textOsFtextTOsFliningLFliningTLFtextosflininglftabulartabproportionalpropsuperiorSup superiorSup

fontspechyperref

PROGRAMA DE SIMULACIÓN DEL PROTOCOLO DE COHERENCIA MESI

Jesus Pimentel Cruz , Patricia Pérez Romero, Israel Rivera Zárate

Centro de Innovación y Desarrollo Tecnológico en Cómputo
Av. Juan de Dios Batiz S/N, C.P. 07738, México, D.F., México

correo@ejemplo.com

Referencia de este artículo [1].

RESUMEN

La diferencia que existe entre el tiempo de ciclo de operación del procesador y el tiempo de acceso a memoria cada vez es mayor. El rendimiento de los procesadores se ha venido incrementando aproximadamente un 60 % cada año debido a la reducción del tiempo de ciclo de reloj y al incremento del número de instrucciones ejecutadas por ciclo (IPC). Sin embargo, el tiempo de acceso a las memorias DRAMS sólo mejora un 10 % por año, aunque la capacidad se duplica cada año y medio, según la Ley de Moore. Para reducir esta diferencia de tiempos se utiliza una organización de memoria jerarquizada con el objetivo de que el nivel cercano al procesador (cache) almacene temporalmente el contenido de la memoria principal que se prevé pronto será utilizado.

Los factores que afectan el rendimiento son: el tiempo necesario para obtener un dato de la cache y el número de accesos que se resuelven directamente desde la cache. Este trabajo se centra en incrementar la frecuencia de aciertos y reducir el tiempo medio de acceso en la cache sin incrementar el tiempo de ciclo del procesador, manteniendo dentro de límites razonables la latencia de acceso. Usando la capacidad de predicción que presentan las referencias a memoria para guiar la gestión y acceso al primer nivel en caches de acceso secuencial, proponemos un esquema dinámico e inteligente para acceder a la cache de datos del primer nivel en un sistema jerarquizado.

Introducción

Este trabajo se enfoca en el desarrollo de una herramienta que sirva para la comprensión de la administración de la información dentro de un sistema jerarquizado de memoria como sucede en los sistemas con procesadores de propósito general y particularmente en la memoria cache. **Ver figura 1.**

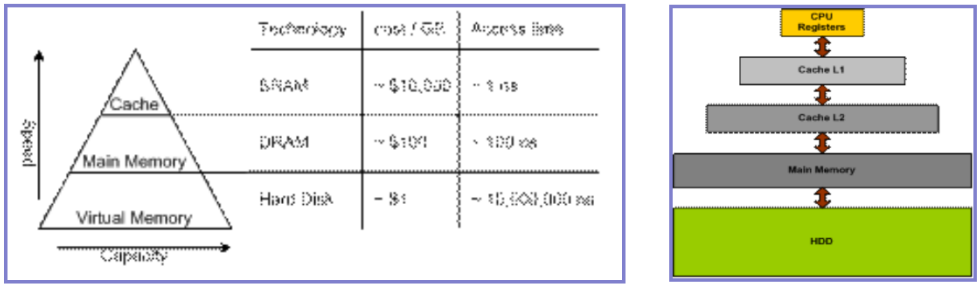


Figura 1: Jerarquía de memoria

En una memoria cache convencional, para leer un dato podemos distinguir varias fases: calcular, a partir de la dirección efectiva, un índice para acceder a la cache; leer las etiquetas; comprobar las etiquetas; y leer el bloque. Dependiendo de la función de mapeo utilizada en la cache, algunas fases pueden efectuarse en paralelo, aunque otras son estrictamente secuenciales. **Ver figura 2.** En una cache de mapeo directo, el bloque referido sólo puede encontrarse en una entrada; pero en una cache asociativa por conjuntos, varias entradas son candidatas para almacenar el bloque.

Grados elevados de asociatividad disminuyen la frecuencia de fallos, pero esto incrementa el tiempo medio de acceso a los datos debido al tiempo que se requiere para la conmutación de las posibles ubicaciones donde puede estar ubicada la información.Ahora bien, el tiempo de acceso y el grado de asociatividad del primer nivel de la jerarquía son un compromiso de diseño, ya que el acceso al primer nivel de cache está incluido en el camino crítico del procesador.

