DESARROLLO DE UN SIMULADOR PARA ALGORITMOS DE SUSTITUCIÓN Y ESCRITURA DE MEMORIA CACHÉ

DEVELOPMENT OF A SIMULATOR FOR CACHE REPLACEMENT ALGORITHMS AND WRITING

Esther Viridiana *Vázquez Carmona*
Instituto Politécnico Nacional/Centro de Innovación y Desarrollo Tecnológico en Cómputo, México
evazquezc@1801alumno.ipn.mx

Rodrigo Vázquez López
Instituto Politécnico Nacional/Centro de Innovación y Desarrollo Tecnológico en Cómputo, México
rvazquezl1800@alumno.ipn.mx

Juan Carlos Herrera Lozada
Instituto Politécnico Nacional/Centro de Innovación y Desarrollo Tecnológico en Cómputo, México
jlozada@ipn.mx

Recepción: 4/junio/2019 **Aceptación:** 3/junio/2020

Resumen

Actualmente los ordenadores trabajan cada vez más rápidos, esto se debe a que, al realizar ciertas tareas, el acceso a la información solicitada por el procesador se adquiere desde la memoria RAM en el menor tiempo posible, sin embargo, mientras se realiza este proceso se pierde la oportunidad de ejecutar otras tareas. La memoria caché es la encargada de solucionar el problema a través de la manipulación de los bloques de memoria, la actualización y escritura en caché utilizando los algoritmos de sustitución. El presente documento presenta el desarrollo de un simulador para algoritmos de sustitución y escritura de memoria caché, dicho simulador está basado en lenguaje HTML5 y JavaScript. El objetivo de este software es servir como apoyo en la impartición de clases para docentes a nivel licenciatura y comprender fácilmente el funcionamiento de la memoria caché, en la que intervienen las políticas de ubicación, extracción, reemplazo y escritura.

Palabras clave: simulador, memoria caché, algoritmos de sustitución, procesador, políticas de reemplazo.

Abstract

Currently computers are working faster and faster, one the reasons is when performing certain tasks, access to the information requested by the processor is acquired from the RAM in the shortest time possible, however, while this process is taking place the opportunity to perform other tasks is lost. The cache memory is responsible for solving the problem through the manipulation of memory blocks, updating and caching using replacement algorithms. This document presents the development of a simulator for replacement algorithms and cache writing, said simulator is based on HTML5 and JavaScript language. The objective of this software is to serve as support in teaching classes for teachers at the undergraduate level and easily understand the operation of the cache, which involves the policies of location, extraction, replacement and writing.

Keywords: *simulator, cache memory, replacement algorithms, processor, replacement policies.*

1. Introducción

Actualmente la memoria caché, ha adquirido gran importancia en el ambiente de los microprocesadores, ya que se encarga de acelerar las lecturas y escrituras que se requieren para el procesamiento de las instrucciones en memoria.

Simuladores

Existe software que puede llegar a reproducir el comportamiento de un sistema, como un simulador de carreras, vuelo, música, entre otros. Sin embargo, también existen simuladores con el propósito de servir como una herramienta didáctica para la comprensión de algunos temas en docencia. Por ejemplo en lo que se refiere a procesadores y memoria se encuentra el simulador 8085, que es un simulador gráfico y depurador para el microprocesador de Intel 8085, que se usó en las plataformas Linux y Windows, también se encuentra el simulador de algoritmos de planificación de la CPU [8085software.tripod, 2019], que consiste en mostrar el funcionamiento de los aspectos de la CPU en un sistema de multiprogramación, el simulador denominado: “Simulación bajo memoria compartida de un sistema

distribuido" el cual simula una memoria compartida [Navarro,1997], realiza una evaluación preliminar de un nuevo protocolo para memoria compartida distribuida. En cuanto a memoria caché existen un número de simuladores entre ellos se encuentran: Simulador para el apoyo docente en la enseñanza de las arquitecturas ILP con planificación dinámica y estática, que hace referencia a las arquitecturas de paralelismo a nivel de instrucción y palabra de instrucción extensa [Castilla,2004]. Existe también otro simulador llamado: Simulador de sistemas de Memoria caché en multiprocesadores simétricos que se basa en un modelo construido según los principios básicos de la arquitectura de estos sistemas, éste simulador, denominado SMPCache, posee una interfaz gráfica completa y funciona en un entorno PC bajo Windows 98/95 [Vega,2000]. Así como estos existen un gran número de simuladores que sirven como apoyo en las distintas asignaturas relacionadas con microprocesadores y memoria.

