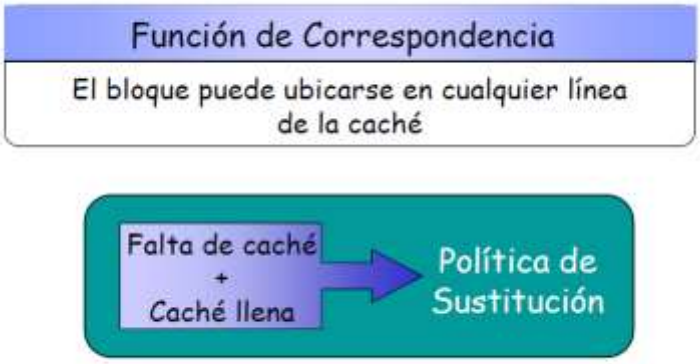
Así, podemos ver cómo al bloque 0 le corresponde la línea 0; al bloque 1, la línea 1; y en último lugar, al bloque 3FFF le corresponde la última línea, la 3FF.

Como ya sabíamos, al bloque 400 H, también le corresponde la línea 0, pero como se puede apreciar, el contenido de esta línea es el del bloque 0, y es razonable, puesto que la etiqueta de la línea 0 es la correspondiente al bloque 0, y no la del bloque 400 H, que tiene una etiqueta 0001.

La técnica de la correspondencia directa es simple y económica, pues la ubicación se obtiene directamente a partir del algoritmo de la función de correspondencia.

Sin embargo, el problema que presenta la correspondencia directa son las **colisiones**; es decir, que a cada línea de caché (donde sólo cabe un bloque) le corresponden muchos bloques de memoria principal. Así, si un programa referencia repetidamente dos palabras a cuyos bloques les corresponde la misma línea de la caché, los dos bloques estarán continuamente expulsándose de la caché, con lo que la tasa de aciertos descenderá drásticamente aún cuando la caché no esté completamente ocupada.

MEMORIA CACHE
DE CORRESPONDENCIA ASOCIATIVA



Visto el problema de las colisiones que presenta la correspondencia directa, veamos otras alternativas

Con la **correspondencia asociativa** (o completamente asociativa) se solventan los problemas de la correspondencia directa, pues aquí se permite que cada bloque de memoria pueda estar en cualquier línea de la caché, por lo que mientras la memoria caché no esté llena, no habrá que hacer ninguna sustitución. Cuando esté llena y haya que traer un nuevo bloque, habrá que sustituir alguno de los bloques según la política de sustitución más apropiada, es decir, la que genere menos faltas de caché.

MEMORIA CACHE
DE CORRESPONDENCIA ASOCIATIVA

- Se permite que cada bloque de memoria pueda estar en cualquier línea de la cache
- Cuando este llena y haya que traer un nuevo bloque se deberá utilizar la política de sustitución más adecuada

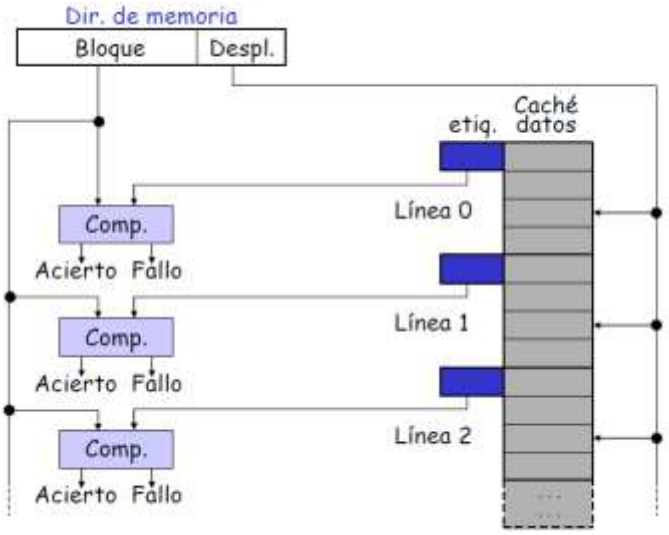
Con la correspondencia asociativa, la caché ve cada dirección de memoria formada solamente por dos campos: el desplazamiento dentro del bloque (los bits menos significativos) y el número de bloque o etiqueta (los más significativos). Ahora cada bloque de memoria principal tiene una única etiqueta posible, que es precisamente el número de bloque.

Así, para saber si un bloque está en la caché, su lógica de control debe comparar la etiqueta de la dirección generada por la CPU con todas las etiquetas de la caché. Para que estas comparaciones puedan realizarse rápidamente, cada entrada de la caché cuenta con un comparador, de tal manera que las comparaciones de la etiqueta de la dirección de memoria con todas las etiquetas de las líneas de la caché se realizan en paralelo. (Este tipo de memorias se denominan **memorias asociativas**).

Con este esquema hay flexibilidad para ubicar un bloque en cualquier línea de la caché, y su espacio se puede aprovechar más eficientemente, pues cuando se trae un bloque a la caché nunca habrá que reemplazar a ninguno de los que ya estaban cargados a menos que todas las líneas estén ocupadas. Con esta técnica ya no deben producirse las repetidas expulsiones mutuas de dos bloques que teníamos con la correspondencia directa. Los algoritmos de sustitución que veremos más adelante se diseñarán precisamente para mejorar lo más posible la tasa de aciertos.

La desventaja obvia de la correspondencia asociativa es el incremento económico que genera la electrónica adicional necesaria.