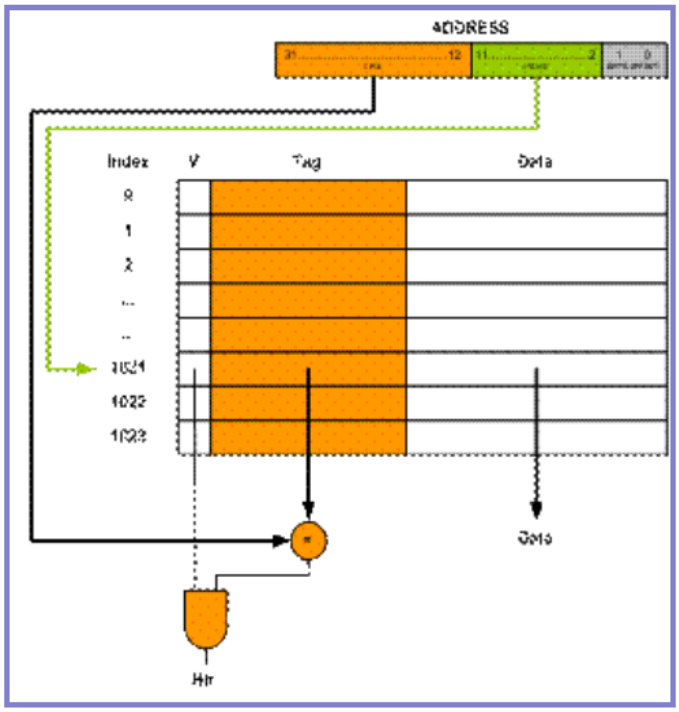


Figura 2: Campos de dirección de memoria

La búsqueda del bloque entre las posibles entradas de la memoria cache puede efectuarse de forma paralela o secuencial. En el caso de búsqueda paralela, después de leer las etiquetas correspondientes a los bloques del conjunto, su comparación con la etiqueta de la dirección efectiva permite determinar la posición en que se almacena el bloque en caso de acierto.

De forma concurrente a la lectura de etiquetas se pueden leer los correspondientes bloques de datos. En búsqueda secuencial, las distintas posiciones del conjunto de entradas en que puede estar ubicado un bloque se recorren de forma secuencial; entonces, el tiempo de acceso en caso de acierto no es constante: depende de la posición del bloque referido en la secuencia de búsqueda.

Debido a la propiedad de localidad temporal que presentan las referencias a memoria, la ubicación más recientemente usada presenta mayor importancia en el orden de búsqueda; por lo tanto, en un esquema de búsqueda secuencial, es importante determinar el orden de búsqueda de la ubicación más recientemente usada (MRU) antes de acceder a la cache.

Al efectuar una búsqueda secuencial de las distintas posiciones en que puede estar ubicado un bloque, el tiempo medio de acceso en caso de acierto es superior al tiempo de acceso a cache; por lo tanto, para reducir el tiempo medio de acceso puede predecirse la primera entrada que debe comprobarse.

Desde la aparición del transistor en diciembre de 1947 hasta nuestros días, la arquitectura de las computadoras ha experimentado un desarrollo a pasos agigantados. Mientras que el primer circuito integrado contenía aproximadamente 2000 transistores, el actual Pentium IV reúne en una sola pieza de silicio cerca de 55 millones de transistores; es 250 mil veces más poderoso, 50 mil veces más barato y mucho más pequeño. A los procesadores modernos se les llama comúnmente procesadores superescalares, y algunos de ellos son: el Pentium IV de Intel, el 21264C de Alpha, el UltraSparc III de SUN, Power 4 de IBM, el Athlon XP de AMD, el R14000 de MIPS, el PA-8700 de HP.

Los diseñadores de este tipo de procesadores mantienen el compromiso, adquirido desde hace cinco décadas, de continuar buscando mecanismos que mejoren su rendimiento. La enorme cantidad de transistores presente en los procesadores superescalares nos ha permitido incorporar técnicas de procesamiento cada vez más complejas; de igual forma, se ha integrado en el dispositivo el primer nivel de memoria, comúnmente llamado memoria cache (la mayoría de procesadores actuales disponen de un primer nivel de memoria cache, la cual se compone de dos módulos independientes, uno para datos y otro para instrucciones). **Ver figura 3.**

