Simulador de Caché Reconfigurable con Unidad de Anticipación (SICAREAN)

J.A. Pérez-Mata[1], E. Herruzo[2], J.I. Benavides[3] y O. Plata [4]

Resumen— **Este trabajo presenta una herramienta que simula una caché reconfigurable cuyos parámetros se pueden cambiar en tiempo de ejecución a través de una instrucción especial dentro del conjunto de instrucciones del sistema. Presenta un total de 298 combinaciones posibles para su configuración, atentiendo a cambios en la capacidad de la memoria caché, la asociatividad y el tamaño del bloque de caché. La caché incluye una unidad de Anticipación de datos que tiene como objetivo maximizar el rendimiento incrementando el número de aciertos, manteniendo dentro de límites razonables el tiempo de acceso. Además, la arquitectura simulada aprovecha la capacidad de predicción que le proporciona el compilador una vez que éste ha analizado el código fuente y lo ha transformado instrucciones máquina.**

Palabras clave— **Memoria caché, simulador software, unidad de anticipación, caché reconfigurable**

I. Introducción

En una caché asociativa por conjuntos, varias entradas son candidatas para almacenar un bloque. Así, un grado alto de asociatividad reduce la frecuencia de fallos pero incrementa a su vez el tiempo medio de acceso a los datos. Ahora bien, el tiempo de acceso y el grado de asociatividad de primer nivel en la jerarquía de memoria son un compromiso de diseño.

Para reducir el tiempo medio de acceso se puede predecir el siguiente bloque a ser accedido.

Una unidad de anticipación se encarga de cargar en cache, con cierta antelación, los bloques que el procesador va a necesitar. Si una búsqueda posterior tiene éxito, la memoria cache experimentará una reducción efectiva en el número de fallos. La anticipación de instrucciones se ha implementado con éxito en diferentes tipos de computadores, no ha ocurrido lo mismo con la anticipación de palabras de datos cuya eficiencia depende de un mayor número de factores de diseño, siendo este último muy sensible a la carga de trabajo. La unidad de anticipación puede ser púramente software o hardware. La idea en la que se basa la técnica software es muy simple: se centra en el estudio del programa durante el proceso de compilación; de esta forma, cada vez que el compilador detecta la inmedidata necesidad de un bloque, inserta una instrucción de prebúsqueda que provocará la carga anticipada de ese bloque. La mayoría de las arquitecturas de lenguage máquina actuales ya incorporan una instrucción de este tipo, y cada vez son más los compiladores que la usan. Sin embargo, presenta dos inconvenientes fundamentales [2]:

- El compilador no siempre es capaz de adivinar el comportamiento futuro de un programa analizando el código estáticamente.
- Las instrucciones que inserta el compilador aumentan el tamaño del código y ocupan recursos en el procesador al ser ejecutadas.

Por otro lado, la anticipación hardware estudia el comportamiento del programa durante su ejecución, casi siempre analizando la sencuencia de direcciones que genera. Si se detectan patrones regulares en esta sencuencia, que permiten preveer accesos futuros, se lanzan las peticiones de los bloques correspondientes sin intervención del procesador. Aunque teóricamente esto debería ser siempre así, en el acceso a estructuras de datos en cualquiera de sus dimensiones, los experimentos realizados muestran que no sucede así cuando se producen accesos consecutivos a las estructuras de datos en dimensiones diferentes a la de su almacenamiento. Tanto el análisis como el lanzamiento de las prebúsquedas requieren hardware específico, a veces de complejidad y coste significativos. Nuestro estudio se centra en este tipo de metodología hardware.

El simulador, `SICAREAN`, recibirá un conjunto de instrucciones que serán previamente optimizadas por el compilador con el objetivo de minimizar el número de conflictos con la memoria caché del sistema. Es de suponer que en la transformación de código, el compilador utilizará técnicas para optimizar la localidad de referencia, reduciendo así el número de conflictos con la memoria caché. En nuestro caso se ha contemplado array padding para maximizar el uso de la caché.