MEMORIA CACHE DE CORRESPONDENCIA ASOCIATIVA



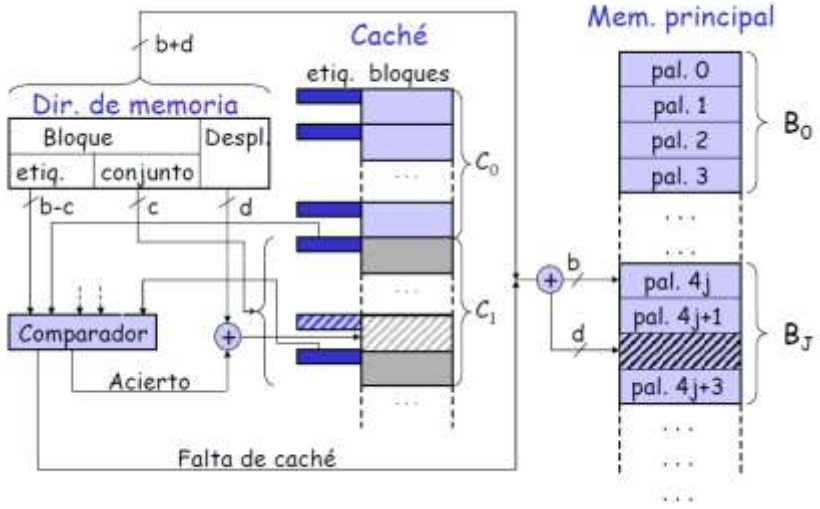
La **correspondencia asociativa de conjuntos** combina la economía de la correspondencia directa con la tasa de aciertos de la correspondencia asociativa. Consiste en agrupar las líneas de la caché en conjuntos, de tal forma que la función de correspondencia permita que un bloque de la memoria principal pueda ubicarse en cualquier línea de un conjunto concreto.

Con esta política, la memoria caché se divide en C conjuntos de L líneas cada uno. Así, el número M de líneas de la caché es $M = C \times L$.

De forma similar a la correspondencia directa, para averiguar el conjunto c de la caché que le corresponde a un cierto bloque b de memoria principal, se aplica la siguiente correspondencia:

$$c = b \text{ módulo } C$$

Una vez averiguado el conjunto c de la caché que le corresponde al bloque b, éste puede ubicarse en cualquiera de las líneas del conjunto c.



También de manera equivalente a la correspondencia directa, cuando la CPU suministra una dirección para acceder a la memoria principal, la dirección se descompone en dos campos: *bloque* y *desplazamiento*. Ya que todos los bloques de memoria principal no caben en la caché, el campo *bloque* se divide, a su vez, en otros dos campos: *conjunto* y *etiqueta*. Ya sabemos que la operación *número módulo* 2^n es lo mismo que tomar los *n* bits de menor peso del *número*, por lo que el conjunto asignado a un bloque de memoria se obtiene con los *c* bits siguientes al desplazamiento. Por último, los bits restantes (los de mayor peso) forman la etiqueta.

Un bloque de memoria solamente puede estar en un conjunto de la caché, pero dentro del conjunto hay varios bloques, por ésto es necesario disponer de la etiqueta, pues no puede haber dos bloques del mismo conjunto con la misma etiqueta.

Obsérvese que si el número de líneas por conjunto se lleva a los **casos extremos**, se da lugar a las otras dos correspondencias que hemos visto. Cuando el número de líneas por conjunto es 1, se está en el caso de la correspondencia directa; mientras que si la caché está formada por un único conjunto que contiene todas las líneas de la caché, se trata de una correspondencia completamente asociativa.

Normalmente **se suelen utilizar 2 líneas por conjunto** (memoria asociativa de dos vías), lo cual mejora notablemente la tasa de aciertos de la correspondencia directa. A medida que se aumenta el número de líneas por conjunto, aumenta el coste de la memoria pero sin conseguir una mejora significativa.

Con esta técnica se disminuye mucho el coste de la electrónica adicional de la correspondencia asociativa, pues mientras que en ésta se requieren tantos comparadores como líneas, en la asociativa de conjunto de dos vías, solamente son necesarios dos comparadores.

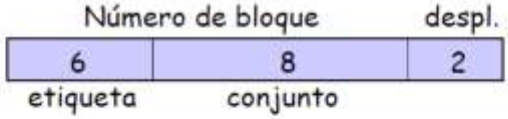
Nuestro Ejemplo

Tamaño caché: 4 Kbytes
Tamaño bloque: 4 bytes
→ Caché de 1 Klínea de 4 bytes

Conjuntos de 4 bloques → Caché de 256 conjuntos

Memoria principal: 64 Kbytes (16 Kbloques de 4 bytes)

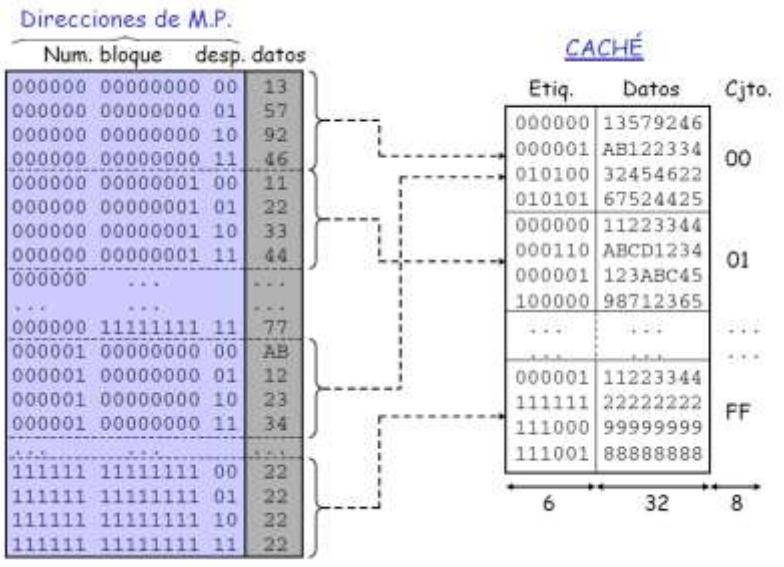
Dirección M.P.