El uso de simuladores en la docencia como en asignaturas de Arquitectura de Computadores resulta ser una práctica muy habitual debido a la complejidad de conectar la teoría de muchos de los temas que se tratan con la experiencia práctica [Barrachina,2015].

El motivo para el desarrollo de este proyecto de investigación fue implementar algoritmos de sustitución dentro de un simulador a través de un software que se encargara de servir como apoyo al docente para dar a conocer las principales características, el funcionamiento y ayudar a comprender mejor los fundamentos de las políticas que integran el diseño de una memoria caché en cursos como arquitectura de computadoras a nivel licenciatura [Alfaro,2010].

Memoria caché

El sistema de memoria caché es complejo y se ha convertido en una amplia gama de posibilidades para su investigación, una de ellas, es que se han creado múltiples herramientas como simuladores [Rodríguez,] que este caso permite la reproducción de ciertas configuraciones para conceptos fundamentales y obtener resultados con ciertas limitaciones. La memoria caché es una memoria rápida que se encuentra entre la unidad central de procesamiento (CPU) y la memoria principal (RAM) su

objetivo es lograr que la velocidad de la memoria principal aumente [Camacho,2003]. La caché contiene una copia de partes de la memoria principal para que posteriormente el procesador lea desde caché y si no se encuentra lo busca en la memoria principal. Sin embargo, no todas las memorias caché se diseñan de la misma manera, pero sí coinciden con los criterios básicos para el diseño una memoria caché [Stallings, 2005] los cuales se muestran en la tabla 1.

Tabla 1 Elementos básicos de diseño para una memoria caché.

Elemento de diseño de caché	Contenido
Función de correspondencia	• Directa • Asociativa • Asociativa por conjuntos
Algoritmo de sustitución	• LRU • FIFO • LFU • ALEATORIO
Política de escritura	• Escritura inmediata • Postescritura • Escritura única
Tamaño de línea	
Número de Cachés	• Uno o dos niveles • Unificada o partida

El diseño de caché implica políticas como la función por correspondencia la cual trabaja para definir un espacio para un bloque de memoria principal, que posteriormente se trasladará a memoria caché [Fan, 2015]. En particular el objetivo de la correspondencia directa es asegurar que cada bloque de memoria principal le corresponda una línea de caché, mientras que la función asociativa permite que cada bloque de memoria se pueda cargar en cualquier línea de la caché, la función asociativa por conjuntos es la combinación de las funciones anteriores en la que la caché se divide en v conjuntos de k líneas.

Algoritmos de sustitución

Cuando la línea de caché se encuentra saturada se requiere desplazar hacía una nueva línea de caché, es aquí donde entran en juego los algoritmos de sustitución, estos se utilizan exclusivamente en las correspondencias asociativas para introducir un nuevo bloque una vez que se ha llenado la caché, dichos algoritmos son implementados en hardware [Qian, 2018].