La mayor parte de las referencias a las estructuras de datos o arrays se producen en los bucles anidados. Las referencias a arrays en estos bucles anidados pueden realizarse en la misma dimensión en la que el array está almacenado (dimensión contigua) o en otra distinta (no contigua). La unidad de anticipación de datos que presenta este trabajo, a diferencia de los métodos de prebúsqueda convencionales, puede realizar la prebúsqueda en cualquier dimensión, no solamente en la dimensión contigua.

El trabajo presentado se estructura del siguiente modo: la sección II detalla la arquitectura simulada; la sección III describe los tipos de instrucciones contempladas en el sistema; a continuación (IV) se indican los posibles estados de la caché; en la sección V se define la unidad de anticipación de datos; la

[1]Dpto. de Arquitectura de Computadores. Universidad de Córdoba, e-mail: `i32pemaj@uco.es.`
[2]Dpto. de Arquitectura de Computadores. Universidad de Córdoba, e-mail: `eze@uco.es`
[3]Dpto. de Arquitectura de Computadores. Universidad de Córdoba, e-mail: `el1bebej@uco.es`
[4]Dpto. de Arquitectura de Computadores. Universidad de Málaga, e-mail: `oplata@uma.es`

sección VI describe distintos algoritmos implementados, y finalmente en la sección VII se indican algunas conclusiones al trabajo presentado.

II. Arquitectura simulada

La caché reconfigurable se integrará en un sistema con una CPU de 32 bits y una memoria principal de 256 Mbytes, que serán accesibles con una dirección de memoria de 26 bits. La caché tendrá una capacidad (`C`) máxima de 256 Kbytes, y una mínima de 256 bytes. La asociatividad (`W`) podrá variar entre 1 y 8 vías. El número de palabras (`L`) por vía oscilará entre 1 y 64 palabras. No se dan todas las combinaciones por que sólo se han permitido potencias de dos en la asociatividad y en el número de palabras por banco, facilitando así su manejo.

La memoria caché que se presenta aquí es una caché `unificada` que almacena las instrucciones y los datos juntos. La política de escritura elegida es `Post-escritura`. Se ha optado por una caché reconfigurable `en línea`, en el que primero se busca en caché y luego en memoria principal, en caso de fallo. La política de sustitución implementada es `pseudo-LRU`[6], que es una variante del algoritmo LRU, que requiere menos uso de la memoria. De este modo, la memoria caché reconfigurable implementa este algoritmo en el módulo LRU dependiendo de la asociatividad suministrada por el registro reconfigurable CRR.

SICAREAN se compone de los siguientes módulos (véase fig. 1):

- `Cache-MainMemory Interface (CMI)`: Se trata de un módulo controlado por la memoria principal que cuenta las palabras que se requieren para una vía de la caché en una operación de lectura/escritura.
- `Control Unit (CU)`: Controla todos los registros y módulos de la caché. Proporciona 3 microoperaciones al CPU: leer, escribir y reconfiguración.
- `Data Ram`: Contiene 8 SRAM'S de 32 bits y 8 Kpalabras cada una. Permite asociatividad 8.
- `Dynamic Mapping Module (DMM)`: La caché simulada incluye un módulo de mapeo dinámico (DMM) que asigna las direcciones emitidas por la CPU dinámicamente, en función del contenido del `CRR`. Este registro (Cache Register Reconfiguration) mantiene los parámetros de configuracion (C,W,L) en forma de máscaras que se pueden calcular fácilmente mediante la aplicación de las ecuaciones 1-2-3 tal que `label`, `set` y `word` son los bits para el rótulo, conjunto y palabra:

$$label = MAR_{longitud} - set - word \qquad (1)$$

$$set = \log_2 \frac{C}{W * L} \qquad (2)$$

$$word = \log_2 L \qquad (3)$$

- `Lru Module (LRUM)`: Módulo LRU (menos recientemente utilizado).
- `Tag Ram`: Contiene 8 SRAM'S de 32 bits y 8 Kpalabras cada una. Permite asociatividad 8. Almacena los rótulos asociados a cada palabra almacenada en caché.
- `Way Selection Module (WSM)`:Es el módulo de selección de vías, activa 1,2,4 u 8 SRAMS de rótulos según los 3 bits más significativos de la dirección de set o conjunto y de la asociatividad que toma el CRR.
- `Prefetch Module (PM)`, se trata del propio módulo de anticipación, es el que se encarga de precaptar las palabras desde memoria principal y los almacena en registros internos. Este módulo se analiza más profundamente en la sección V de este trabajo.