Cjto. caché	Bloques de memoria principal
0	0, 100, 200, ..., 3F00
1	1, 101, 201, ..., 3F01
...	...
FF	FF, 1FF, 2FF, ..., 3FFF

Veamos ahora cómo aplicar la correspondencia asociativa de conjuntos a nuestro ejemplo tipo.

Ahora la dirección de 16 bits vuelve a descomponerse en tres campos. El desplazamiento sigue estando indicado por los dos bits de menor peso, puesto que los bloques siguen siendo de 4 palabras. En cuanto al número de bloque, queda indicado por los 14 bits de mayor peso, aunque en este caso, para conocer la ubicación de cada bloque en la caché solamente necesitamos los 8 bits de menor peso del número de bloque, que es el resultado de la operación **`Num_Bloque módulo Num_Conjuntos`**. Así, tenemos que los bloques con número 0, 100H, 200H, etc., deben ubicarse en el conjunto 0 de la caché; los que tienen número 1, 101H, 201H, etc., corresponden al conjunto 1, y así sucesivamente.



Al igual que en la correspondencia directa, nos encontramos con que hay colisiones, es decir, hay muchos bloques que les corresponde el mismo conjunto. Sin embargo, mientras que en la correspondencia directa solamente podía haber un bloque en una dirección, ahora tenemos que en una misma dirección de la caché puede haber tantos bloques como permita el tamaño del conjunto. En nuestro ejemplo los conjuntos son de 4 bloques, como el Motorola 68040 y el PowerPC 604, que también utilizan una memoria asociativa con conjuntos de 4 bloques (memoria asociativa de conjuntos de 4 vías). Los Pentium, sin embargo, disponen de una caché asociativa de conjuntos de 2 vías.

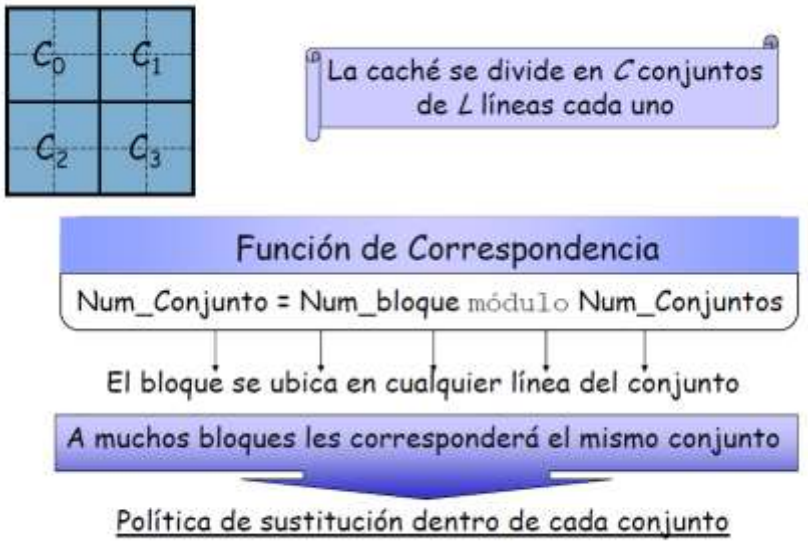
Volviendo a nuestro ejemplo, podemos ver que al primer bloque de memoria principal (el bloque 0) le corresponde el conjunto 0 de la caché, y al segundo bloque, el conjunto 1. Como ya sabíamos, nos encontramos con que al bloque 100H también le corresponde el conjunto 0, pero en esta ocasión, para cargar el bloque 100H no hay que expulsar al bloque 0 (como sucedía en la correspondencia directa), ya que ahora en cada conjunto se pueden ubicar hasta 4 de los bloques a los que la función de correspondencia otorga la misma dirección. Para diferenciar los distintos bloques que corresponden al mismo conjunto se utiliza la etiqueta de 6 bits que tiene cada bloque. No puede haber dos bloques con la misma etiqueta que les corresponda el mismo conjunto.

De esta manera se evita en gran medida el problema de la correspondencia directa, en la que dos bloques muy usados en un bucle podían estar expulsándose mútuamente, desaprovechando así la eficacia de la memoria caché.

CORRESPONDENCIA ASOCIATIVA DE CONJUNTOS

- La memoria cache se divide en C conjuntos de L líneas cada uno
- Normalmente se utilizan dos líneas por conjunto
- Combina la economía de la correspondencia directa con la tasa de aciertos de la correspondencia asociativa

Correspondencia Asociativa de Conjuntos



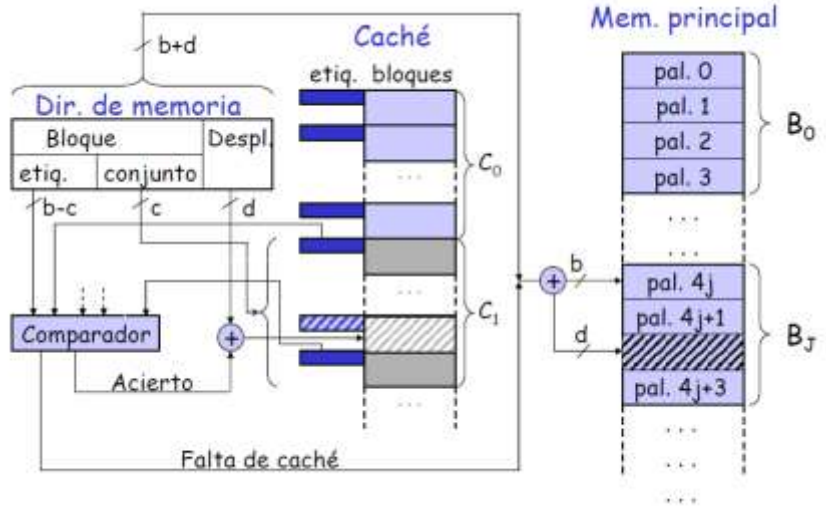
La **correspondencia asociativa de conjuntos** combina la economía de la correspondencia directa con la tasa de aciertos de la correspondencia asociativa. Consiste en agrupar las líneas de la caché en conjuntos, de tal forma que la función de correspondencia permita que un bloque de la memoria principal pueda ubicarse en cualquier línea de un conjunto concreto.

