**Universidad Católica Andrés Bello**

**Facultad de Ingeniería**

**Escuela de Ingeniería Informática**

**Estructura del Computador**

**Prof. Laura Navarro**

**Memoria Caché**

**Integrantes:**

* **De Oliveira José Francisco**
* **Drachenberg Christian**

**Introducción**

Hoy en día las computadoras son por mucho más complejas que hace medio siglo, actualmente se trabajan con grandes cantidades de información, al tener tantos datos que manejar o almacenar los procesos para la gestión de dicha información se hacían más lentos, por esta razón nace la jerarquía de memorias, por medio de la cual nuestra computadora accede a distintos niveles o jerarquías de memoria para acceder a los datos que se necesiten en un momento determinado. Uno de los niveles más comunes, importantes y que es utilizado actualmente es el perteneciente a la memoria caché, donde se busca guardar la información a la cual se accede con mayor frecuencia para así facilitar los procesos de búsqueda.

En este documento explicaremos los fundamentos teóricos que permiten comprender el funcionamiento de este tipo de memoria, para ello hablaremos sobre las funciones de correspondencia o mapeo, las políticas de reemplazo, las fallas y su clasificación y por último analizaremos los resultados obtenidos en los casos de prueba en el proyecto desarrollado.

**Marco Teórico**

**Funciones de mapeo**

Existen tres funciones de mapeo que son utilizados por el computador para el manejo de memoria en la actualidad, estos son:

* **Mapeo directo:** este consiste en agarrar cada bloque de la memoria principal y transformarlo en un único bloque de la memoria cache. Cuando la CPU genera una dirección de (n bits) para acceder a una palabra de la memoria principal, su formato desde el punto de vista de la memoria caché se divide en tres campos: **tag, línea y byte**. Su principal desventaja es que cualquier bloque dado tiene asignada una posición fija en la memoria cache. Así, si ocurre que un programa efectúa el mismo bloque de la memoria cache estos bloques se estarán moviendo continuamente entre la memoria cache y la memoria principal, con la consiguiente pérdida de rendimiento del sistema.

**tag** = bmp – linea – byte

**línea** = log2(bmc)

**byte** = log2(tb)

Donde:

**bmp:** bloques en memoria principal.

**bmc:** bloques en memoria caché.

**tb:** tamaño de cada bloque.

**línea:** representa la línea de memoria caché donde será ubicado el bloque.

* **Mapeo asociativo:** esta función permite arreglar el inconveniente de la función de mapeo directo que se mencione anteriormente, ya que permite que un bloque de la memoria principal se cargue en cualquier bloque de la memoria caché. Cuando la CPU genera una dirección para acceder a una palabra de la memoria principal, su formato desde el punto de vista de la memoria caché se divide en dos campos: tag y byte. Cuando se lee un nuevo bloque, con esta función hay que decidir generalmente por cual se va a sustituir en la memoria cache.La principal desventaja de este procedimiento es la necesidad de una circuitería bastante compleja, para examinar en paralelo los campos de “tags” de todos los bloques de la memoria cache. Pero tiene como ventaja que dos bloques con localidad de referencia temporal no competirán por la misma línea (bloque en la caché).

**COLOCAR FORMULA Y X VAINA**

* **Mapeo asociativo por conjunto:** es un compromiso que trata de beneficiarse de los dos métodos anteriores. La memoria caché se divide en un número “nc” de conjuntos compuestos por una cantidad de bloques “tc”. Cada bloque de la memoria principal puede ubicarse en uno de los tantos bloques de la memoria caché que componen cada conjunto. Esta función se puede ver como una función directa entre los bloques de la memoria principal y los conjuntos de la memoria caché y como una función asociativa entre los bloques de un mismo conjunto. Cuando el CPU genera una dirección para acceder a una palabra de la memoria principal, su formato desde el punto de vista de la memoria cache se divide en tres campos: tag, conjunto y byte.

**tag** = log2(bmp)-tc

**conjunto** = log2(nc)

**byte** = log2(tb)

Donde:

**bmp:** bloques en memoria principal.

**tc:** tamaño en bloques de cada conjunto.

**nc:** número de conjuntos.

**tb**: tamaño de cada bloque.

**Políticas de reemplazo de bloques en memoria caché**

En el manejo de la memoria caché se utilizan diferentes políticas para reemplazar bloques en dicha memoria, estas son:

* **Less recently used (LRU):** Éste es el algoritmo más utilizado. Tiene la ventaja de que luego de cada referencia, se actualiza una lista que indica cuan reciente fue la última referencia a un bloque determinado. Si se produce un desacierto, se reemplaza aquel bloque cuya última referencia se ha producido en el pasado más lejano, diversas simulaciones indican que las mejores razones de acierto se producen aplicando este algoritmo.
* **First in first out (FIFO):** En este caso los bloques dentro del conjunto están ordenados de acuerdo a la secuencia con que son cargados. Cuando se debe reemplazar un bloque se elimina de la memoria caché aquel que fue cargado en primer lugar; los bloques siguientes son removidos en el mismo orden. El principal problema de éste algoritmo se presenta cuando un bloque es requerido repetidamente.
* **Less frecuently used (LFU):** Es similar al algoritmo FIFO solo que en lugar de reemplazar al último que entró, se reemplaza el bloque que ha sido menos referenciado.
* **Random (RAND):** Un bloque “b” es elegido al azar entre bloques que forman el conjunto en el cual se ha producido una falla. Esta política es contraria al principio de localidad por la cual no es recomendada; sin embargo, algunos resultados de simulaciones indican que al utilizar este algoritmo se obtienen razones de aciertos superiores a los anteriormente mencionados. Posee las ventajas de que por un lado su implementación requiere un mínimo de hardware y no es necesario por otro lado almacenar información alguna para cada bloque.