Además del sistema modular descrito en líneas anteriores, incorpora un conjunto de registros que se encarga del correcto funcionamiento de la caché.

Para mayor información sobre la arquitectura del simulador puede consultar el trabajo [7] publicado en el año 2004 que corresponde a la primera versión de este proyecto en el que aún no incorporaba ninguna unidad de anticipación.

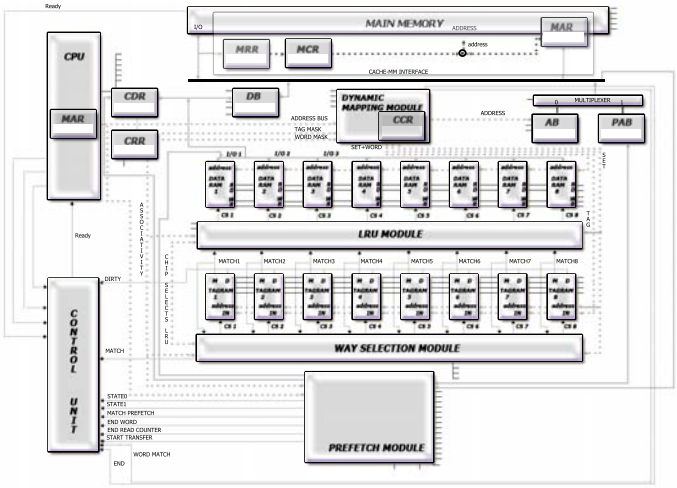

Fig. 1. SICAREAN. Interfaz principal

III. Repertorio de Instrucciones

Se distingue cinco tipos de referencias o instrucciones: 1) `referencia de lectura de instruccion` (LI), 2) `referencia de lectura de dato` (LD), 3) `referencia de escritura` (E), 4) `referencia de reconfiguración` (RC) y 5) `reconfiguración de stride` (RS). Esta última se encarga de la reconfiguración de los strides; entendiendo por stride la distancia en memoria de dos referencias a un array, realizadas por iteraciones consecutivas dentro de un bucle. La referencia `RC` se encarga de la configuración de la caché con relación a C,L,W.

`SICAREAN` trabajará sobre arrays que operen dentro de bucles perfectamente anidados. El proceso

de precaptación de las palabras se realizará como máximo sobre tres de esos arrays. Así, si el compilador detecta que el acceso a memoria sobre un determinado array se realiza siguiendo un patrón constante (en base al stride que este presenta) se lo indicará al simulador mediante la instrucción `RS`, que a partir de ese instante de tiempo `SICAREAN` comenzará a traerse los siguientes bloques, que se serán referenciados por la caché, paralelamente a la secuencia de instrucciones y sin afectar al funcionamiento normal de la ejecucion.

Cuando la `CPU` detecta una instrucción RS, envía directamente los 128 bits que van asociados a la misma. Este conjunto de bits contiene la siguiente información: 1) número de arrays que el módulo de anticipación debe tener en cuenta para la prebúsqueda posterior (máximo 3), 2) strides de cada uno de ellos y 3) dirección de comienzo de cada array. En el caso de que el CPU no detecte ninguna instrucción de este tipo, o que directamente la detecte pero el número de arrays indicados sea cero, entonces el módulo de anticipación quedará deshabilitado hasta que de nuevo sea detectada una nueva instrucción de reconfiguración de strides.

IV. Estados de la caché

HAY tres estados básicos que la caché puede producir:

- `Fallo de caché`, la dirección referenciada no se encuentra en caché ni tampoco en el módulo de anticipación.
- `Acierto en módulo de anticipación`, la caché no encuentra la palabra pero si se encuentra en el módulo de anticipación, no hace falta traerse el bloque desde memoria principal.
- `Acierto`, la palabra referenciada se encuentra en caché, no hace falta traerse el bloque desde memoria principal.