Con esta política, la memoria caché se divide en *C* conjuntos de *L* líneas cada uno. Así, el número *M* de líneas de la caché es $M = C \times L$.

De forma similar a la correspondencia directa, para averiguar el conjunto *c* de la caché que le corresponde a un cierto bloque *b* de memoria principal, se aplica la siguiente correspondencia:

$$c = b \text{ módulo } C$$

Una vez averiguado el conjunto *c* de la caché que le corresponde al bloque *b*, éste puede ubicarse en cualquiera de las líneas del conjunto *c*.



También de manera equivalente a la correspondencia directa, cuando la CPU suministra una dirección para acceder a la memoria principal, la dirección se descompone en dos campos: *bloque* y *desplazamiento*. Ya que todos los bloques de memoria principal no caben en la caché, el campo *bloque* se divide, a su vez, en otros dos campos: *conjunto* y *etiqueta*. Ya sabemos que la operación *número módulo* 2^n es lo mismo que tomar los *n* bits de menor peso del *número*, por lo que el conjunto asignado a un bloque de memoria se obtiene con los *c* bits siguientes al desplazamiento. Por último, los bits restantes (los de mayor peso) forman la etiqueta.

Un bloque de memoria solamente puede estar en un conjunto de la caché, pero dentro del conjunto hay varios bloques, por ésto es necesario disponer de la etiqueta, pues no puede haber dos bloques del mismo conjunto con la misma etiqueta.

Obsérvese que si el número de líneas por conjunto se lleva a los **casos extremos**, se da lugar a las otras dos correspondencias que hemos visto. Cuando el número de líneas por conjunto es 1, se está en el caso de la correspondencia directa; mientras que si la caché está formada por un único conjunto que contiene todas las líneas de la caché, se trata de una correspondencia completamente asociativa.

Normalmente **se suelen utilizar 2 líneas por conjunto** (memoria asociativa de dos vías), lo cual mejora notablemente la tasa de aciertos de la correspondencia directa. A medida que se aumenta el número de líneas por conjunto, aumenta el coste de la memoria pero sin conseguir una mejora significativa.

Con esta técnica se disminuye mucho el coste de la electrónica adicional de la correspondencia asociativa, pues mientras que en ésta se requieren tantos comparadores como líneas, en la asociativa de conjunto de dos vías, solamente son necesarios dos comparadores.

Nuestro Ejemplo

Tamaño caché: 4 Kbytes
Tamaño bloque: 4 bytes
→ Caché de 1 Klínea de 4 bytes

Conjuntos de 4 bloques → Caché de 256 conjuntos

Memoria principal: 64 Kbytes (16 Kbloques de 4 bytes)

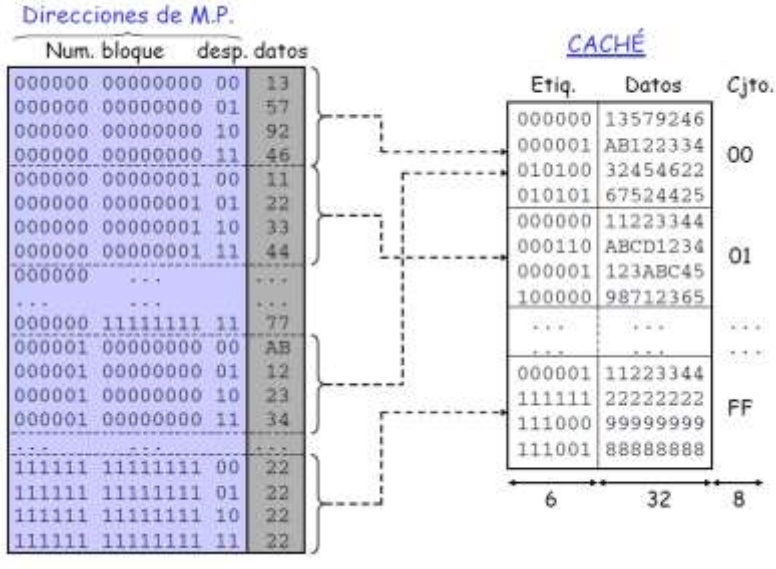
Dirección M.P.

Número de bloque		despl.
6	8	2
etiqueta	conjunto	

Cjto. caché	Bloques de memoria principal
0	0, 100, 200, ..., 3F00
1	1, 101, 201, ..., 3F01
...	...
FF	FF, 1FF, 2FF, ..., 3FFF

Veamos ahora cómo aplicar la correspondencia asociativa de conjuntos a nuestro ejemplo tipo.

Ahora la dirección de 16 bits vuelve a descomponerse en tres campos. El desplazamiento sigue estando indicado por los dos bits de menor peso, puesto que los bloques siguen siendo de 4 palabras. En cuanto al número de bloque, queda indicado por los 14 bits de mayor peso, aunque en este caso, para conocer la ubicación de cada bloque en la caché solamente necesitamos los 8 bits de menor peso del número de bloque, que es el resultado de la operación **`Num_Bloque módulo Num_Conjuntos`**. Así, tenemos que los bloques con número 0, 100H, 200H, etc., deben ubicarse en el conjunto 0 de la caché; los que tienen número 1, 101H, 201H, etc., corresponden al conjunto 1, y así sucesivamente.



