**MEMORIA CACHÉ**

**BREVES CONCEPTOS TEÓRICOS**

http://www.atc.uniovi.es/inf\_med\_gijon/3ingcomp/practicas/SMC/teor%C3%ADa-conceptos.htm

[1 Introducción](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_INTRODUCCIÓN)

[1.1 Sistema de memoria](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Sistema_de_memoria)

[1.2 Jerarquía de memoria](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Jerarquía_de_memoria)

[1.2.1 Principio de funcionamiento](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Principio_de_funcionamiento)

[1.2.2 Principio de localidad temporal](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Principio_de_localidad_temporal)

[1.2.3 Principio de localidad espacial](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Principio_de_localidad_espacial)

[1.3 Memoria principal](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Memoria_principal)

[1.4 Memoria caché](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Memoria_caché)

[1.4.1 Organización de la memoria caché](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Organización_de_la_memoria_caché)

[1.4.1.1          Niveles de caché](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Niveles_de_caché)

[1.4.1.2          Caché unificada](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Caché_unificada)

[1.4.1.3          Caché dividida](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Caché_dividida)

[1.4.2 Estrategias de correspondencia](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Estrategias_de_correspondencia)

[1.4.2.1          Directa](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Directa)

[1.4.2.2         Totalmente asociativa](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Totalmente_asociativa)

[1.4.2.3         Asociativa por conjuntos](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Asociativa_por_conjuntos)

[1.4.3 Estrategias de reemplazo](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Estrategias_de_reemplazo)

[1.4.3.1          LRU (Least Recently Used)](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_LRU_(Least_Recently_Used))

[1.4.3.2         Reemplazo aleatorio](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Reemplazo_aleatorio)

[1.4.4 Estrategias de escritura](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Estrategias_de_escritura)

[1.4.4.1          Write-through o escritura a través](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Write-through_o_escritura_a_través)

[1.4.4.2         Write-back o escritura diferida](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Write-back_o_escritura_diferida)

[1.4.5 Medidas del rendimiento en las cachés](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Medidas_del_rendimiento_en_las_cach)

[1.4.5.1          Coste temporal de un fallo de caché](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Coste_temporal_de_un_fallo_de_caché)

[1.4.5.2         Coste temporal de la actualización de un bloque de memoria principal](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Coste_temporal_de_la_actualización_)

[1.4.5.3         Frecuencia de aciertos (%)](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Frecuencia_de_aciertos_(%))

[1.4.5.4         Frecuencia de fallos (%)](file:///\\156.35.151.4\www-data\inf_med_gijon\3ingcomp\practicas\SMC\teoria-conceptos.htm#_Frecuencia_de_fallos_(%))

**1**         **INTRODUCCIÓN**

Este proyecto permite configurar un sistema de memoria sobre el cual simular el comportamiento de acceso a memoria durante la ejecución de programas.

Para comprender los conceptos que trataremos en este proyecto, incluimos aquí un breve documento de conceptos teóricos. Se comienza con una definición de un sistema de memoria, que se completará a continuación con el análisis de por qué es necesario organizar una estructura jerárquica de memoria.

Hay algunos tipos de memoria que no son aplicables para este proyecto, como por ejemplo la memoria virtual, ya que aquí vamos a centrarnos en el nivel de memoria más cercano a la CPU, es decir, la caché, cuyas características y propiedades hablaremos más adelante. También mencionaremos en este apartado la memoria principal, ya que ésta sí se tiene en cuenta en nuestra aplicación. Los demás tipos de memoria no los trataremos en este apartado de conceptos teóricos por la razón mencionada anteriormente. Sin embargo, no podemos perder de vista que aunque este simulador se centre en las relaciones entre la CPU, memoria caché y memoria principal, en un sistema real habría que tener también en cuenta la memoria virtual.