V. Unidad de Anticipación de datos

EXISTEN dos estrategias para almacenar los bloques que se van buscando anticipadamente: 1) `almacenamiento en memorias RAMs` y 2) `almacenamiento en registros individuales`. La primera de ellas sigue la misma filosofía que se emplea en el simulador implementado, después de llevar a cabo una minuciosa investigación se ha determinado que esta técnica no es efectiva ya que duplica el número de ciclos del proceso habitual de caché. En cambio, utilizando la segunda estratégia, si se utiliza como buffer un conjunto de registros se reduce significativamente el tiempo de acceso y transferencia a la caché del simulador, siendo bastante más eficiente que la anterior.

En la sección V-A se detallará el conjunto de registros que conforma este módulo de anticipación, seguidamente se explicará cómo se lleva a cabo la configuración de los strides para que se pueda efectuar prebúsquedas anticipadas (sección V-B) y finalmente se describirá su funcionamiento interno (Sección V-C).

A. Registros

El buffer de anticipación lo conforma 32 registros de 64 bits cada uno que almacenarán 1)`identificador de registro (6 bits)`, 2) `dirección donde se ubica la palabra en memoria principal (25 bits)` y 3) el `contenido de la palabra` que se ha traído al PM (32 bits). Junto a estos registros y a los que ya incluye la primera versión del simulador (véase [7]) están:

- `Array Reference Register (ARR)`, que indicará en todo momento el array que está siendo referenciado.
- `Counter Register (CR)`, que contabilizará el próximo registro de anticipación a utilizar.
- `Data Prefetch (DP)`, que almacenará el contenido de una palabra así como el identificador de registro donde esta se ubica.
- `Enable Prefetch Register (EPR)`, que indicará el número de strides que ha sido configurado previamente por el compilador.
- `New Address Register (NAR)`, que almacenará la próxima dirección a prebuscar para cada uno de los arrays que el compilador haya considerado previamente.
- `Prefetch Counters Registers (PCR, PCR_BLOCK`, que funcionarán como contadores de palabras y bloque respectivamente.
- `State Register (SR)`, que almacenará el número del estado en el que se encuentra el modulo de anticipación. Véase seccion V-C.4 para mayor información.
- `Strides`, lo conforman tres registros que almacenarán los valores de los strides que serán utilizados por el simulador.

Cabe destacar la existencia de dos registros buffer, `AB` y `PAB` que se encargarán de almacenar la dirección que contiene la siguiente palabra a ser leída/escrita desde la memoria principal. La unidad de control (CU) será la encargada de indicarle a la memoria principal donde se ubica la siguiente dirección que deba utilizar, para ello dispone de 40 microoperaciones que tendrán la función de garantizar el correcto funcionamiento de la caché reconfigurable.

B. Reconfiguración de strides

La reconfiguración de strides se llevará a cabo por medio de la instrucctión RS y realizará las siguientes operaciones, en base a los 128 bits recibidos desde la misma:

- `Inicialiar los contadores PCR/PCR_BLOCK a 0`
- `Inicializar el registro EPR con los bits [106:105]`
- `Inicializar los registros stride a partir de los bits [104:95], [94:85] y [84-75]`
- `Inicializar los registros NAR` de la siguiente forma:
  - NAR0 = Bits [74:50] + Bits [104:95]
  - NAR1 = Bits [49:25] + Bits [94:85]

– NAR2 = Bits [24:0] + Bits [84:75]
- `Inicializar el registro ARR, con el contenido de los bits [106:105]`

Todo este procedimiento se llevará a cabo mediante la instrucción `INIT` que lanzará la Unidad de Control.

C. Funcionamiento general

C.1 Detección de la palabra referenciada en el buffer

El módulo de anticipación recibe desde la CPU la dirección que está siendo referenciada, ésta es comparada con cada uno de los registros, de la misma forma como lo haría una memoria RAM y en caso de que exista coincidencia se lanzará una señal `MATCH_PREFTECH` para indicar a la Unidad de Control que la palabra referenciada está prebuscada. El registro DP almacenará el contenido e identificador de la palabra coincidente, para posteriormente (en caso de fallo en caché y acierto en anticipación) transmitirla al registro `CDR, Cache Data Register` que será el enlace entre la CPU y la memoria caché. La figura 2 ejemplifica esta situación.