Al igual que en la correspondencia directa, nos encontramos con que hay colisiones, es decir, hay muchos bloques que les corresponde el mismo conjunto. Sin embargo, mientras que en la correspondencia directa solamente podía haber un bloque en una dirección, ahora tenemos que en una misma dirección de la caché puede haber tantos bloques como permita el tamaño del conjunto. En nuestro ejemplo los conjuntos son de 4 bloques, como el Motorola 68040 y el PowerPC 604, que también utilizan una memoria asociativa con conjuntos de 4 bloques (memoria asociativa de conjuntos de 4 vías). Los Pentium, sin embargo, disponen de una caché asociativa de conjuntos de 2 vías.

Volviendo a nuestro ejemplo, podemos ver que al primer bloque de memoria principal (el bloque 0) le corresponde el conjunto 0 de la caché, y al segundo bloque, el conjunto 1. Como ya sabíamos, nos encontramos con que al bloque 100H también le corresponde el conjunto 0, pero en esta ocasión, para cargar el bloque 100H no hay que expulsar al bloque 0 (como sucedía en la correspondencia directa), ya que ahora en cada conjunto se pueden ubicar hasta 4 de los bloques a los que la función de correspondencia otorga la misma dirección. Para diferenciar los distintos bloques que corresponden al mismo conjunto se utiliza la etiqueta de 6 bits que tiene cada bloque. No puede haber dos bloques con la misma etiqueta que les corresponda el mismo conjunto.

De esta manera se evita en gran medida el problema de la correspondencia directa, en la que dos bloques muy usados en un bucle podían estar expulsándose mútuamente, desaprovechando así la eficacia de la memoria caché.

Políticas de sustitución

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 recién 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 del 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 está 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 período 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 víctima, parece razonable elegir el que lleva más tiempo sin ser referenciado (el menos recientemente referenciado, *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 se referencia un bloque su contador se pone a cero, y los de 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 a cero.

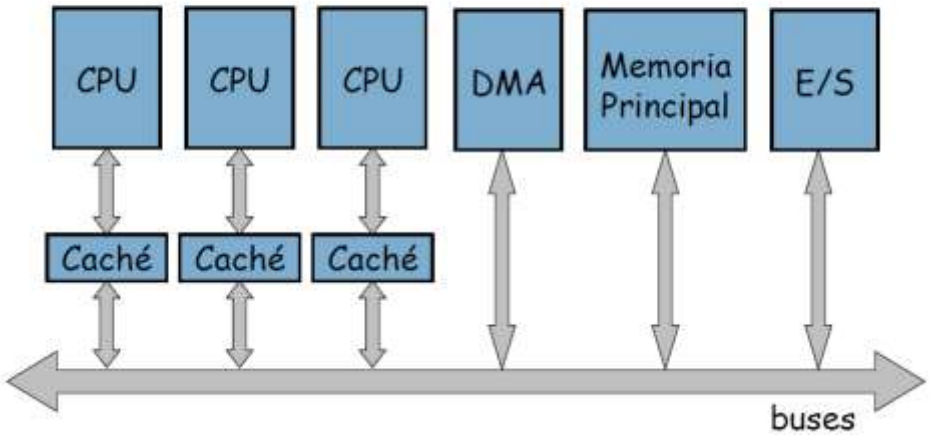
Esta política de sustitución falla cuando se está accediendo de forma secuencial y cíclica a elementos de una matriz que no cabe completamente en la caché.

Una política parecida es la **LFU (*Least Frequently Used*)**, en la cual se 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 deben al hecho de que solamente tiene en cuenta el tiempo que lleva un bloque en la caché, y no cuáles han sido las últimas 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 fácil y económico de implementar.

El Problema de Coherencia de las Cachés



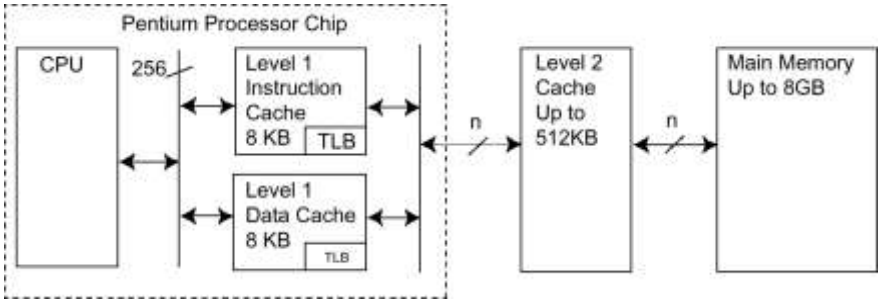
Coherencia

El problema de la coherencia de cachés tiene que ver con la **política de actualización**.

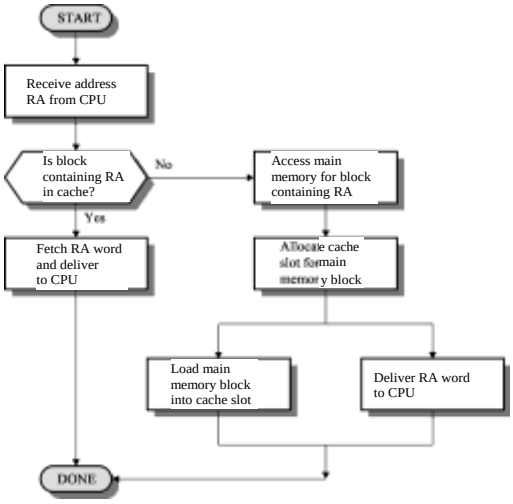
Aquí hay dos situaciones a considerar. En un ordenador puede haber más de un elemento que acceda a la memoria principal, pues en un ordenador actual no es raro encontrarnos con varias CPU's o, simplemente, los dispositivos de entrada/salida gobernados directamente desde alguna CPU o a través del controlador de DMA. Según esto, cuando desde una CPU se modifica un dato en su caché, el correspondiente dato en memoria principal queda obsoleto, con lo que si desde otra CPU o dispositivo se accede al dato original en memoria principal, resulta que se accede a un dato que no está actualizado. También puede suceder que sea un dispositivo de entrada el que modifica el dato en memoria principal con un nuevo valor, con lo que entonces es el dato de la caché el que queda obsoleto.