***1.1***        ***Sistema de memoria***

La misión del sistema de memoria es servir como almacén para las instrucciones y datos que constituyen los programas. Por tanto, interesa que sea grande para poder almacenar muchos programas, muy grandes y con muchos datos. Además, interesa que sea rápida ya que la CPU accede a memoria al menos una vez en cada instrucción, en la fase de búsqueda del código de instrucción. Si además el operando de la instrucción también está en memoria, debe realizar un nuevo acceso. En los accesos a memoria, como la CPU es más rápida que la memoria es necesario añadir estados de espera en los cuales la CPU está parada, como sucede para las instrucciones del tipo MOV [Ri], Rs ó MOV Rd, [Ri] en la CPU teórica.

El problema de la rapidez de la memoria se ha ido acentuando con el paso del tiempo y el desarrollo tecnológico seguido por los distintos elementos del computador. Así, la velocidad de la CPU ha experimentado un incremento muy importante, duplicándose aproximadamente cada tres años. La memoria, en cambio, ha evolucionado principalmente en densidad, es decir, en capacidad de almacenamiento, pero en menor medida en velocidad.

Para solucionar estos problemas se decide utilizar una Jerarquía de Memoria.

***1.2***        ***Jerarquía de memoria***

En la construcción del sistema de memoria de un computador se persiguen entre otros tres objetivos: alta velocidad, alta capacidad y bajo coste.

Si analizamos los diferentes tipos de tecnologías de memoria existentes en la actualidad (SRAM, DRAM y almacenamiento magnético) llegamos a la conclusión de que no hay ninguna tecnología de memoria que por sí sola satisfaga los requisitos antes mencionados, debido a que:

* La SRAM es la más rápida, pero enormemente costosa.
* El almacenamiento magnético es muy económico y tiene una enorme capacidad pero es extremadamente lento.
* La tecnología DRAM se encuentra en el punto intermedio, no es suficientemente rápida, ni suficientemente grande, pero tampoco excesivamente costosa.

Como consecuencia de esto, la solución al problema reside en combinar memorias rápidas y pequeñas con memorias lentas y grandes de tal forma que:

* Las  memorias rápidas y pequeñas contengan los datos con más probabilidad de acceso.
* Las memorias lentas y grandes contengan los datos con menos probabilidad de acceso.

Con esto se consigue que la capacidad del sistema de memoria sea elevada, pues contiene memorias grandes. Además, en la mayor parte de los casos el sistema de memoria será rápido, pues en la mayor parte de los casos accederá a las memorias rápidas.

Para poder llevar a la práctica la solución anterior es necesario que se cumplan dos condiciones:

* Conocer a priori los datos con mayor probabilidad de acceso futuro para así poder almacenarlos en las memorias pequeñas y rápidas.
* Que estos datos sean pocos y con una gran probabilidad de ser accedidos en el futuro. De esta forma cabrán en las memorias pequeñas y rápidas y además servirán la mayor parte de los accesos futuros, obteniéndose un sistema de memoria rápido en la mayor parte de los casos.

La combinación de las diferentes tecnologías de memoria se hace en niveles de diferente capacidad y velocidad, formando lo que se conoce como jerarquía de memoria.

1.2.1       Principio de funcionamiento

La CPU cuando desea acceder a una posición del sistema de memoria, accede al nivel de la memoria caché. Si esta posición se encuentra dentro de la memoria caché, la CPU accede a ella de forma muy rápida. Esta es la situación más habitual, pues la mayor parte de los accesos de la CPU al sistema de memoria son servidos directamente por la caché.

Si por el contrario la posición no está dentro de la memoria caché, se produce un fallo que trae como consecuencia que un conjunto de posiciones consecutivas que incluyen la posición deseada se transfieran desde la memoria principal a la memoria caché. A este conjunto de posiciones de memoria consecutivas se le denomina bloque. Una vez se ha realizado la transferencia, la CPU accede directamente a la caché. En general, cada vez que se produce un fallo en un nivel N del sistema de memoria, se copia del nivel N+1 al nivel N el bloque que contiene la posición deseada y a continuación se realiza el acceso.