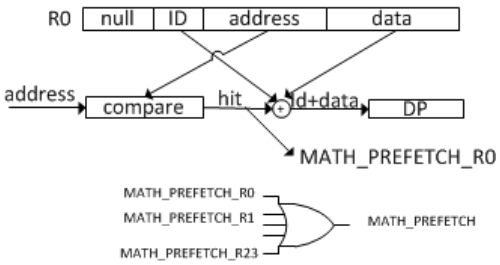

Fig. 2. SICAREAN. Detección de palabra en buffer

C.2 Transmisión del bloque a caché

Cuando exista un fallo en caché y un acierto en anticipación, la Unidad de Control activará la microoperación `LOAD_BLOCK` que se encargará de verificar si tenemos en el buffer todo el bloque. Solo en el caso de que tengamos todo el bloque de palabras dentro del buffer se activará la señal `START_TRANSFER` que le indicará a la Unidad de Control que este módulo está listo para entregar el bloque a caché. Por otro lado, si aún no tenemos ninguna palabra de la que conforma el bloque al que se hace referencia con la dirección que ha causado el fallo en caché, se activa la señal `START_TRANSFER` desasertada y será entonces cuando la caché acceda a memoria principal para traerse todo el bloque siguiendo el procedimiento normal de carga de caché. Siempre que se dispone a enviar una palabra a caché, se transmite el contenido del registro `DP` (la parte correspondiente al dato, los 32 bits menos significativos) al registro `CDR` que será el enlace entre la caché y el `CPU`.

C.3 Prebúsqueda de una palabra

El módulo de anticipación, `PM` tiene dos microoperaciones que le permiten almacenar una palabra transmitida desde la memoria principal: `MAINMEMORY_TO_MP` y `PAB_TO_MP`. Esta última microoperación se encarga de enviar la dirección que ha sido prebuscada de nuevo al PM.

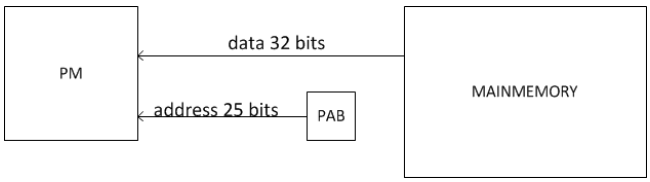

Fig. 3. SICAREAN. Transmisión al buffer desde memoria principal

El registro `CR` direcciona el próximo registro buffer que será escrito. Por tanto, en base a sus 5 bits y el uso de un demultiplexor doble se puede escribir en él el contenido tanto de la palabra enviada desde la memoria principal como la dirección transmitida desde el `PAB`.

Por otro lado, la Unidad de Control utiliza la microoperación `ARR_TO_PAB` para transmitir al registro `PAB` la siguiente dirección en la que se encuentra la palabra a prebuscar. Siempre que el algoritmo de anticipación se encuentre en su primer estado será cuando establezca comunicación con la memoria principal y le indique a esta mediante el contenido del `PAB` el dato que desea obtener. Gracias al registro `ARR` se puede saber en todo momento a cuál stride se le está haciendo referencia y por tanto cuál es el registro `NAR` que tiene la próxima palabra a prebuscar.

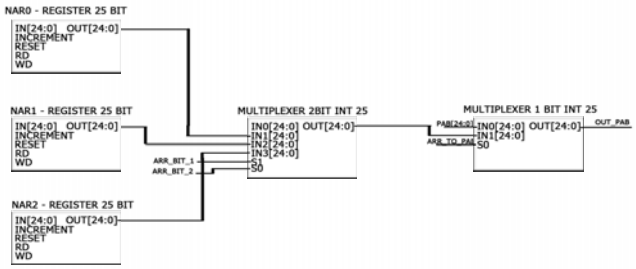

Fig. 4. SICAREAN. Cálculo de la siguiente dirección a prebuscar

C.4 Algoritmo de anticipación

Este algoritmo se ejecuta paralelamente junto a la ejecución normal de los algoritmos le lectura/escritura de caché, siempre que la unidad funcional no se encuentre operativa en ese momento. Cada vez que se trae una palabra desde memoria principal transcurre cuatro ciclos en el que en cada uno de ellos se efectuará una operación diferente de acuerdo al estado en el que se encuentre el módulo de anticipación en ese instante. Puede presentar 4 estados diferentes:

- `0`, la anticipación se encuentra deshabilitada.