En el algoritmo Aleatorio simplemente se escoge un bloque al azar sin ningún problema, sin embargo, en el algoritmo de sustitución LRU (*Least Recently Used*) se utiliza un bloque que no ha sido referenciado ya en algún tiempo, a este algoritmo se le asocian contadores a las líneas de caché para saber cuando es un acierto o un fallo, cabe destacar que un acierto sucede cuando se hace referencia a un marco de bloque. En el algoritmo FIFO (*First In First Out*) sucede algo muy diferente ya que se sustituye el bloque que ha estado durante más tiempo en la memoria caché [Hendrantoro,2015]. En tanto, el proceso para actualizar la memoria principal se basa en políticas de escritura, las cuales se dividen en: la política de escritura directa y la política de escritura llamada postescritura, durante la escritura inmediata o directa, las operaciones se realizan sobre la memoria caché y la memoria principal, en cambio con la postescritura las actualizaciones se realizan directamente en memoria caché. En consecuencia, conforme aumenta el tamaño del bloque de memoria, la información debe trasladarse hacía la memoria caché para reutilizar la mayor cantidad de datos, por lo tanto, requiere de un mayor tamaño, a esto se le llama tamaño de línea, la cual puede variar entre los 8, 64 y 128 bytes.

Hoy en día el número de cachés puede variar, en el que entran en juego los niveles de caché, el uso de caché unificada y cachés separadas, la caché unificada distribuye proporcionalmente los bloques, en cambio las cachés separadas se forman a partir de dos cachés, una para datos y otra para código, la desventaja es que el rendimiento es menor debido a que cada caché podría contar con bloques libres [Castro,2007].

En este trabajo se presenta un simulador de memoria caché en el que implementaron los principales componentes de diseño mencionados en la tabla 1. Este simulador permite reproducir el comportamiento de la memoria con los diferentes algoritmos de sustitución como: LRU, FIFO y RANDOM, con diferentes tamaños de caché, además de manejar ciertos números de conjuntos y una cadena de entrada. Esta herramienta tiene por objetivo servir como apoyo a alumnos de licenciatura, la ventaja es comprender mejor el funcionamiento de las políticas de memoria caché, considerando que es un soporte visual este método de enseñanza, puede ayudar a los alumnos a consolidar ideas a través de la visión.

2. Métodos

En esta sección se tratará de explicar el proceso de desarrollo del proyecto, así como algunas pruebas realizadas. Comenzaremos por describir los conceptos principales utilizados para el desarrollo de software, posteriormente se hablará acerca de cada una de las partes que integran la interfaz gráfica del simulador.

El simulador reproduce la forma en que se lleva a cabo la actualización de la caché con los datos, posteriormente se divide la caché en marcos de bloque o líneas de igual tamaño, dicho bloque será el intercambio de información entre la memoria principal y la caché. En la figura 1 se puede observar de manera gráfica como es el comportamiento entre la memoria principal y la memoria caché en la vida real, lo cual pretende ser semejante a nuestro simulador desarrollado (Vega,2000).

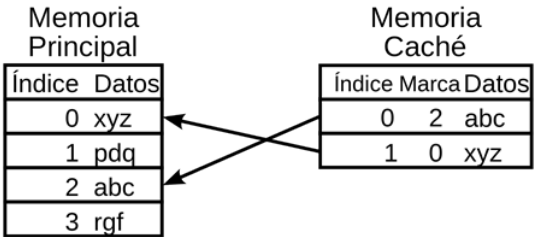

Figura 1 Funcionamiento de la memoria caché y memoria principal en una computadora.

La elaboración de este software se basó principalmente en el uso de HTML5, lenguaje básico de World Wide Web y JavaScript que es un lenguaje interpretado orientado a objetos [Meizhen,2013]. La razón de por la se eligieron estos lenguajes es, por que suelen ser compatibles con los distintos navegadores, y trabajan en conjunto creando páginas dinámicas y llamativas en las que se puede interactuar más con los usuarios [Next_U,2019]. La figura 2 muestra el diagrama de flujo que caracteriza dicho simulador, en él se puede observar cada uno de los procesos que lo integran.