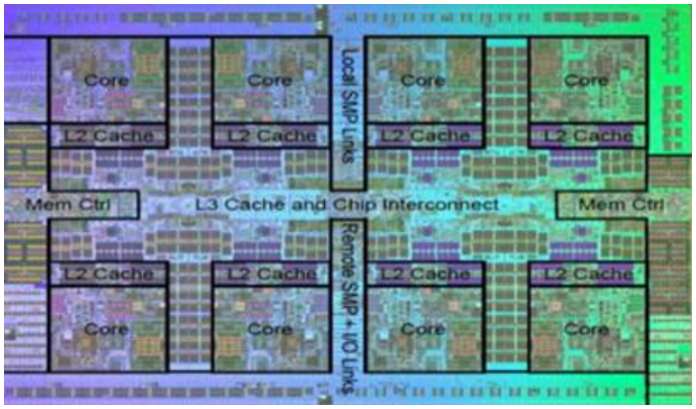


Figura 3: Diagrama Arquitectual IBM Power 7

Actualmente todos los procesadores de propósito general son procesadores segmentados con una estructura superescalar. La segmentación (Pipelining) surge con la idea de mejorar la productividad (en cuanto al número de instrucciones ejecutadas se refiere) de los procesadores, y consiste en dividir la ejecución de una instrucción entre las distintas etapas que intervienen en su proceso de ejecución, de tal forma que cuando la instrucción X libera la etapa N para continuar con la etapa N+1, la siguiente instrucción inicia su ejecución en la etapa N.

Esta idea se puede ver como una línea de ensamble de automóviles donde finalmente, cuando la cadena de montaje se encuentra llena, en cada etapa se produce un automóvil, y de esta forma se pueden estar construyendo varios automóviles a la vez. De igual manera, con la segmentación es posible ejecutar varias instrucciones a la mismo tiempo; esto acuñó el nombre de Paralelismo a Nivel de Instrucción (Instruction Level Parallel: ILP).

El tiempo en ciclos de reloj del procesador requerido para la ejecución de una instrucción, multiplicado por el número de instrucciones que conforman el programa, nos proporciona el tiempo que requiere un procesador para procesar dicho programa, el cual depende de tres factores: número de instrucciones necesarias para procesar el programa, ciclos de reloj necesarios para ejecutar una instrucción y período del ciclo de reloj.

Reduciendo cualquiera de los tres factores en la ejecución de un determinado programa, se mejora el rendimiento del procesador. Por ejemplo, la segmentación (Pipelining) es una innovación que permitió reducir el período del ciclo de reloj. Debido al comportamiento natural de los programas, las instrucciones y datos que se presentan de forma estática (código que entrega el compilador) en el código del programa al ser ejecutadas son accedidas más de una vez en forma dinámica (en ejecución); tal es el caso de las instrucciones de referencia a memoria.

Debido a que la mayoría de los programas no accesan a todo el código o a los datos de memoria uniformemente, es decir, favorecen una parte de su espacio de direcciones en cualquier instante de tiempo -por ejemplo, el cálculo de una matriz- es necesario mantener más cercana al procesador la información que se prevé que será utilizada en el corto tiempo y la información que se encuentre contigua a la información que se está procesando.

Cuando se accesa al mismo espacio de direcciones en períodos de tiempo relativamente pequeños, estamos hablando de la propiedad de localidad temporal; cuando se accesa a espacios contiguos de direcciones en períodos de tiempo relativamente pequeños, estamos hablando de la propiedad de localidad espacial. Estas propiedades, bien utilizadas, evitan un viaje demasiado largo (en tiempo de procesamiento) hasta el sistema principal de memoria, donde se encuentra almacenado todo el programa.

Con el objetivo de reducir el tiempo necesario para accesar a la memoria principal, tomando en cuenta las propiedades de localidad, se ha propuesto una memoria jerarquizada donde el nivel más cercano al procesador es más pequeño, más rápido, aunque de mayor costo y que usa una tecnología distinta a los niveles más lejanos. A este nivel más cercano al procesador dentro de la jerarquía se le denomina memoria cache. En los procesadores modernos (por ejemplo el Pentium III de INTEL), en el mismo circuito integrado se tienen los niveles de memoria cache, y esto ha mejorado el rendimiento general, pero aun así no se consigue cerrar la brecha que existe entre la velocidad de procesamiento y el tiempo de acceso al sistema de memoria.