Esto es lo que se conoce como el **problema de la coherencia de las cachés**.

Sistema de memoria INTEL



Operación de lectura del cache



Memorias

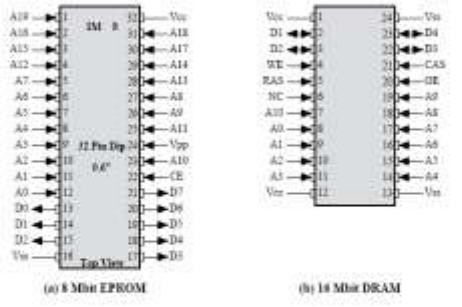
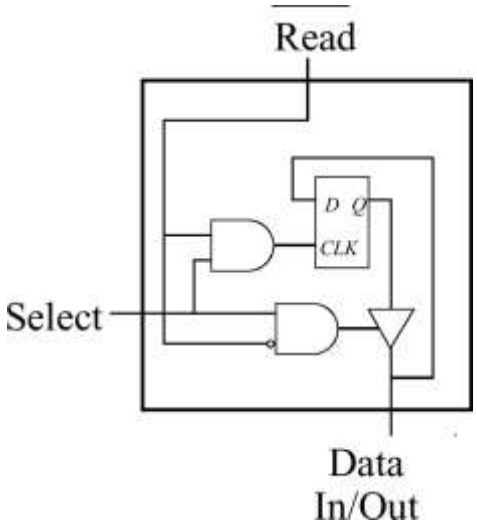


Figure 4.5 Typical Memory Package Pins and Signals

Diagrama funcional de una celda RAM



- ✓No es la representación física real sino la ideal
- ✓Se conocen como SRAM: Static RAM
- ✓El valor se mantiene mientras se mantenga la alimentación del circuito integrado.

RAM – Memoria de acceso aleatorio

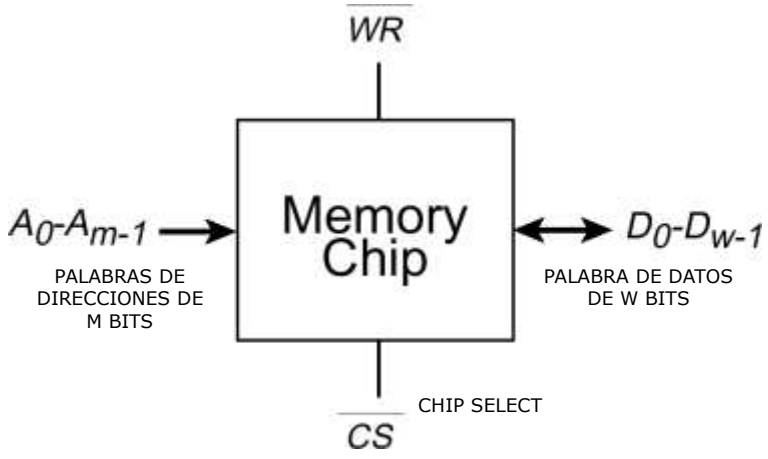
- ESTATICAS (SRAM)
  - Los bits se almacenan como si fueran en llaves si/no
  - No requieren refresco
  - Construcción mas compleja
  - De mayor tamaño
  - **Mas caras**
  - **Mas rápidas**
  - Uso: CACHE

RAM – Memoria de acceso aleatorio

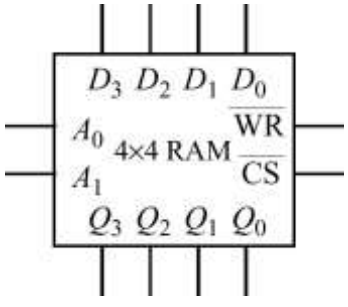
- DINAMICAS (DRAM)
  - Los bits se almacenan como si fueran capacitores
  - Requieren refresco
  - Construcción mas SIMPLE
  - De menor tamaño
  - **Mas baratas**
  - **Mas lentas**
  - Uso: MEMORIA PRINCIPAL

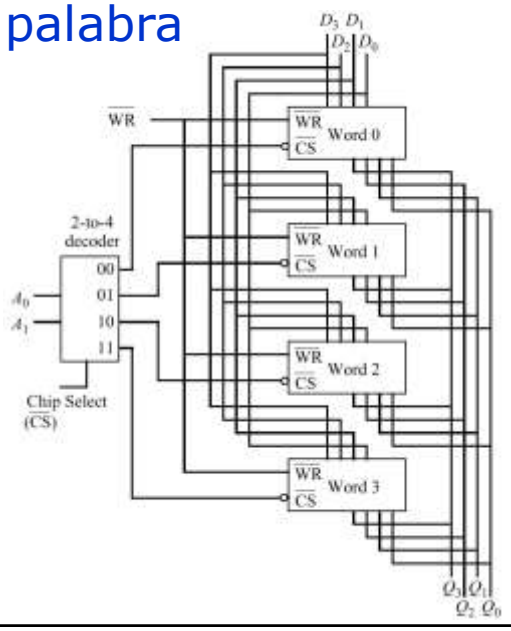
Salidas simplificadas de un chip RAM

EL RETARDO DE TIEMPO HASTA QUE APARECEN LAS SEÑALES EN LA SALIDA ES T_{AA}, QUE ES EL TIEMPO EN QUE TIENEN QUE SER VALIDAS LAS SEÑALES DE ENTRADA