El proceso comienza ingresando los datos de prueba, posteriormente se procede a leer tamaño de la memoria caché y el número de conjuntos que tendrá disponibles ya que con dichos valores se generan las tablas internas que simulan la memoria caché. Enseguida, se lee el conjunto de datos de entrada, los cuales simulan el bloque de datos que serán almacenados temporalmente dentro de la memoria

caché, por lo cual, cada elemento del conjunto de entrada es analizado al momento de realizar una inserción dentro de la memoria considerando dos casos: el espacio de memoria caché que contiene espacios libres y el espacio de memoria se encuentra lleno. Para el primer caso, si la memoria caché cuenta con espacio disponible el elemento se inserta en alguno de los espacios libres y se procede a analizar el siguiente elemento del conjunto de datos. En el segundo caso, si la memoria no cuenta con espacios disponibles se requiere hacer uso de alguna de las políticas de reemplazo de caché, para lo cual es necesario verificar la política elegida previamente por el usuario (LRU, FIFO o RANDOM), ejecutando el procedimiento correspondiente a dicha política. En ambos casos, si el elemento del conjunto de datos a insertar ya existe dentro de la memoria, dicha casilla se marca y se considera como un 'hit'.

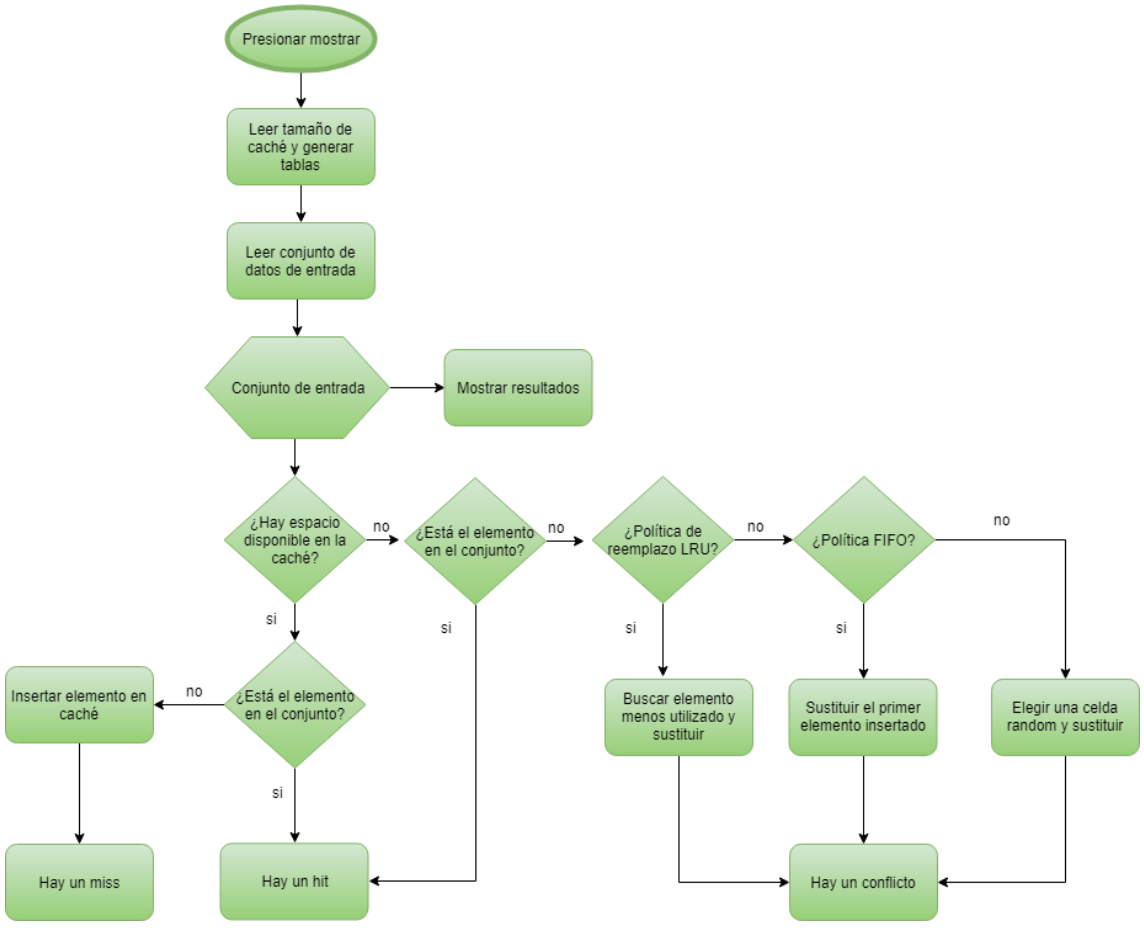

Figura 2 Diagrama de flujo del funcionamiento del simulador.