Debe notarse que la unidad de transferencia de información entre dos niveles contiguos de la jerarquía es el bloque. También debe tenerse en cuenta la posibilidad de encadenar fallos en diferentes niveles.

1.2.2       Principio de localidad temporal

Si en un momento dado se accede a una posición de memoria, es probable que poco después se acceda a esa misma dirección de memoria. Un ejemplo de localidad temporal en el acceso a código se encuentra también en la ejecución de bucles, la misma instrucción se accede para ejecutarla varias veces. Un ejemplo de localidad temporal en el acceso a datos es por ejemplo el acceso a un contador durante la ejecución de un bucle.

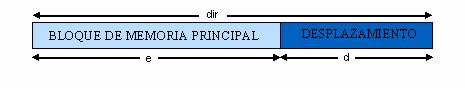
1.2.3       Principio de localidad espacial

Este principio indica que: si en un momento dado se accede a una dirección de memoria, es probable que poco después se acceda a una dirección de memoria cercana. Un ejemplo de localidad espacial es el acceso al código durante la ejecución secuencial de un programa. Otro ejemplo aparece con las instrucciones de un bucle. Un ejemplo de localidad espacial en el acceso a datos se tiene en el acceso a los datos de un array.

***1.3***        ***Memoria principal***

La memoria principal se construye empleando memorias RAM dinámicas, las cuales son de velocidad media y capacidad media.

La dirección emitida por la CPU  se divide en dos campos: el bloque de memoria principal y el desplazamiento dentro del bloque. El bloque de memoria principal en el cual está incluida una dirección emitida por la CPU se obtiene fácilmente teniendo en cuenta que los bits menos significativos de la dirección se emplean para definir el desplazamiento del byte dentro del bloque, y por lo tanto los demás proporcionan el número de bloque de memoria principal.



donde,

2dir = Espacio de direcciones físicas   
2d = Tamaño del bloque   
2e = Número de bloques   
e = dir-d

***1.4***        ***Memoria caché***

La memoria caché es la memoria más rápida del sistema de memoria. En la actualidad se implementa con chips de memoria RAM estática, los cuales son relativamente caros y de baja capacidad.

La memoria caché se divide en bloques, todos ellos conteniendo el mismo número de bytes. Cada bloque contiene un conjunto de datos ubicados en posiciones de memoria consecutivas. Cada vez que la CPU intenta acceder a una dirección de memoria que no está en ningún bloque de la caché se produce un **fallo** de caché, y en caso contrario un **acierto** de caché. La consecuencia del fallo es que se suspende momentáneamente el acceso, se copia el bloque que contiene la dirección desde la memoria principal a la memoria caché y a continuación se reanuda el acceso. Desde el punto de vista de interacción con la memoria caché, la memoria principal se puede ver como un gran almacén de bloques del mismo tamaño que los bloques de la memoria caché.

Por ejemplo,  una memoria caché de 32 bytes organizada en bloques de 4 bytes tendrá 8 bloques. Si la memoria principal es de 256 bytes, la memoria caché percibe a la memoria principal como 64 bloques de 4 bytes cada uno.

Toda memoria caché lleva asociado un controlador de memoria caché, el cual se encarga de gestionar todas las lecturas y escrituras en la memoria caché y decidir por ejemplo si una dirección está cacheada, es decir, ubicada dentro de la memoria caché, o por el contrario se ha producido un fallo de caché.

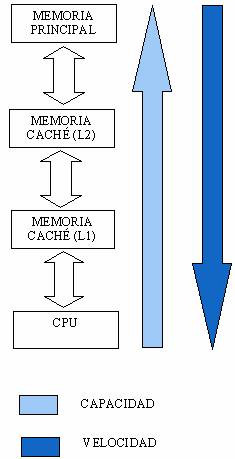
En general, el funcionamiento de cualquier memoria caché viene definido por las siguientes características: estrategia de correspondencia, de reemplazo y de escritura.