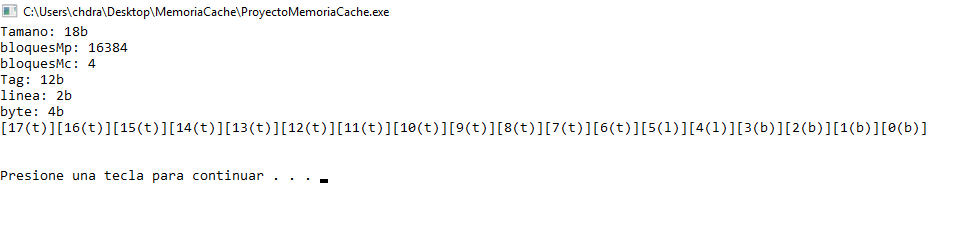
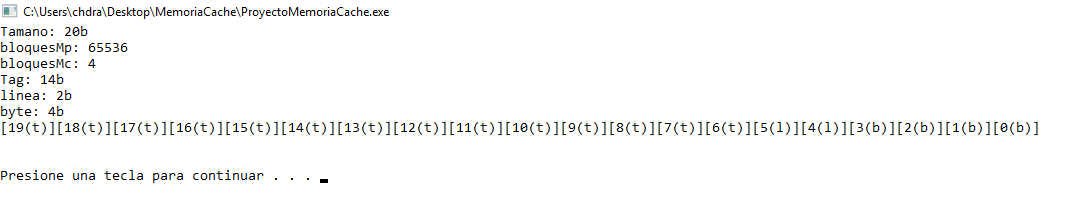
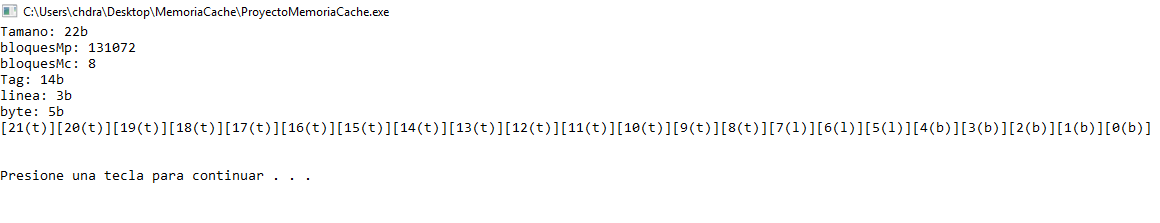
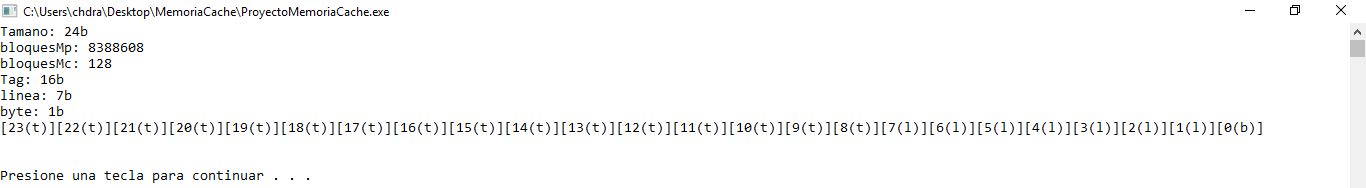
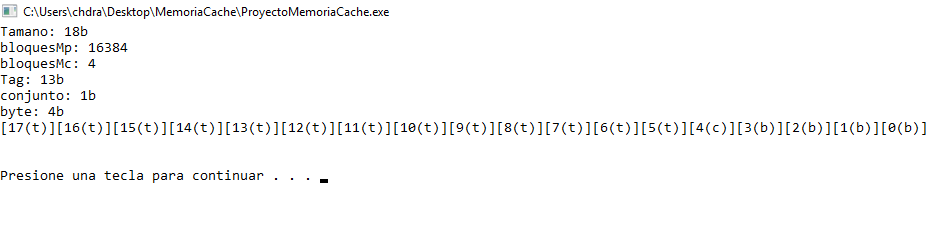
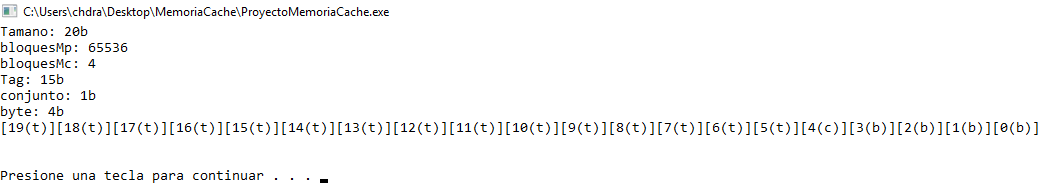
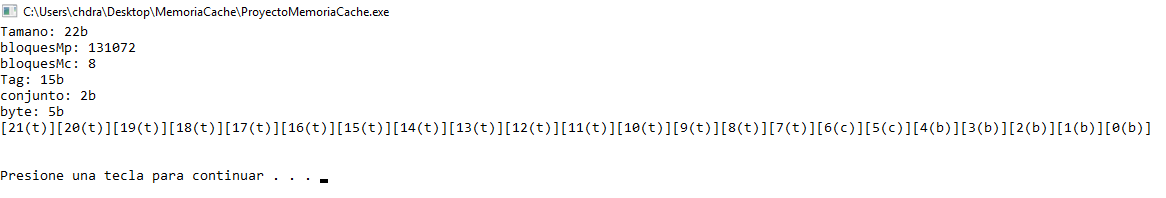
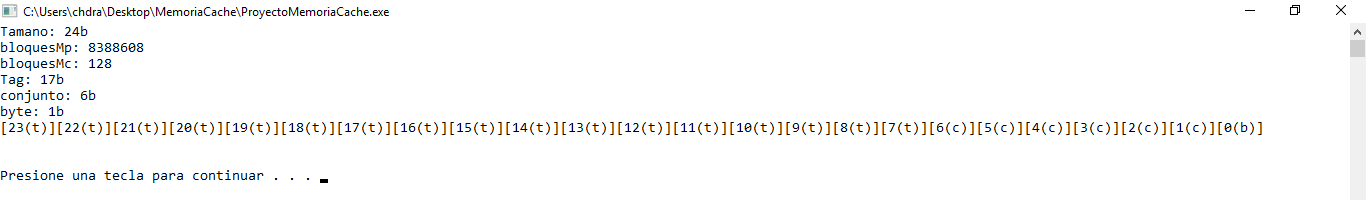
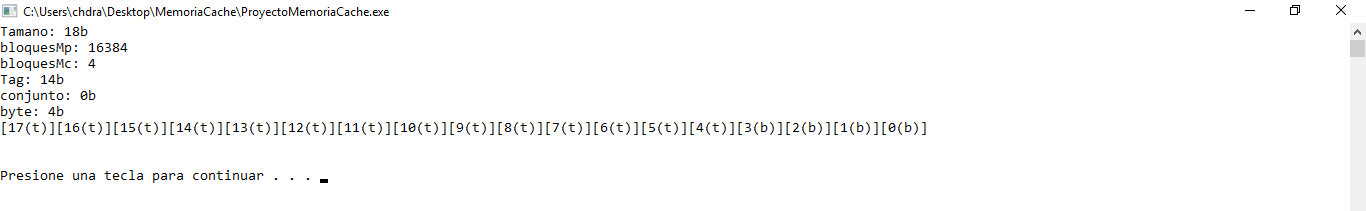
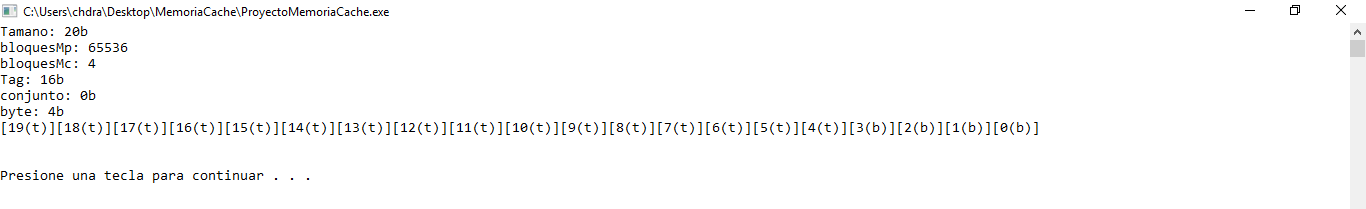
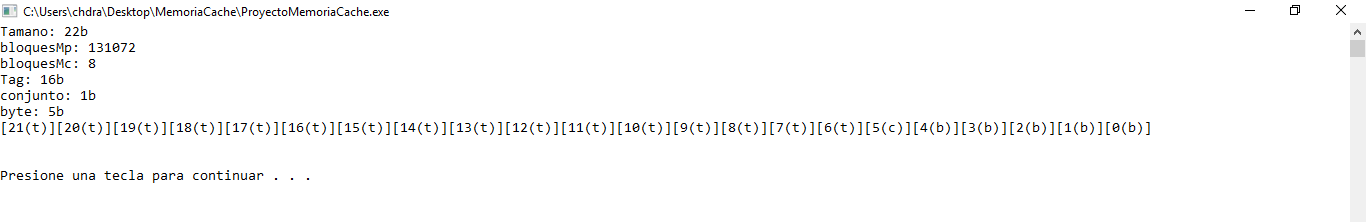
**Fallas y Tipos de fallas:**