El ingreso de datos contiene parámetros como tamaño total de la memoria caché que van desde 4 hasta los 256 MB, conjuntos de tamaño 1, 2, 4, 8 hasta 256, cabe destacar que el tamaño de la caché no debe ser menor al tamaño de conjuntos, además de introducir estos parámetros se debe ingresar el algoritmo de sustitución que se probará y una o varias cadenas de entrada que pueden ser introducidas por el usuario o con simplemente activar una casilla que ingresará valores aleatorios. La figura 3 muestra la interfaz en la que el usuario debe introducir cada valor para generar el comportamiento de la memoria caché.

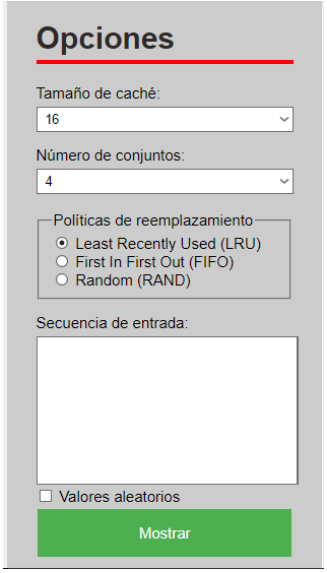

Figura 3 Interfaz gráfica correspondiente al ingreso de datos.

El contenido de la memoria caché muestra diferentes celdas en el que se observa la información ingresada anteriormente y las direcciones correspondientes de cada marco de bloque. La figura 4 muestra la sección en la que se observa la interfaz correspondiente.

Adicionalmente se muestran la cantidad de operaciones realizadas, así como una de gama de colores que indican los fallos en la memoria caché que se presentaron durante el proceso, estos fallos pueden ser fallo obligatorio que se presenta cuando el bloque debe leerse de la memoria independientemente del diseño de la memoria caché, el fallo por capacidad que ocurre cuando el caché es demasiado pequeño para contener todos los datos utilizados simultáneamente y el fallo por conflicto que

se producen cuando varias direcciones se asignan al mismo conjunto y desalojan los bloques que aún son necesarios, por último se muestra un apartado con un número total de cada operación realizada por la memoria caché por ejemplo el número total de consultas, el total de errores, una tasa de pérdida y una tasa de operaciones con éxito.

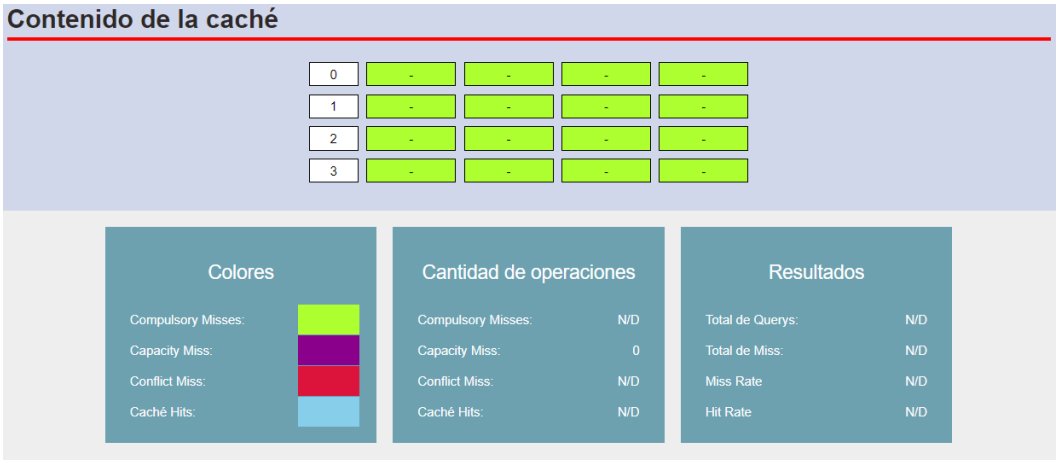

Figura 4 Simulación del contenido y operaciones de la memoria caché.