1.4.1       Organización de la memoria caché

1.4.1.1     Niveles de caché

Inicialmente cuando se introdujeron las cachés, el sistema típico tenía una única caché. Sin embargo recientemente lo habitual es disponer de dos niveles de caché o incluso tres en computadores de gama alta. Estos niveles se denominan L1,L2,etc., donde el nivel L1 es el más cercano al procesador.

El objetivo de la caché L2 es reducir la penalización debida a los fallos de caché del nivel L1. En teoría, cuantos más niveles de caché haya entre el procesador y la memoria principal mejor rendimiento proporciona el sistema de  memoria pero con un coste mayor.



1.4.1.2     Caché unificada

La caché unificada sirve tanto para datos como para código. Tiene la ventaja de que se reparte automáticamente los bloques cacheados entre datos y código. De tal forma que si son necesarios menos bloques de datos y más de código estos se reparten de forma eficiente y transparente.

1.4.1.3     Caché dividida

La memoria caché dividida está formada por dos cachés, una para datos y otra para código. Podría suceder que la memoria caché de datos tuviese bloques libres o muy poco usados, mientras que la memoria caché de código estuviese generando fallos de caché. Como se puede ver, en este caso el rendimiento sería menor.

Las primeras cachés que aparecieron solían ser unificadas debido a la observación anterior. Sin embargo, las cachés L1 actuales suelen ser divididas, debido a que las CPUs actuales tienen un elevado grado de paralelismo. La caché dividida permite un mayor grado de paralelismo pues una parte de la CPU puede escribir un dato en el sistema de memoria mientras simultáneamente otra parte puede leer instrucciones del sistema de memoria.

1.4.2       Estrategias de correspondencia

Establece a qué bloque o bloques de memoria caché se puede llevar cada bloque de memoria principal.

1.4.2.1     Directa

En este tipo de correspondencia cada bloque de memoria principal se asigna siempre a un mismo bloque que caché, siguiendo esta fórmula:

Bloque de cache = (Bloque de MP) MOD (Nº Bloques cache)

La dirección emitida por la CPU se divide en bits de esta manera,



donde:

2dir = E.D. físicas   
2d = Tamaño del bloque   
2dir-d= Número de bloques en E.D   
2c = Número de bloques en cache   
e = Número de bits para el campo etiqueta (dir - c - d)

La etiqueta es necesaria para diferenciar si un determinado bloque de memoria principal está cacheado o no, ya que a varios bloques de memoria principal les corresponde un mismo bloque en cache y por tanto si no fuera por los bits de etiqueta sería imposible diferenciar qué bloque de memoria principal se hace referencia.

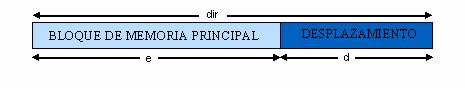
Además de la etiqueta, se guarda en la caché un bit de validez para cada bloque, que indica si los datos que hay en dicho bloque son válidos o no. En la inicialización del computador, se marcarán todos estos bits a falso ya que no son válidos y posteriormente cuando se vayan trayendo datos desde memoria principal hasta la caché se marcará el bloque correspondiente como válido.

1.4.2.2     Totalmente asociativa

La estrategia de correspondencia directa, explicada anteriormente, es muy sencilla de implementar. Su mayor inconveniente proviene de la rigidez a la hora de asignar bloques de memoria principal a memoria caché pues no hay ninguna libertad. Cada bloque de memoria principal puede ir únicamente a un bloque de memoria caché. Si resulta que la CPU accede con mucha frecuencia a dos bloques de memoria principal asignados al mismo bloque de caché, aparecerán con gran frecuencia fallos de caché, lo cual repercute en un bajo rendimiento.