La cache convencional es un buffer que almacena información; dependiendo de la forma en que se escribe o se lee dicha información, ésta puede ser de mapeo directo o asociativa de n vías; cuando n alcanza su valor máximo se dice que es completamente asociativa. La cache se presenta como un arreglo de dos dimensiones; la primera dimensión contiene el conjunto donde se encuentra el bloque de datos y la segunda, contiene el grado de asociatividad con que cuenta dicho conjunto (número fijo de posiciones donde puede ubicarse cada bloque); para asociatividad igual a uno, se dice que tenemos una cache de mapeo directo. Para asociatividades mayores a uno se dice que tenemos una cache asociativa por conjuntos de n vías.

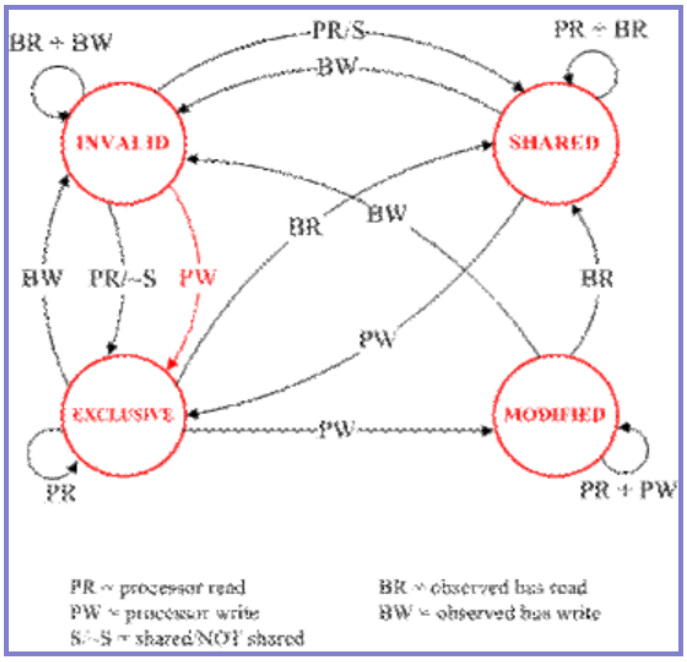
Cuando n alcanza su máximo posible se dice que la cache es completamente asociativa. Para mejorar el rendimiento de la cache, se puede especular entre cuál es la capacidad óptima y cuál es el grado de asociatividad que mejor rendimiento presenta, entre otras características. Para medir estos parámetros se usan como figuras de mérito: la tasa de fallos y el tiempo medio de acceso. Un fallo se presenta cuando la información que se busca no se encuentra en la cache y es necesario ir al siguiente nivel en la jerarquía de memoria para traer el dato requerido.

2. Desarrollo del Simulador de Protocolo de coherencia MESI

Cuando múltiples procesadores con cache integrada son colocados en un bus común compartiendo una misma memoria, es necesario garantizar que las cachés de los procesadores se mantengan en un estado coherente. Para lograrlo existe el protocolo MESI, dónde se manejan cuatro estados en los que una línea de cache puede estar:

Modified: Esta línea de cache ha sido modificada. La copia de la memoria es inválida.
Exclusive: Esta línea de cache tiene la única copia de los datos. La copia de la memoria es válida.
Shared: Más de una cache cuentan con una copia de esta línea. La copia de la memoria es válida.
Invalid: Esta línea de cache no es válida.

En la imagen siguiente se aprecia un diagrama de estados del funcionamiento del protocolo MESI.



Mantener la coherencia tiene dos componentes: lecturas y escrituras.

- En el caso de las lecturas, tener múltiples copias no es un problema. Lo que sí es importante es tener la copia más reciente, por lo que después de una escritura se ha de conseguir el nuevo valor.
- En el caso de las escrituras, se ha de tener acceso exclusivo.

En consecuencia, estos protocolos tienen que conocer todas las caches que comparten un objeto a escribir. La consecuencia de una escritura a datos compartidos es una invalidación de las copias o una actualización de las copias con el nuevo valor.

- En un fallo de lectura, todas las caches comprueban si tienen una copia del bloque solicitado y después toman la acción correspondiente como la de suministrar el dato a la cache que ha fallado.

- En una escritura, todas las caches comprueban si tienen una copia y entonces actúan invalidando su copia o actualizándola con el nuevo valor.