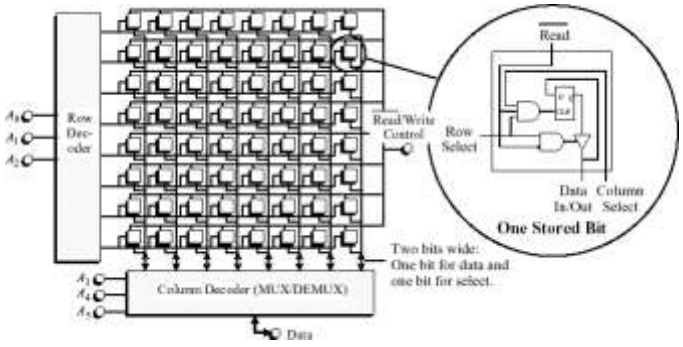


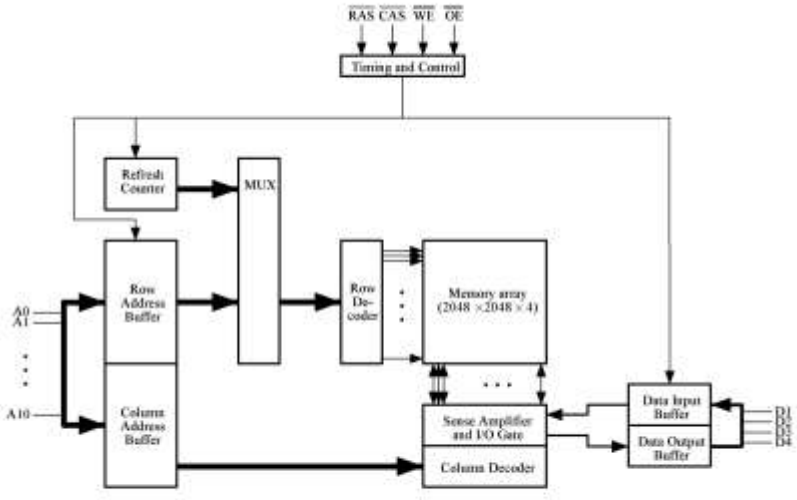
Memoria de cuatro palabras con cuatro bits por palabra





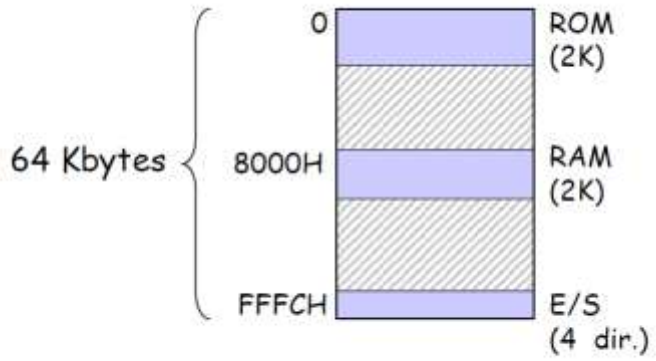
Ram de 64 palabras de 1 bit



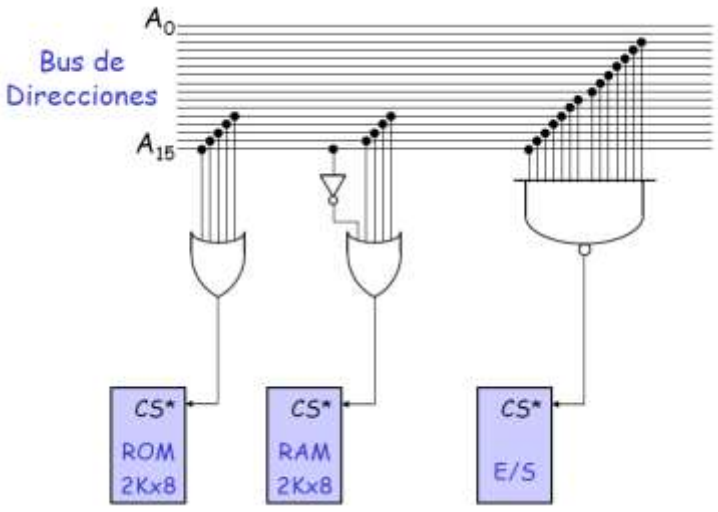


Typical 16 Megabit DRAM (4M × 4)

MAPA DE MEMORIA PRINCIPAL

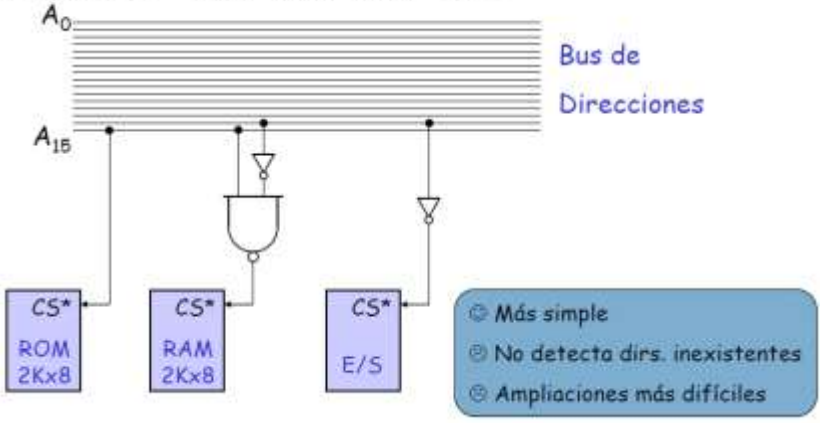


DECODIFICACION TOTAL



DECODIFICACION PARCIAL

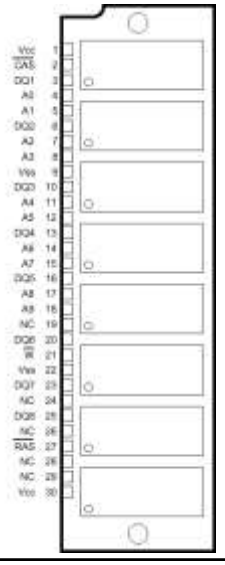
Direcciones ROM: 0000 0xxx xxxx xxxx
Direcciones RAM: 1000 0xxx xxxx xxxx
Puertos de E/S: 1111 1111 1111 11xx