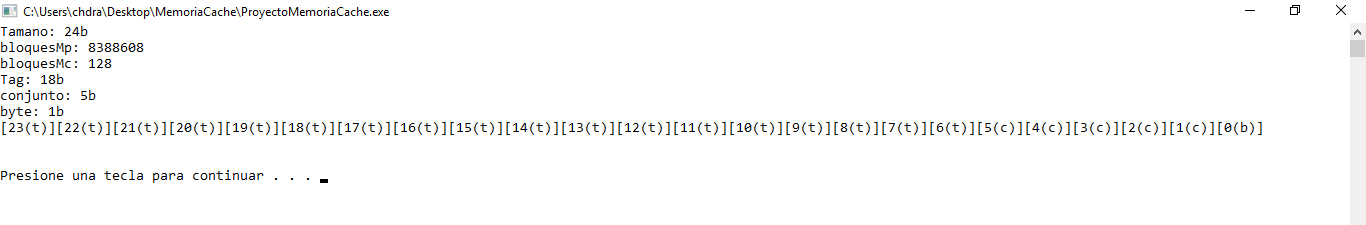
Cuando una dirección de memoria es referenciada y no está contenida en el los bloques correspondientes a la memoria caché se dice que ocurre una falla o miss, dependiendo de las circunstancias bajo las que esto ocurra se pueden dividir las fallas en:

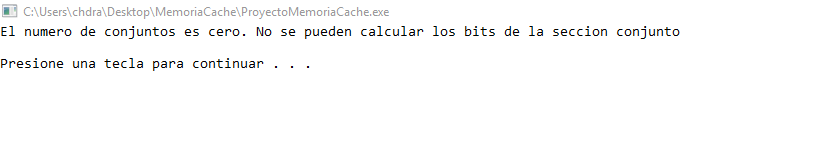
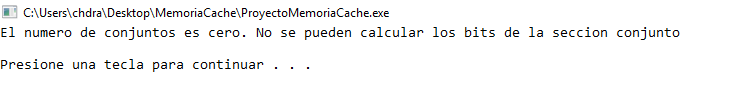
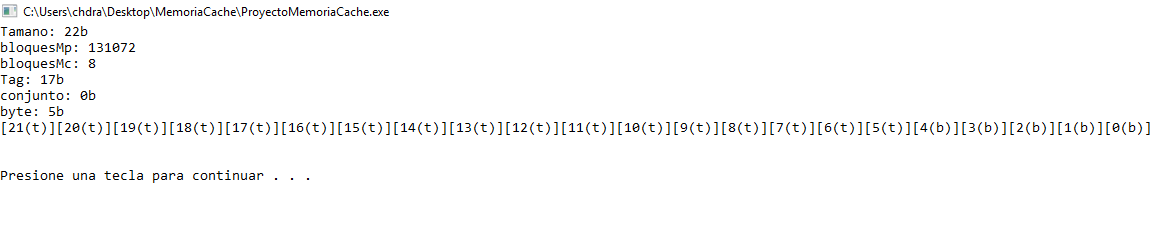
* **Falla Forzosa:** Ocurre cuando la dirección referenciada no se encuentra en los bloques del conjunto correspondiente de la memoria caché, es decir que el primer acceso al bloque no se encuentra en la memoria caché y debe buscarse en la memoria principal.
* **Falla por Conflicto:** Ocurre cuando dos bloques buscan el mismo espacio dentro del mismo conjunto en la memoria caché.
* **Falla por Capacidad:** Ocurre cuando la memoria caché no puede contener todos los bloques que se necesitan durante la ejecución de un programa.