Como cada transacción del bus comprueba las etiquetas de las direcciones de las caches, se puede pensar que esto interfiere con el procesador. Interferiría si no se replica la parte de etiquetas de direcciones de la cache, no la cache al completo. Cuando hay una interferencia, el procesador se bloquea porque la cache está inaccesible.

- Invalidación por escritura: El procesador que realiza la escritura causa la invalidación de las copias en las otras caches antes de realizar la escritura; y después es libre de actualizar los datos locales hasta que otro procesador los necesite. El procesador que escribe envía una señal de invalidación al bus y todas las caches comprueban si tienen copia; en caso afirmativo, éstas invalidan el bloque que contiene la palabra. Este esquema permite múltiples lectores pero un solo escritor.

- Actualización por escritura: En lugar de invalidar cada bloque que está compartido, el procesador escritor transmite los nuevos datos por el bus a todos los procesadores; es decir, todas las copias se actualizan con el nuevo valor.

La actualización de escritura es como la escritura directa porque todas las escrituras van sobre el bus para actualizar copias del dato compartido. La invalidación de escritura utiliza el bus solamente en la primera escritura para invalidar las demás copias; las demás copias no aparecen en la actividad del bus.

Los multiprocesadores comerciales basados en cache utilizan caches de invalidación ya que se reduce el tráfico del bus y así permite más procesadores en un solo bus.

Protocolo de escritura:

Hay que distinguir tres casos:

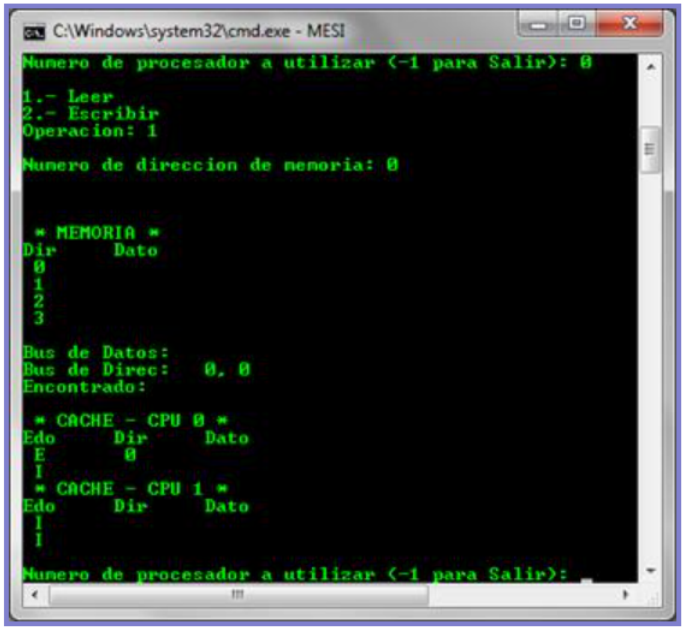
1. El bloque está presente y es la única copia (Exclusivo activado), la palabra se escribe de manera local.

2. El bloque está presente pero no es la única copia. Se envía primero un paquete de invalidación por el anillo para que las otras máquinas desechen sus copias. Cuando el paquete de invalidación regresa, el bit Exclusivo se activa y se procede a la escritura.

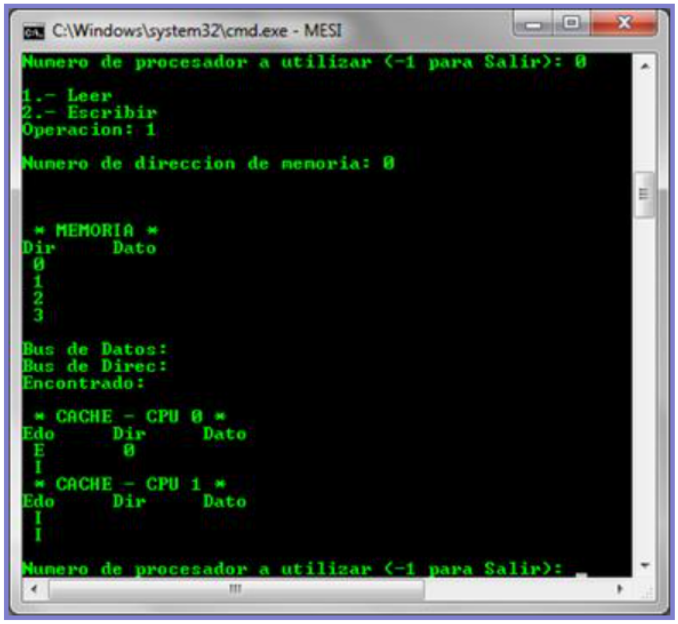
3. Si el bloque no está presente, se envía un paquete que combina una solicitud de lectura y una de invalidación. La primera máquina que tenga el bloque, lo copia en el paquete y desecha su copia. Todas las máquinas posteriores sólo desechan el bloque de sus caches.