- `1`, se establece la comunicación con la memoria principal, y si ésta se encuentra libre se le transmite la dirección a prebuscar.
- `2`, continúa el proceso de comunicación con la memoria principal. Mientras tanto se puede determinar si se ha cargado o no todo un bloque.
  - `Todo un bloque se encuentra en el buffer de anticipación`. Se actualiza el registro `NAR`, se reinicia el contador interno y se incrementa su estado. El registro `NAR`, para el stride actual, se actualiza en base al valor del registro `ARR` de la siguiente forma:

    ```
    Si ARR=01
    Si Stride0 > 1
    Actualizar NAR0 = PAB + Stride0
    Si Stride0 = 1
    Actualizar NAR0 = PAB-L+1+Stride0

      Si ARR=10
    Si Stride1 > 1
    Actualizar NAR1 = PAB + Stride1
    Si Stride1 = 1
    Actualizar NAR1 = PAB-L+1+Stride1

      Si ARR=11
    Si Stride2 > 1
    Actualizar NAR2 = PAB + Stride2
    Si Stride2 = 1
    Actualizar NAR2 = PAB-L+1+Stride2
    ```

    , tal que L es el tamaño de bloque.
  - `No se ha cargado todo el bloque`, en este caso se incrementa el registro `NAR` y vuelve al estado inicial 1.
- `3`, actualiza el registro `ARR` para direccionar al siguiente array. En el caso de que solo exista un único array, volverá a direccionar al mismo.

VI. Algoritmia

A. Algoritmo de Estado en Espera

Cuando la caché no está procesando lecturas, escrituras o reconfiguraciones que le manda la CPU, está en estado de espera. En este estado, el controlador de la caché debe activar la señal de ready para indicarle a la CPU que la caché está lista. Se podría habilitar una micro-operación para situar a la caché en alta impedancia pero se ha decidido dejar a la caché con su selección por defecto mientras está esperando una orden de la CPU. De esta manera la caché será más rápida ya que los búferes triestado que pueden poner alta impedancia en sus conexiones son relativamente lentos, además, el tiempo que pasa la caché desocupada no suele ser grande como para desconectarla. Mientras la CPU no le ordene otra cosa al controlador de la caché, seguirá ejecutando una y otra vez la micro-operación Ready.

B. Algoritmo de Reconfiguración

Este algoritmo reconfigura la caché y la memoria principal con los valores que tenga en el CDR. Una reconfiguración implica cambiar el tamaño del rótulo, del conjunto o/y de la palabra, por lo que conservar el contenido de la caché sólo acarrearía problemas de incoherencia. Es por ello que lo primero que hace el algoritmo es limpiar la RAM de rótulos, sin resetear la RAM de datos. Los datos antiguos se sustituirán por los nuevos sin más, implicando fallos forzosos en los primeros accesos Mientras se pone a cero el contenido de la RAM de rótulos, se introduce en el CRR y en el DB la palabra de reconfiguración. En el siguiente ciclo se comprueba que la memoria principal esté disponible para decirle que se reconfigure. Para finalizar se salta al estado de espera.