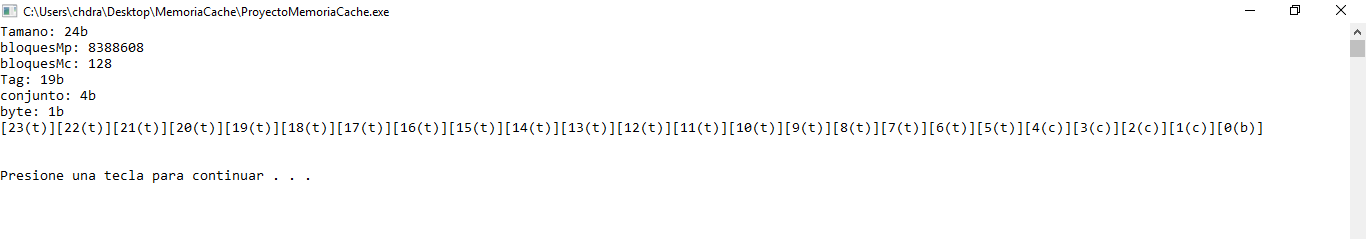
**Resultados Obtenidos y Análisis**

**Módulo 1:**

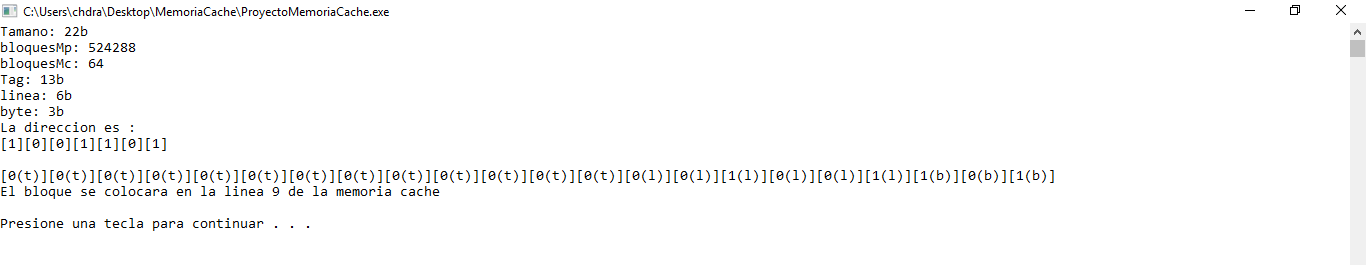
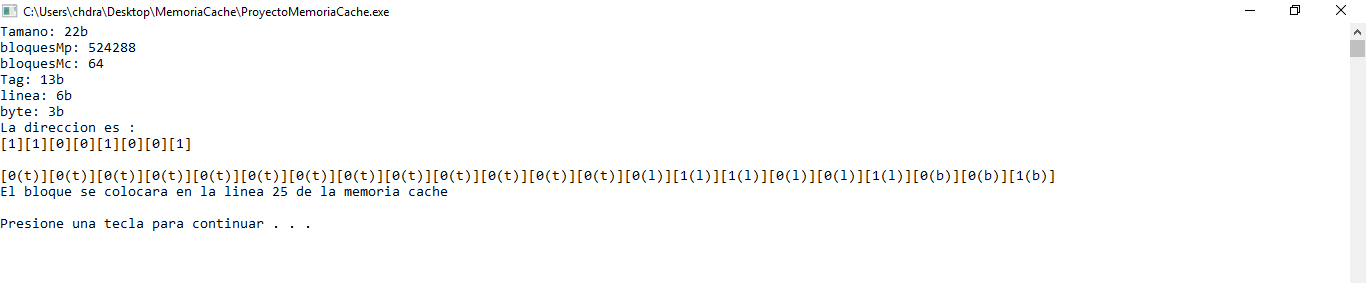
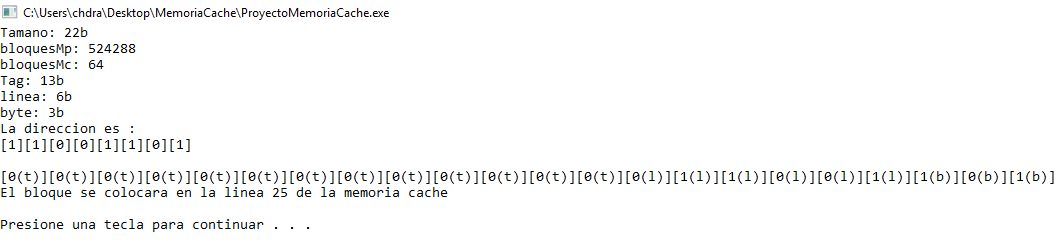
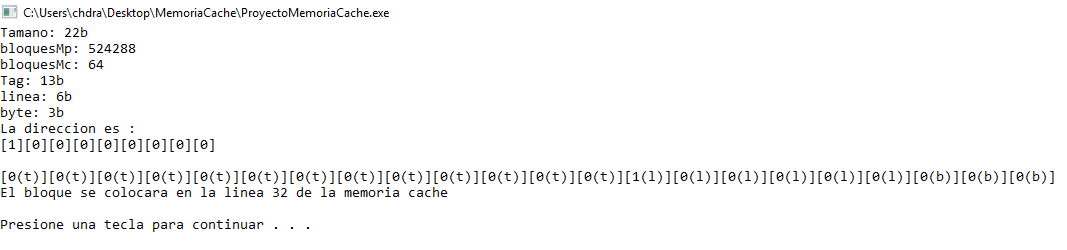
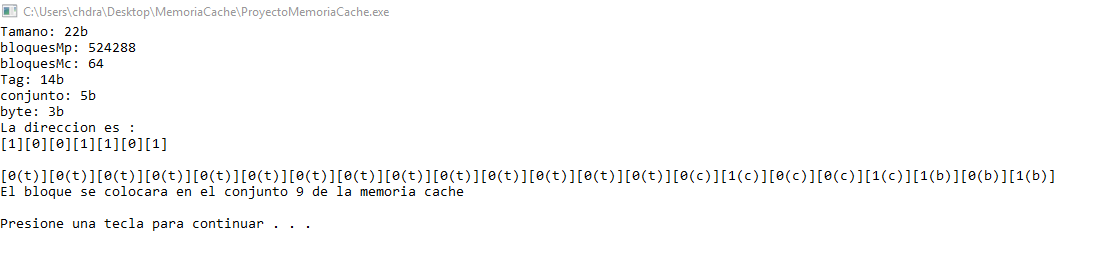
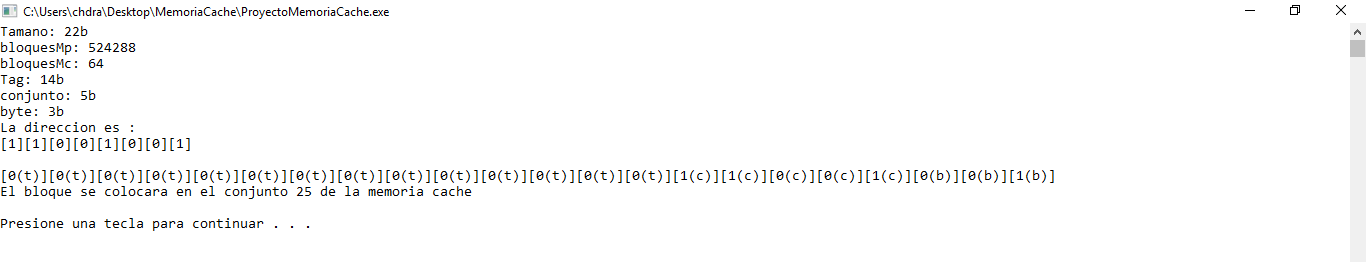
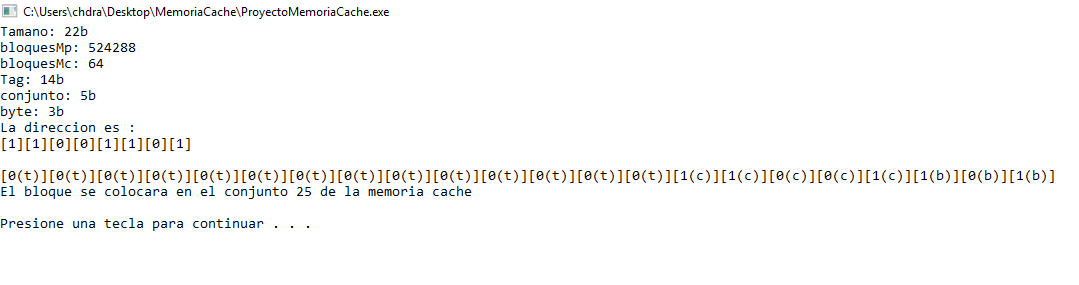
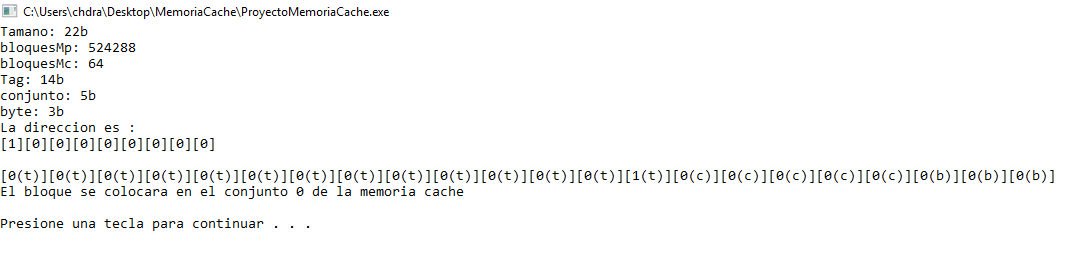
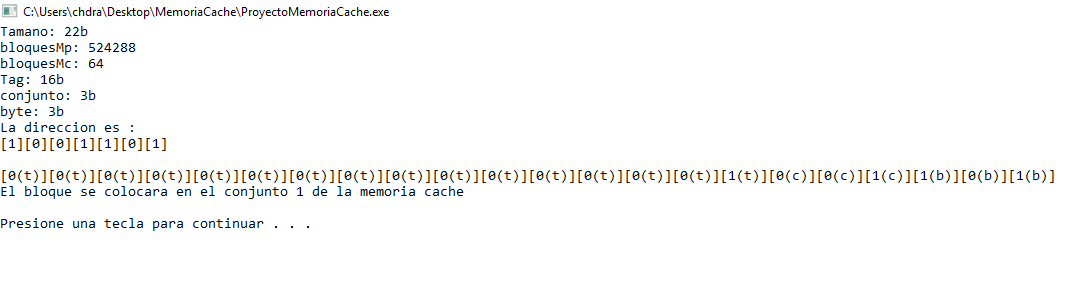
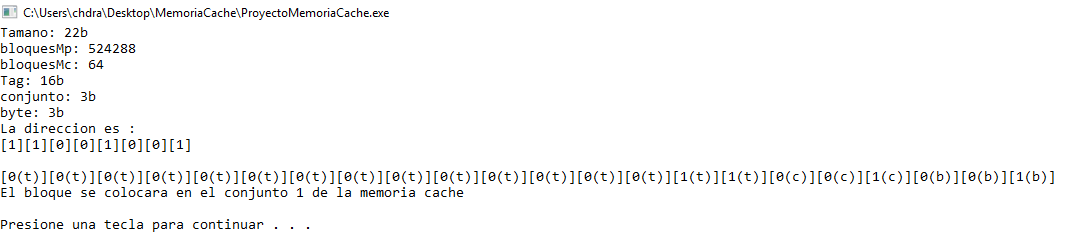
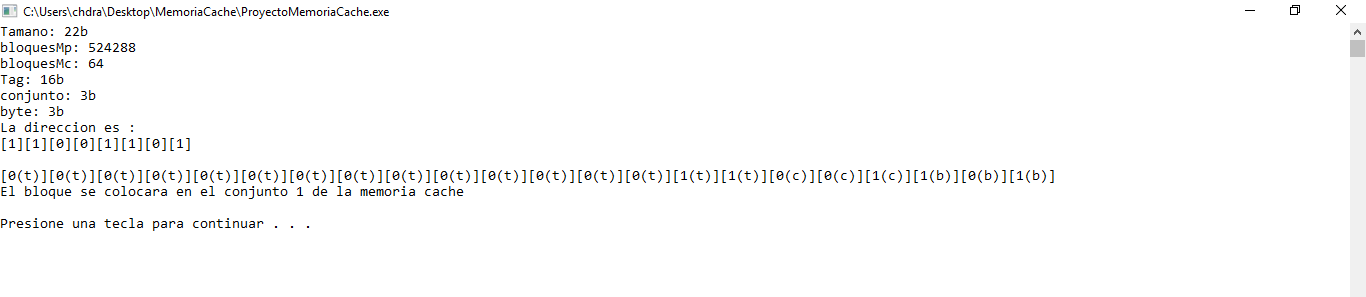
1. **Mapeo Directo**
   1. ****Memoria principal 256KB, memoria caché 64B, bloque 16B:
   2. Memoria principal 1MB, memoria caché 64B, bloque 16B****:
   3. Memoria principal 4MB, memoria caché 256B, bloque 32B
   4. Memoria principal 16MB, memoria caché 256B, bloque 2B
2. Mismos 4 casos para mapeo asociativo por conjunto con 2 bloques por conjunto.
   1. Memoria principal 256KB, memoria caché 64B, bloque 16B:
   2. Memoria principal 1MB, memoria caché 64B, bloque 16B:
   3. Memoria principal 4MB, memoria caché 256B, bloque 32B:
   4. Memoria principal 16MB, memoria caché 256B, bloque 2B:
3. Mismos 4 casos para mapeo asociativo por conjunto con 4 bloques por conjunto:
   1. Memoria principal 256KB, memoria caché 64B, bloque 16B:
   2. Memoria principal 1MB, memoria caché 64B, bloque 16B:
   3. Memoria principal 4MB, memoria caché 256B, bloque 32B:
   4. Memoria principal 16MB, memoria caché 256B, bloque 2B:

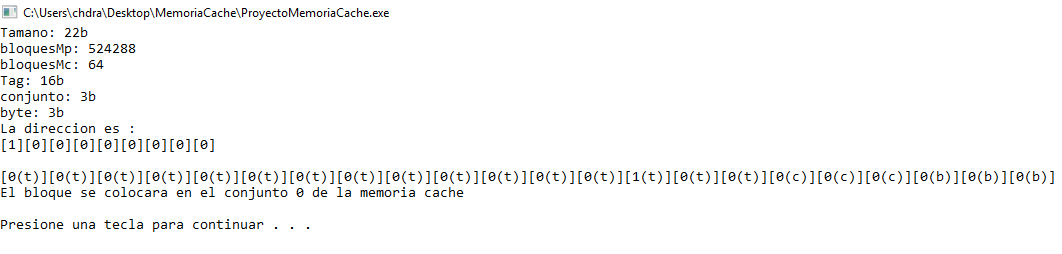


1. Mismos 4 casos para mapeo asociativo por conjunto con 8 bloques por conjunto:
   1. Memoria principal 256KB, memoria caché 64B, bloque 16B:
   2. Memoria principal 1MB, memoria caché 64B, bloque 16B:
   3. Memoria principal 4MB, memoria caché 256B, bloque 32B:
   4. Memoria principal 16MB, memoria caché 256B, bloque 2B:



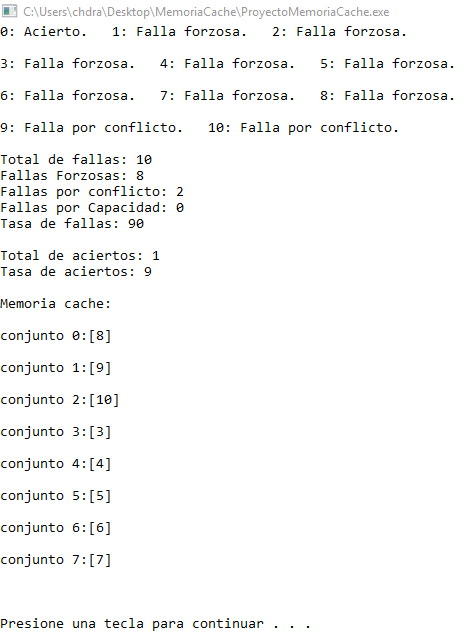
**Módulo 2:**

1. Mapeo directo, tamaño de memoria principal de 4MB, tamaño de memoria caché de 512B, tamaño de bloque de 8B:
2. **77:**
3. **201:**
4. **205:**
5. **256:**
6. Mapeo Asociativo por conjunto, tamaño de memoria principal de 4MB, tamaño de memoria caché de 512B, tamaño de bloque de 8B, tamaño del conjunto de 2 bloques:
7. **77:**
8. **201:**
9. **205:**
10. **256:**
11. Mapeo Asociativo por conjunto, tamaño de memoria principal de 4MB, tamaño de memoria caché de 512B, tamaño de bloque de 8B, tamaño del conjunto de 8 bloques:
12. **77:**
13. **201:**
14. **205:**
15. **256:**