Para el siguiente ejemplo de funcionamiento el sistema se representa con 2 CPUs con caches de 2 localidades y una memoria principal de 4 localidades. A continuación se presentan varios casos de acceso que se pueden presentar en este tipo de arquitectura.

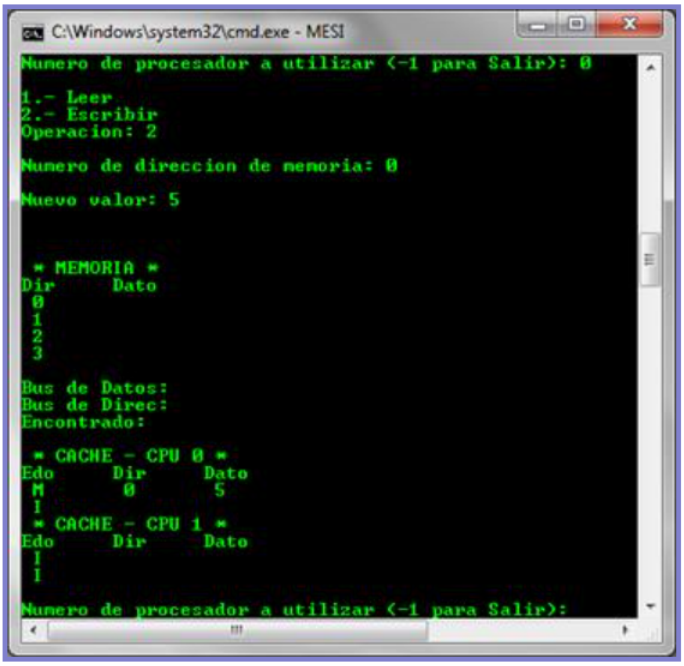
3. Prueba del Simulador y resultados



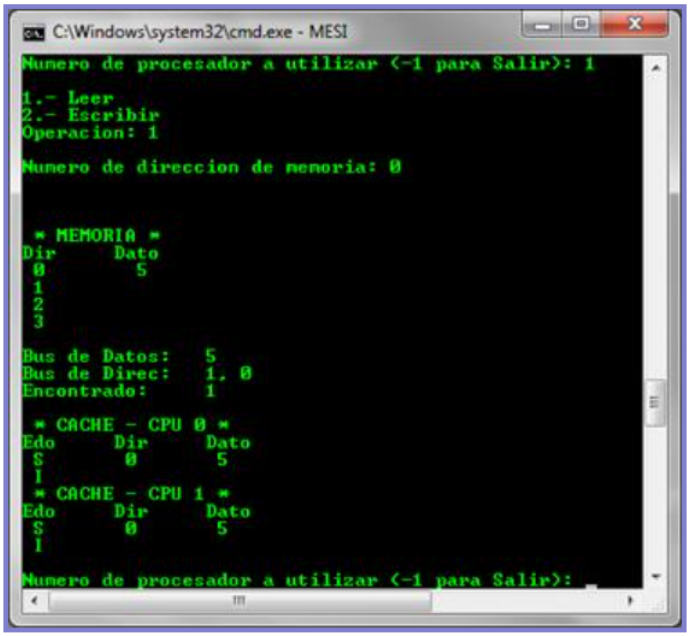
1.CPU 0 lee 0 - El CPU 0 lee la dirección 0 de la memoria (no compartida) - E