C. Algoritmo de Lectura

En la lectura de la caché, la CPU pone en el MAR la dirección de la palabra que necesita, lo que provoca que la lógica del DMM calcule el rótulo, el conjunto y la palabra, y la lógica del WSM realice lo propio con los CS's de las RAM's de rótulos. Después de que la CPU sitúe la dirección en el MAR activa la lectura de la caché. Para entonces la caché ya debe tener activas las RAM's de rótulos que se necesiten, direccionadas por el DMM, y los comparadores de las mismas estarán dando su resultado a la salida, por lo que nada más detectar la lectura, el controlador de la caché comprobará si existe acierto mirando la señal MATCH. En caso de acierto, la salida del comparador de la RAM de rótulos que ha sido acertada activará su RAM de datos correspondiente y en el ciclo que sigue a la comprobación del MATCH se copiará el dato que está en la dirección de conjunto + palabra que viene del DMM en el CDR, y se escribirán los bits LRU.

En caso de fallo, habrá que sustituir un banco del conjunto de memoria accedido. Con esta selección activada se comprueba el bit dirty del bloque seleccionado. Si el banco a sustituir en caso de fallo tiene el bit dirty a 0, se puede sustituir sin más. Para ello, basta con poner el nuevo rótulo en la RAM. Si el banco a sustituir en caso de fallo tiene el bit dirty a 1, dicho banco ha sido modificado por la CPU y hay que devolverlo a memoria antes de sustituirlo. Una vez que el rótulo anteriormente guardado se encuentra en el AB, se escribe el nuevo procedente del DMM en la RAM de rótulos que selecciona el LRUM. Entonces, el rótulo proveniente del MAR será igual al que está en la RAM recientemente escrita y su comparador activará la RAM de datos correspondiente. A continuación se comprueba si la palabra que ha dado fallo en caché, así como todo su bloque, se encuentra en los buffers de anticipación, en tal caso se transmite todo el bloque desde el módulo de anticipación a la RAM de datos. Si el módulo de anticipación se encuentra en proceso de carga del bloque, se espera a que éste finalice para posteriormente enviarlo por completo a la RAMs de datos de la caché. Por el contrario, si la palabra que ha dado fallo no se encuentra en el buffer de anticipación, se comprueba si la memoria principal está lista para que se escriba en ella y si es así se multiplexa la dirección

de palabra que llega desde el MAR por la del contador del DMM, y con esa dirección de palabra se escribe el contenido de la RAM de datos en el DB. Después de que la palabra se encuentre en el registro DB se avisa a la memoria principal para que la recoja. Se incrementa el contador de palabra del DMM y se comprueba si se llegó al final. Si no es así, se vuelve al punto del algoritmo en que se espera a que la memoria principal esté lista para mandarle otra palabra.

Después de almacenar el nuevo rótulo en la RAM de rótulos se procede a guardar en la RAM de datos el bloque de memoria que contiene la palabra que pide la CPU. El primer paso introducir en el registro AB la dirección base, se pone a 0 el contador y se modifican los bits LRU. Una vez que se encuentra la dirección en el AB se espera a que la memoria principal este lista. Posteriormente dejará en el DB una palabra y se vuelve a esperar a que esté disponible. Cuando lo esté signifiacará que el registro DB contiene la palabra buscada, si esta es la que solicitaba en el registro MAR se envía al CDR para que la CPU pueda acceder a ella. Si aún no se ha completado todo el bloque se vuelve a ejecutar de nuevo el algoritmo, en caso de que el bloque esté completo se activa la señal ready indicando al CPU que la caché está lista y que la palabra solicitada está en el registro CDR.

D. Algoritmo de Escritura

Cuando la CPU quiere escribir en la memoria principal debe poner en el MAR la dirección en la que quiere escribir, después se tiene que escribir en el CDR la palabra que quiere almacenar en la memoria principal, y al final, ejecutar el algoritmo de escritura. Si se produce un acierto se selecciona la SRAM de rótulos que ha producido el acierto con la microoperación MATCH SELECTION que nos proporciona el WSM y se pone el bit dirty a 1, ya que tras la escritura de la palabra el banco quedará modificado. Al existir un acierto, el rótulo de la dirección del MAR es el mismo que el que había en la RAM de rótulos, por lo que el escribirlo de nuevo no supone ningún error. En el mismo ciclo se escribe en la RAM de datos la palabra del CDR y se modifican los bits LRU.

Los fallos de escritura tienen un tratamiento similar a la lectura, salvo que se activa el bit dirty a 1.