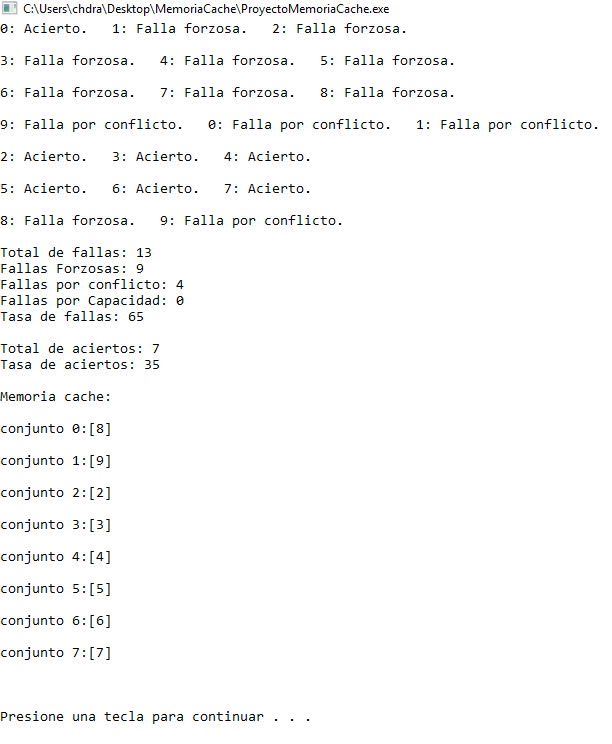
****

**Módulo 3:**

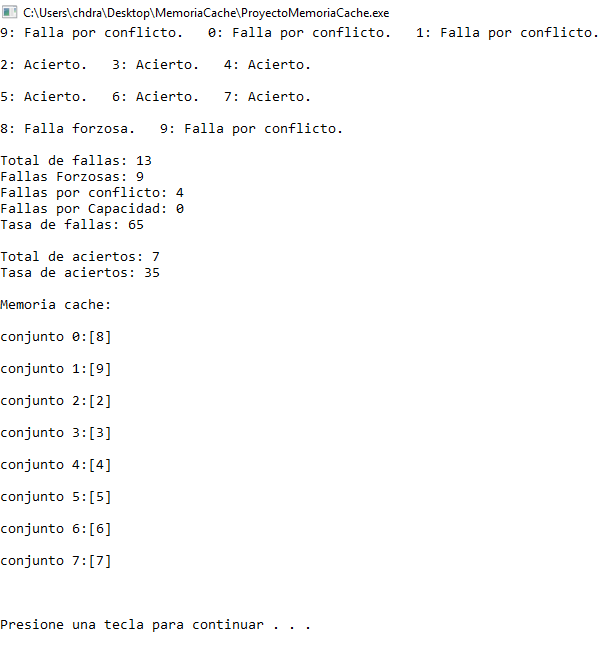
1. Numero de bloques en la memoria caché de 8 y cantidad de conjuntos 8 (Mapeo directo). Utilice algoritmo de remplazo LRU con la secuencia de direcciones del 0 al 10.



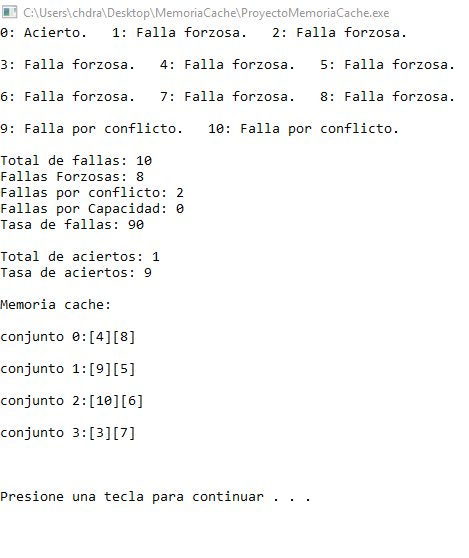
1. Numero de bloques en la memoria caché de 8 y cantidad de conjuntos 8 (Mapeo directo). Utilice algoritmo de remplazo LRU con la secuencia de direcciones 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9.