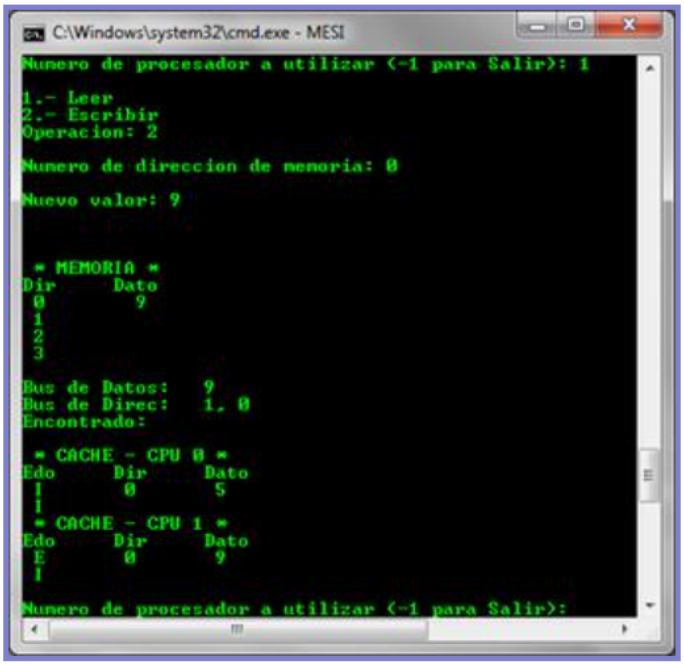
2.CPU 0 escribe 0 - El CPU 0 actualiza el valor de la dirección 0 en cache únicamente – M



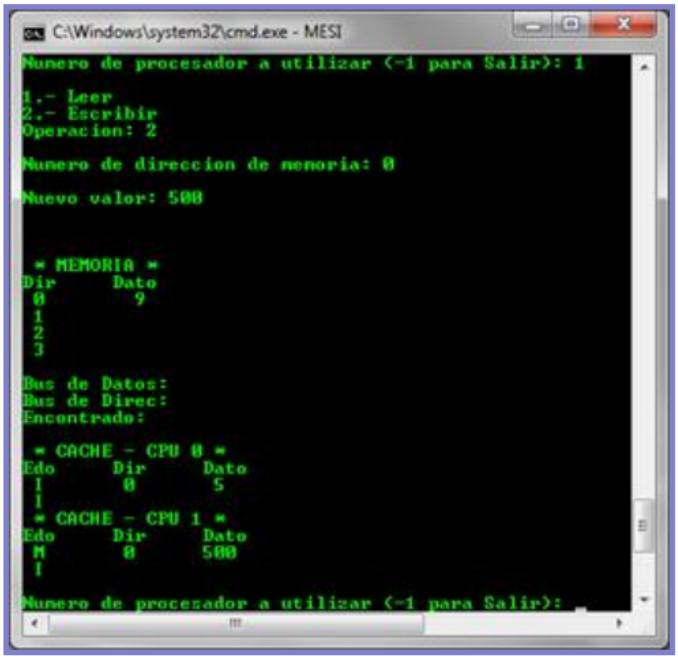
3.CPU 0 lee 0 - El CPU 0 lee la dirección 0 de la cache – E



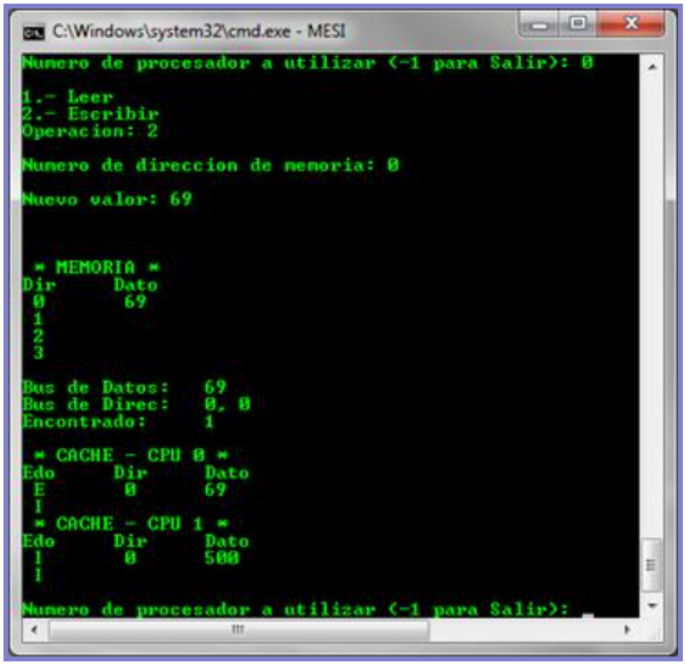
4.CPU 0 escribe 0 - El CPU 0 actualiza el valor de la dirección 0 en cache únicamente – M