En la figura 5 se muestra la simulación con algunos parámetros al azar utilizando el algoritmo FIFO para observar el comportamiento de una memoria caché y como es que va guardando la información en los bloques. Los datos ingresados se muestran en la figura 6.

Figura 5 Contenido de memoria caché al utilizar parámentros al azar.

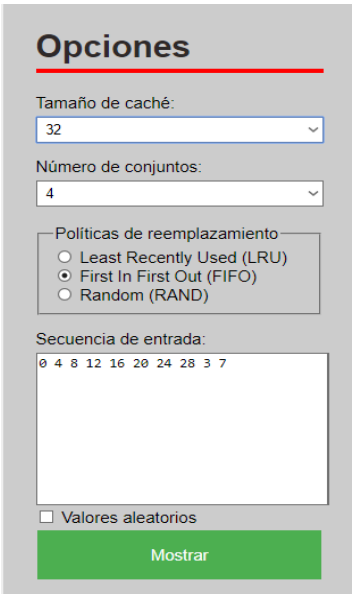

Figura 6 Datos introducidos para demostrar el funcionamiento con el algoritmo FIFO.

3. Resultados

Como se puede observar en la figura 5 se tiene que con un tamaño de caché de 32 Mb e ingresando la cadena con los números 0, 4, 8, 12, 16, 20, 24, 28, 3 y 7, con un número de conjuntos de tamaño 4 se obtiene que existen 10 fallos obligatorios puesto que bloque se está leyendo de la memoria independientemente de la memoria caché, En el apartado de resultados se muestra el porcentaje de consultas caché que se dan cuando los datos no se encuentran en dicha memoria, entonces la memoria principal emite una consulta, el número de errores, la tasa de pérdida y la tasa de aciertos la cual se refiere al porcentaje de llamadas a la caché que tienen éxito en la recuperación de los datos deseados (Molero,2001).

En la figura 7 ocurre lo contrario puesto que se está ingresando una secuencia de entrada con los dígitos 1, 2, 3, 4, 5, 6, 7, 8 y 9 con un tamaño de caché de 8 MB un con 4 números de conjuntos, en él se puede observar que existe un error de conflicto en el que varias direcciones se asignan al mismo conjunto y quitan bloques que aún son necesarios en este caso el 1 fue el bloque desalojado. En la figura 8 se presenta un caso en el que se ingresó la secuencia 1, 2, 3, 4, 5, 1, 2, 7, 8, 9, 10, 11, 12, 12, 13, 15, 17, 22, 2, 5, 27, 29 y 24 con un tamaño de caché de 4 Mb y 1 conjunto. Como se puede observar, arroja 15 errores de capacidad, ya que la caché es

demasiado pequeña para contener todos los datos utilizados simultáneamente. Finalmente, en la figura 9 se observa el caso en el que se ingresó la secuencia de entrada con los datos 1, 2, 3, 4, 5, 1, 2, 3, 4 y 5 con un tamaño de caché de 4 MB y un número de 2 conjuntos con el algoritmo LRU, en esta prueba se activaron la casilla de aciertos puesto que el contenido de la dirección se encuentra en un bloque ubicado en la memoria caché.

Figura 7 Contenido de memoria caché con errores de conflicto.