Para paliar este problema, la estrategia de correspondencia asociativa da total libertad. Un bloque  de memoria principal puede asignarse a cualquier bloque de memoria caché.

En este caso, el campo etiqueta  coincide con el número de bloque de memoria principal.



donde,

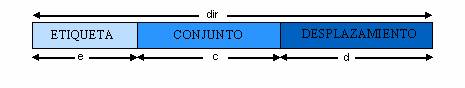
2dir = Espacio de direcciones físicas   
2d = Tamaño del bloque   
e = Bits destinados al campo de etiqueta (dir-d)

1.4.2.3     Asociativa por conjuntos

Esta estrategia tiene como principal cometido el proporcionar una mayor libertad de elección del bloque a reemplazar respecto a la correspondencia directa, pero sin llegar al extremo de la totalmente asociativa, ya que esta última presenta el inconveniente del gran número de comparadores que usa, lo cual incrementa mucho el coste.

La estrategia asociativa por conjuntos divide la memoria cache en conjuntos que contendrán un número fijo de bloques. Al copiar a caché un bloque de memoria prinicpal, obtendremos el número de conjunto al que pertenece, y tendremos libertad para copiarlo en cualquiera de los bloques de dicho conjunto.

La dirección emitida por la CPU se separa de esta forma:



donde,

2dir = E.D. físicas   
2d = Tamaño del bloque   
2dir-d= Número de bloques en E.D   
2c = Número de conjuntos   
e = Bits destinados al campo etiqueta (dir - c - d)

Debido a la distribución lógica interna de una caché asociativa por conjuntos podemos ver que:

* Una caché con correspondencia directa equivale con una asociativa por conjuntos de un bloque por conjunto.
* Una caché con correspondencia totalmente asociativa equivale con una asociativa por conjuntos en la que hay un solo conjunto que almacena todos los bloques.

Adicionalmente se debe tener en cuenta que comúnmente se usa el término **vía** para referirse al número de bloques por conjunto.

1.4.3       Estrategias de reemplazo

Las estrategias de reemplazo establecen qué bloque de la memoria caché debe reemplazarse por un bloque de memoria principal cuando se producen un fallo de caché., Esto sólo tiene sentido cuando el bloque de memoria principal puede ir a más de un bloque de la memoria caché.

1.4.3.1     LRU (Least Recently Used)

El bloque a reemplazar es el que lleva más tiempo sin haber sido accedido. Por tanto, una caché que use esta estrategia, debe mantener esta información de alguna forma.

1.4.3.2     Reemplazo aleatorio

En el caso del reemplazo aleatorio se elige de forma aleatoria entre uno cualquiera de los bloques de caché candidatos.

1.4.4       Estrategias de escritura

Las estrategias de escritura establecen la relación entre las escrituras en la memoria caché y la memoria principal. El objetivo fundamental de estas estrategias es garantizar la coherencia entre la información de la caché y la memoria principal, afectando lo mínimo posible al rendimiento del sistema.

1.4.4.1     Write-through o escritura a través

La estrategia de escritura *write-through* también se denomina escritura a través. En ella, cada vez que se escriba un dato en memoria caché, se modificará inmediatamente en memoria principal. Esto elimina problemas de coherencia, es decir, que cuando un periférico acceda a memoria, siempre accederá a datos actualizados, con lo cual se evita el tener que comprobar si el dato está cacheado y si es diferente al que hay en la memoria principal. Pero por el contrario, en esta estrategia, no se produce la ganancia de tiempo en las escrituras, sino sólo en las lecturas, ya que cada escritura en caché en realidad implica una escritura en memoria principal, con el coste que ello implica.

1.4.4.2     Write-back o escritura diferida

Con la escritura diferida, los datos inicialmente sólo se escriben en la caché. El dato escrito aparece reflejado en la memoria principal sólo cuando el bloque que lo contiene es reemplazado. Con la escritura diferida aparecen problemas de coherencia.