E. Algoritmo de Anticipación

La unidad de anticipación en el `estado 0` queda deshabilitado y no realiza ninguna operación de prebúsqueda de datos. En el `estado 1` se espera a que la memoria principal esté disponible y cuando lo esté se le envía la dirección de la palabra que desea transferir. Posteriormente incrementa su estado y el contador `PCR` para verificar más adelante si se ha completado la precarga del bloque al completo.

En el `estado 2` vuelve a comprobar si la memoria principal está disponible. En caso afirmativo, la memoria principal ha terminado de enviar el dato al buffer correspondiente; por tanto, se reinicia su estado a 1 y se incrementa el contador de registro `CR`. En caso contrario, aún no se ha terminado de transferir el contenido de la direccion de memoria a la unidad de anticipación.

Finalmente, en el `estado 3` se verifica que la memoria principal está libre y por tanto la palabra prebuscada ya se encuentra en el correspondiente buffer, se actualiza el registro `ARR` en base al número de strides con el que el simulador esté trabajando (valor contenido en el `EPR`), para concluir este ciclo se incrementa el contador `CR` y se reestablece el estado a 1.

VII. Conclusiones

CON esta herramienta se ha pretendido desarrollar un concepto nuevo de simulador, que puede ser interesante para su aplicación con fines docentes; siendo un complemento muy valioso para los estudiantes, motivándolos y facilitando su enseñanza en la temática de la memoria caché.

El objetivo de `SICAREAN` se ha logrado, reduciendo significativamente la tasa de fallos así como el tiempo de procesador destinado al tratamiento de carga y almacenamiento de referencias de memoria.

Referencias

[1] B. Calder, D. Grunwald and J. Emer, *Predictive Sequential Associative Cache*, *Proc. 2nd HPCA, 1996.*

[2] K. Bolland, A. D., *Predicting and precluding problems with memory latency*, IEEE Micro, 1994, Vol. 14. nº 4. , 59-67.

[3] Y. Jégou, Temam Speculative prefetching In: ICS '93 Proceedings of the 7th international conference on Supercomputing

[4] John W. C. Fu, Janak H. Patel, Bob L. Janssens Stride directed prefetching in scalar processors Newsletter ACM SIGMICRO Newsletter Vol. 23 (1,2), 1992

[5] Gornish E.H Addaptive and Integrated data cache prefetching for shared-memory multiprocessors University of Illinois at Ubana-Champaign, 1989

[6] J. Handy *The Cache Memory Book*, San Diego, CA: Academic,1993, 0-12-322985-5.

[7] R. Quislant, Reconfigurable cache memory system. Design and simulation, M .S. thesis, University of Córdoba, Córdoba, Spain, 2004

[8] Sánchez F.J., González A. Recopilación sobre técnicas de prebúsqueda software Universitat Politècnica de Catalunya. Departement d'Arquitectura de Computadors, 1996

[9] Vanderwiel, S.P., Lilja, D.J. Data Prefetch Mechanisms Computing Surveys Vol. 32 (2) pp. 196-199., 2000

[10] Quislant R., Herruzo E., Plata O., Benavides J.I., Zapata E. The Cache Memory System Using a Reconfigurable Approach IEEE Transactions on education Vol. 51 (3) pp. 336-341., 2008

[11] Shi, L., Fan, X., Chen, J., Huang, X., Tian, H. Tolerating memory latency using a hardware-based active-pushing technique Proceedings - 2009 International Conference on Embedded Software and Systems, ICESS 2009 (5066675) pp. 407-411

[12] Kandemir, M., Zhang, Y., Ozturk, O. Adaptive prefetching for shared cache based chip multiprocessors Proceedings -Design, Automation and Test in Europe (5090768) pp. 773-778., 2009

[13] Ramos L.M., Briz J.L., Ibáñez P.E., Viñals V. Low-Cost Adaptive Data Prefetching roceeding Euro-Par '08 Proceedings of the 14th international Euro-Par conference on Parallel Processing

[14] Ding Y., Zhang W. Loop-based Instruction Prefetching to Reduce the Worst-Case Execution Time IEEE Transactions on computers Vol. 59 (6), 2010