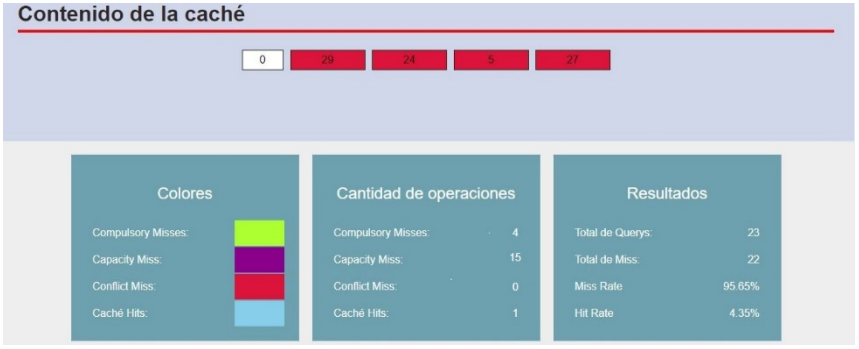

Figura 8 Contenido de memoria caché con errores de capacidad.

Figura 9 Contenido de memoria caché con aciertos.

El simulador cuenta con las principales características de diseño de una memoria caché, como se mencionó anteriormente incluye tres de los algoritmos de sustitución, sustitución (en un futuro se integrarán los algoritmos faltantes y sus derivaciones), maneja diferentes tamaños de caché y los números de conjuntos necesarios que requiere, sin embargo aún se encuentra en una fase preliminar, puesto que falta por añadir los tipos de escritura y los niveles de caché, unificada y dividida así como el algoritmo de sustitución LFU, pero en general está cumpliendo con el objetivo puesto que se está observando el comportamiento de los algoritmos de sustitución y sobre como realiza la escritura sobre los bloques de marco en memoria caché.

La memoria principal es un dispositivo que trabaja lentamente en comparación con la memoria caché, debido a su tamaño. La memoria caché contiene copias de fragmentos de la memoria, para que posteriormente el procesador adquiera la información desde está y evite largos periodos de tiempo para dicho proceso.

3. Conclusiones

Se propuso un simulador de memoria caché, en el que se lograra implementar los 3 de los algoritmos de sustitución, así como indicar cada uno de los errores que conlleva el proceso de escritura y almacenamiento en dicha memoria. Con el objetivo de ser un apoyo en materias afín de licenciatura y favorecer el aprendizaje de los alumnos.

Como ya se indicó el software está escrito en los lenguajes HTML5 y JavaScript, con el cual se logró desarrollar un simulador de memoria caché en el que se implementaron los algoritmos de sustitución LRU, FIFO y ALEATORIO, así como los errores (capacidad, obligatorio y conflicto) que se dan durante el almacenamiento de datos. Para ello se llevaron a cabo pruebas en las que se ingresaron diferentes secuencias de datos con diferentes algoritmos, tamaños de caché y varios conjuntos para observar el comportamiento de cada uno de ellos y poder comprobar resultados a través de un simulador previo (López,2015). Las pruebas realizadas tuvieron como objetivos principales: probar casos particulares como el funcionamiento de los algoritmos de sustitución, así como obtener

resultados que se puedan comparar con otros simuladores. Para el primer punto se verificó el comportamiento de los algoritmos en casos particulares, variando el tamaño de la memoria y la cantidad de conjuntos, esto con la finalidad de comparar los resultados obtenidos que se encuentran en la literatura. El segundo punto permitió demostrar el correcto funcionamiento del simulador, además de servir como herramienta de depuración en caso de encontrar algún error en el código. En trabajos futuros se pretende añadir los tipos de escritura y los niveles de caché, unificada y dividida, así como el algoritmo de sustitución LFU y sus derivaciones, se pretende evaluar el simulador en clase, posteriormente se encontrará disponible en línea.

4. Bibliografía y Referencias

[1] Alfaro, F. J., Bermúdez, A., García, P. J., & Sánchez, J. L. (2010, July). SJM: Un simulador de jerarquías de memoria orientado a la docencia de arquitectura de computadores. In XVI Jornadas de Enseñanza Universitaria de la Informática (pp. 397-404). Universidade de Santiago de Compostela. Escola Técnica Superior d'Enxeñaría.