Modulo de memoria Single-in-line SIMM

Utilizado en las 486, trabajaban de a pares.
Sustituidos por los DIMM

PIN NOMENCLATURE	
A0-A9	Address Inputs
$\overline{CAS}$	Column-Address Strobe
DQ1-DQ8	Data In/Data Out
NC	No Connection
$\overline{RAS}$	Row-Address Strobe
V_{CC}	5-V Supply
V_{SS}	Ground
$\overline{W}$	Write Enable

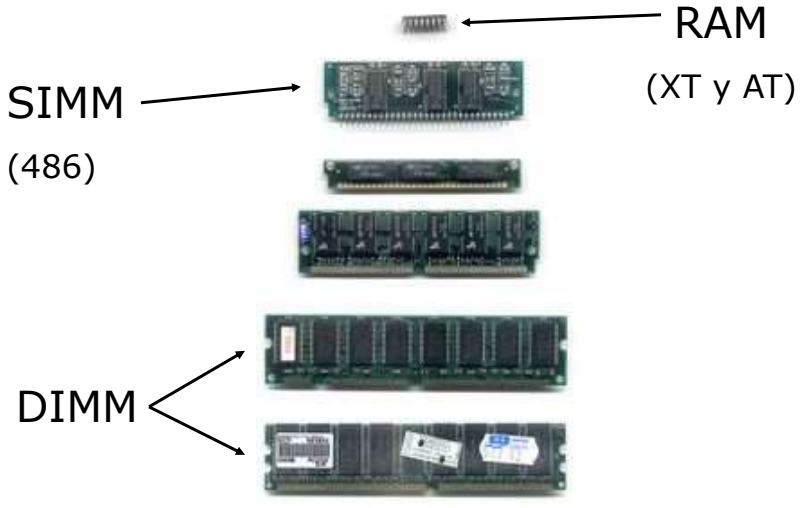


RAM

DIMM Dual In Line Memory Modules

Tienen los contactos de cada cara de la plaqueta separados

NOMBRE		CONTACTOS	CAPACIDAD (MB)	CLOCK (MHz)	Obs.
SO DIMM	**S**MALL **O**UTLINE	72	Hasta 512		Se usan en Laptops
SO DIMM	**S**MALL **O**UTLINE	144	Hasta 1 GB		
SO DIMM	**S**MALL **O**UTLINE	200	Hasta 1GB		
SDRAM	**S**YNCHONOUS **D**INAMIC **R**ANDOM **A**CCES **M**EMORY	168	64, 128, 256 y 512	66 a 133	Se conecta al clock del sistema. Lee o escribe a un ciclo de reloj por acceso
DDR SDRAM	**D**OUBLE **D**ATA **R**ATE	184	Hasta 1 GB	200 a 400	Transmiten por dos canales distintos simultáneamente en el mismo ciclo de reloj
DDR2 SDRAM		240	Hasta 2x2 GB	400 a 1200	Transmiten por cuatro canales simultáneamente



SO DIMM

DDR2

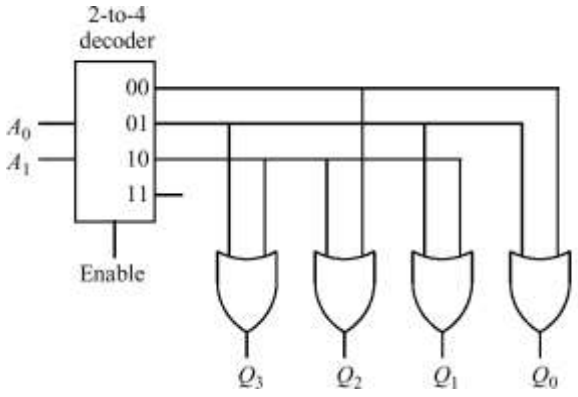
ROM – Memoria de solo lectura

- Almacenamiento permanente
  - Microprogramables
  - Subrutinas
  - BIOS
  - Tablas de función

ROM de palabras de 4 bits

Tengo **1** solo en la salida elegida, y **0** en todos los demás.

Si A_0A_1 es 00, en la salida 00 tengo **1**, y en 01, 10 y 11 **0** como salida



Location	Stored word
00	0101
01	1011
10	1110
11	0000

ROM

NOMBRE		CONTACTOS	CAPACIDAD (MB)	CLOCK (MHz)	Obs.
PROM	**P**ROGRAMABLE **R**EAD **O**NLY **M**EMORY	28			Los datos almacenados no se pueden modificar
EPROM	**E**RASABLE **P**ROGRAMABLE **R**EAD **O**NLY **M**EMORY	28	Hasta 8		Los datos almacenados se pueden borrar mediante una luz ultravioleta
EAROM	**E**LECTRICALLY **A**LTERABLE **R**EAD **O**NLY **M**EMORY		MUY BAJA	W: 0,001 R: 1	ESCRITURA 1 mseg LECTURA 1 µseg
. . . . .	**E**LECTRICALLY **E**RASABLE **P**ROGRAMABLE **R**EAD **O**NLY **M**EMORY				Pueden borrarse y reprogramarse entre 100.000 y 1.000.000 de veces
FLASH	Pen drive PC card etc		HASTA 32 GB	20	EEPROM EVOLUCIONADA

ROM

PROM

MEMORIAS PROM COMO ALU

Se usan como UNIDADES DE CONTROL y ALU

Como ALU se almacenan los resultados en las locaciones de memoria que correspondan,

Es útil cuando se utilizan palabras de hasta 8 bits

$2^{16} \times 2^2 = 2^{18}$

Donde 2^{16} son dos palabras de 8 bits.

Cantidad de operaciones posibles

Memoria posible

Si fuera con palabras de 32 bits quedaría

$$\mathbf{2^{64} \times 2^2 = 2^{128}}$$

EPROM