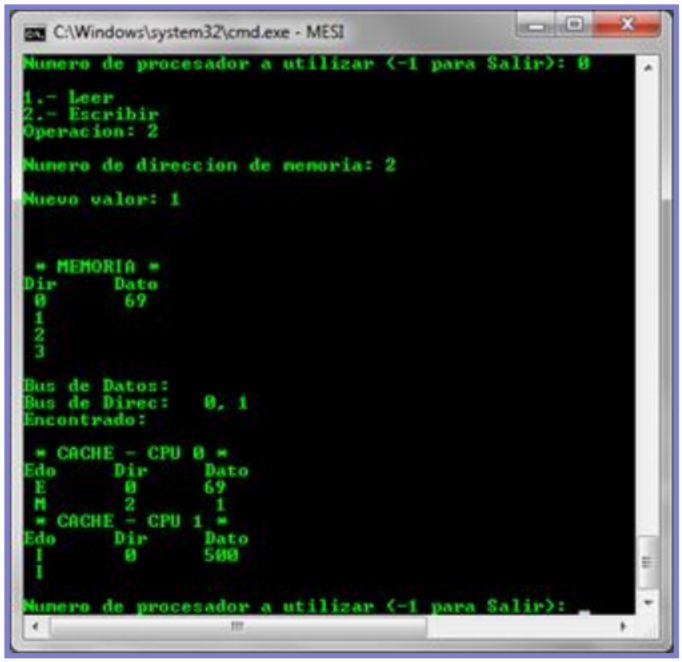
5.CPU 1 lee 0 - El CPU 1 lee la dirección 0, el CPU 0 interviene y actualiza el valor hacia cache y memoria – S



6.CPU 1 escribe 0 - El CPU 1 actualiza el valor de la dirección 0 en cache únicamente – M



7.CPU 0 escribe 0 - El CPU 0 lee la dirección 0, el CPU 1 interviene y actualiza el valor hacia cache y memoria (S), entonces el CPU 0 actualiza el valor de la dirección 0 en cache y memoria invalidando todas las demás caches con la dirección 0 – E



8.CPU 0 escribe 2 - El CPU 0 lee la dirección 2 de memoria (E) y actualiza su valor – M

4. Conclusiones

El presente simulador resultó una herramienta sencilla y de fácil uso que permitió ilustrar de manera muy evidente el movimiento de información entre los distintos elementos del esquema propuesto de memoria compartida. De igual forma, fue posible apreciar las transiciones que ocurren en el manejo de la coherencia de memoria para los diferentes casos propuestos. Resultó sumamente útil una herramienta de esta naturaleza ya que posibilitó el entendimiento de las acciones de lectura y escritura así como de invalidación propias de un protocolo de coherencia.

Este trabajo permitió la incorporación de sólo dos procesadores, así como el empleo de una memoria compartida y dos memorias cache, sin embargo como puede apreciarse en el código fuente anexo, resulta sencillo realizar una modificación que permita el uso de más elementos procesadores así como de almacenamiento principal y de cache e incluso de un aumento en el tamaño de los mismos. Debido a lo anterior, el trabajo constituye una base para futuras propuestas de estudio y de análisis en el campo de los protocolos de coherencia de esquemas de memoria distribuida y compartida, por lo que es posible a partir de esta primer propuesta evolucionar hacia otros protocolos que exijan una mayor complejidad tanto en número de elementos cómo de estados del protocolo.

Referencias

[1] Alan Jay Smith. Cache memories. Computing Surveys, 14(3):473-530, September 1982.

[2] Sally A. McKee and Wm. A. Wulf. Access ordering and memory-conscious cache utilization. In Proceedings of the First International Symposium on High-Perfotmance Computer Architecture, pages 253-262, January 1995.

[3] Disponible 16 de octubre 2012. `https://www.cs.tcd.ie/Jeremy.Jones/vivio/caches/MESIHelp.html`

[4] William Stallings, Computer Organization and Architecture Designing for Performarce, Fourth Edition Prentice Hall.

Referencias

[1] Albert Einstein, Isaac Newton, Marie Curie, Galileo Galilei, Charles Darwin ***(mayo - junio, 2025)*** *La teoría de la evolución biológica. Boletín UPIITA. año 19, (108) 2025* liga del artículo