[2] Barrachina Mir, S., Fabregat Llueca, G., Fernández Fernández, J. C., & León Navarro, G. (2015). Utilizando ARMSim y QtARMSim para la docencia de arquitectura de computadores.

[3] Camacho Nieto, O., Villa Vargas, L. A., Díaz de León Santiago, J. L., & Yáñez Márquez, C. (2003). Diseño de Sistemas de Memoria Cache de Alto Rendimiento aplicando Algoritmos de Acceso Seudo-Especulativo. Computación y Sistemas, 7(2), 130-147.

[4] Castilla, I., Moreno, L., Sigut, J., González, C., & González, E. J. (2004). SIMDE: Un simulador para el apoyo docente en la enseñanza de las arquitecturas ILP con planificación dinámica y estática. Proceedings of X Jornadas de Ensenanza Universitaria de la Informática (JENUI 2004), 505-508.

[5] Fan, Z., Haghdoost, A., Du, D. H., & Voigt, D. (2015, October). I/o-cache: A non-volatile memory-based buffer cache policy to improve storage

performance. In 2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (pp. 102-111). IEEE.

[6] Castro, J. L. A. (2007). 8. Cache memory coherence protocol for distributed systems. Revista Técnica de la Facultad de Ingeniería Universidad del Zulia, 30(2).

[7] Hendrantoro, G., & Affandi, A. (2015, May). Early result from adaptive combination of LRU, LFU and FIFO to improve cache server performance in telecommunication network. In 2015 International Seminar on Intelligent Technology and Its Applications (ISITIA) (pp. 429-432). IEEE.

[8] López O, J., Rivera Zárate, I. & Hernández Bolaños, M. (2015). Simulador de Políticas de Ubicación, Sustitución y Escritura de Memoria Caché. [online]: http://www.boletin.upiita.ipn.mx/index.php/ciencia/607-cyt-numero-47/1115-simulador-de-politicas-de-ubicacion-sustitucion-y-escritura-de-memoria-cache.

[9] Meizhen, W., Yanlei, S., & Yue, T. (2013, August). The design and implementation of LRU-based web cache. In 2013 8th International Conference on Communications and Networking in China (CHINACOM) (pp. 400-404). IEEE.

[10] Molero, X., Rodas, Á., Pont, A., Sahuquillo, J., & Valiente, J. M. (2001). Diseño de experiencias prácticas sobre memoria cache. Actas de las VII Jornadas de Enseñanza Universitaria de Informática, Jenui, 3-8.

[11] Navarro, J. E. (1997). Simulación bajo memoria compartida de un sistema distribuido que simula memoria compartida. In III Congreso Argentino de Ciencias de la Computación.

[12] Next_U. (2019, junio 4) Ventajas y desventajas de Javascript [online]: https://www.nextu.com/blog/conoce-las-ventajas-y-desventajas-de-javascript/.

[13] Qian, L., Mei, Z., Pang, H., Yu, J., Zhu, G., Chen, H., & Bu, M. (2018, August). understanding Cache Policy to Evaluate Write Performance with Hierarchical Hardware. In 2018 10th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC) (Vol. 2, pp. 308-311). IEEE.

[14] Rodríguez, M. A. V., Pulido, J. A. G., & Pérez, J. M. S. Enseñanza de Arquitecturas de Memorias Caché mediante Simuladores.

[15] 8085software.tripod (2019, junio 4) Microprocessor® 8085® software simulator kit Available at: http://8085software.tripod.com/index.html.

[16] Vega, M. A., Martín, R., Zarallo, F. A., Sánchez, J. M., & Gómez, J. A. (2000). Smpcache: simulador de sistemas de memoria cache en multiprocesadores simétricos. XI Jornadas de Paralelismo. Granada.

[17] Stallings, William. (2005). Organización y arquitectura de computadores (7 ª edición). Madrid (España): Pearson.