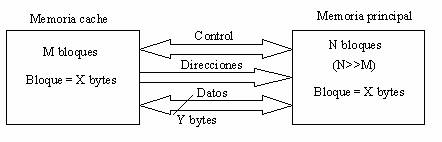
Comparativamente, la estrategia *write-back* proporciona un mayor rendimiento que la *write-through*, a costa de una mayor complejidad hardware para el mantenimiento de la coherencia.

Conviene resaltar que se asocia un bit más a cada bloque de caché, el bit *dirty*, el cual se pone a uno cada vez que la CPU escribe en el bloque de caché.

1.4.5       Medidas del rendimiento en las cachés

1.4.5.1     Coste temporal de un fallo de caché

La conexión entre la memoria caché y la memoria principal puede esquematizar de la siguiente forma:



Cada vez que se produce un fallo de caché se transfiere un bloque de X bytes desde la memoria principal a la memoria caché. Por lo tanto, la memoria caché y la principal se comunican en unidades de bloque.

La transferencia de un bloque desde la memoria principal a la memoria caché requiere un número de accesos de lectura a la memoria principal que depende de dos factores:

* El tamaño de bloque (X bytes)
* El número de líneas de datos (Y bytes)

En la práctica X es un múltiplo de Y, y por lo tanto el número de accesos necesarios es X/Y.

Cada acceso de lectura de memoria principal requiere un tiempo que mediremos en ciclos del bus. Básicamente, un acceso de lectura se divide en las tres operaciones siguientes, cada una con un número de ciclos de bus asociado:

* (OP1) Envío de la dirección por parte del controlador de caché a través de las líneas de direcciones.
* (OP2) Búsqueda del dato direccionado, de tamaño Y bytes, dentro de la memoria principal.
* (OP3) Transmisión del dato desde la memoria principal a la memoria caché sobre las líneas de datos.

1.4.5.2     Coste temporal de la actualización de un bloque de memoria principal

Un bloque de memoria principal se actualiza cuando se cumple alguna de las condiciones siguientes:

* Si la memoria sigue una estrategia de escritura a través o *write-through*, se lleva a cabo el proceso de actualización cada vez que se modifica un bloque de memoria caché.
* Si la memoria sigue una estrategia de escritura diferida o *write-back*, la actualización del bloque de memoria caché se produce:
  + Cuando el bloque de caché modificado es reemplazado.
  + Si un bloque de memoria principal es accedido por la interfaz de un periférico con capacidad DMA, y resulta que el bloque de caché asociado ha sido modificado por la CPU.

La actualización de un bloque de memoria principal trae consigo la escritura de un bloque de caché en memoria principal. Igual que sucedía en el caso de un fallo de caché, son necesarios X/Y accesos a memoria principal para llevar a cabo la transferencia. La diferencia es que en este caso los accesos son de escritura en la memoria principal.

Cada acceso de escritura en memoria principal requiere un tiempo que mediremos también en ciclos de bus y que descomponemos en las siguientes operaciones:

* (OP1) Envío de la dirección por parte del controlador de caché a través de las líneas de direcciones.
* (OP2) Transmisión del dato, de tamaño Y bytes, sobre las líneas de datos desde la memoria caché a la memoria principal
* (OP3) Escritura del dato dentro de la memoria principal.

Nota: X e Y son parámetros que hacen referencia a la figura anterior

1.4.5.3     Frecuencia de aciertos (%)

La frecuencia de aciertos se calcula como sigue:

f = (aciertos/accesos)\*100

donde,

aciertos=aciertos de caché

accesos=accesos a una determinada caché

1.4.5.4     Frecuencia de fallos (%)

La frecuencia de fallos se calcula como sigue:

f = (fallos/accesos)\*100

donde,

fallos=fallos de caché

accesos=accesos a una determinada caché