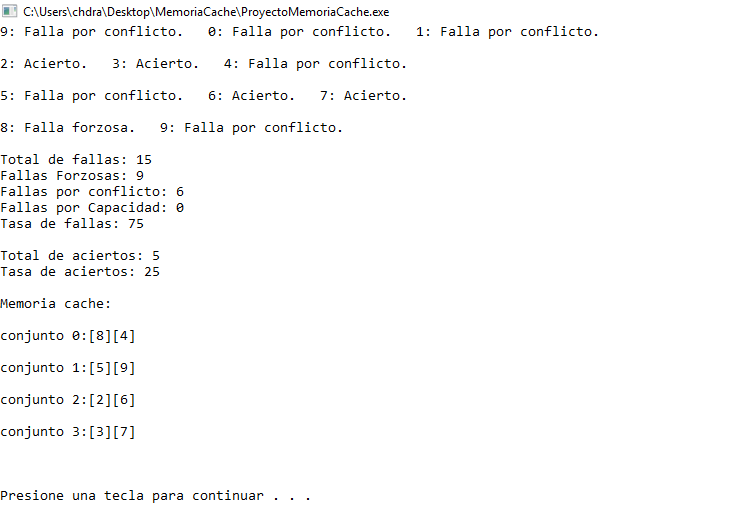


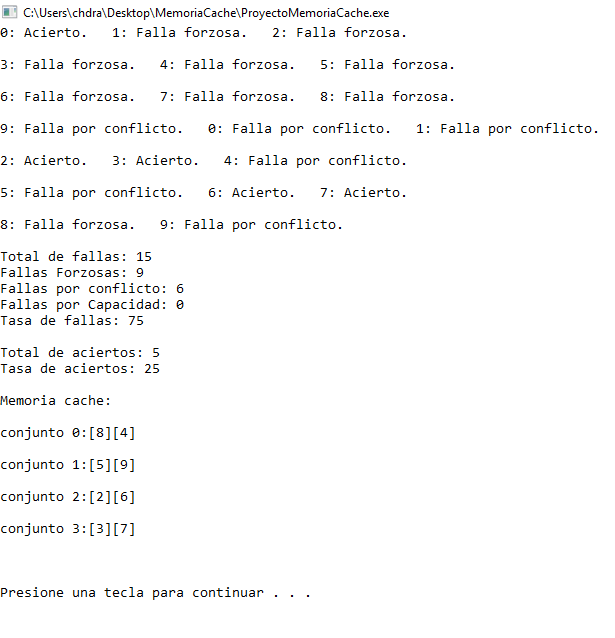
1. Numero de bloques en la memoria caché de 8 y cantidad de conjuntos 8 (Mapeo directo). Utilice algoritmo de remplazo FIFO con la secuencia de direcciones 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9.



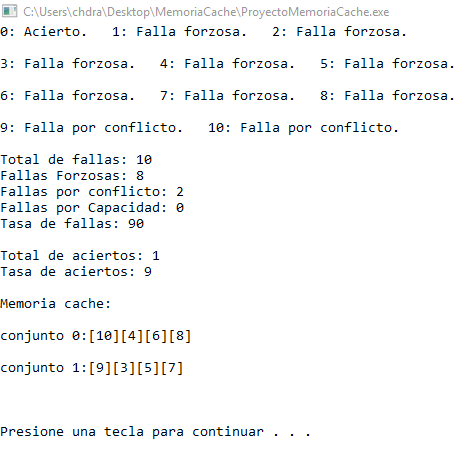
1. Numero de bloques en la memoria caché de 8 y cantidad de conjuntos 4:
2. Utilice algoritmo de reemplazo LRU con la secuencia de direcciones de 0 al 10:

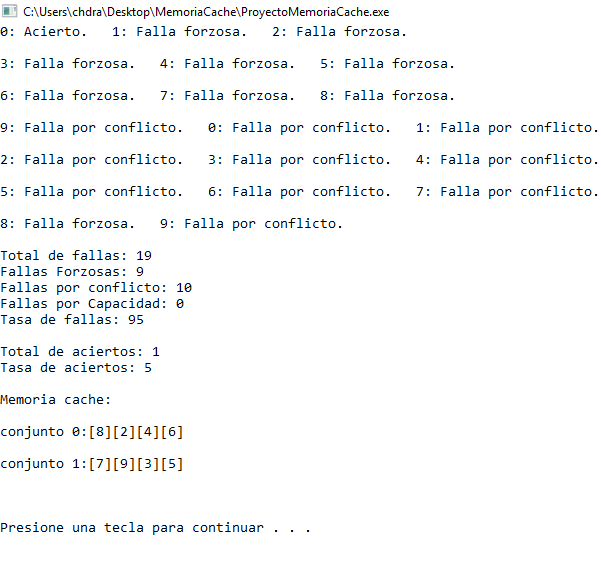
****

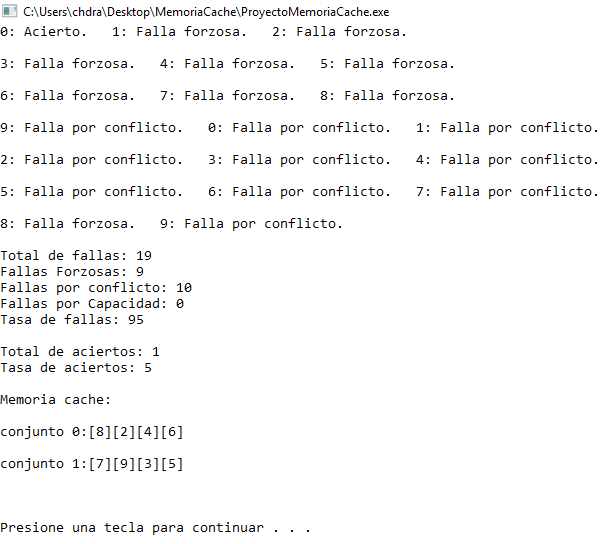
1. Utilice algoritmo de reemplazo LRU con la secuencia de direcciones 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9:****
2. Utilice algoritmo de reemplazo FIFO con la secuencia de direcciones 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

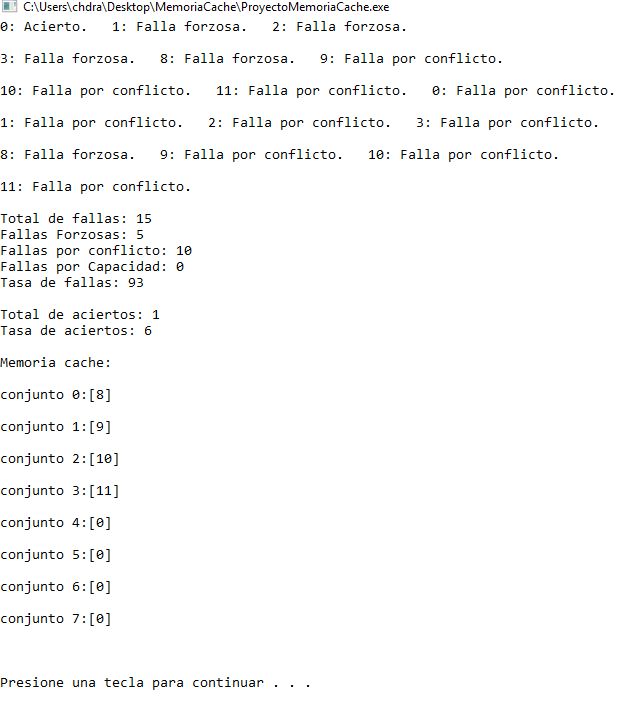
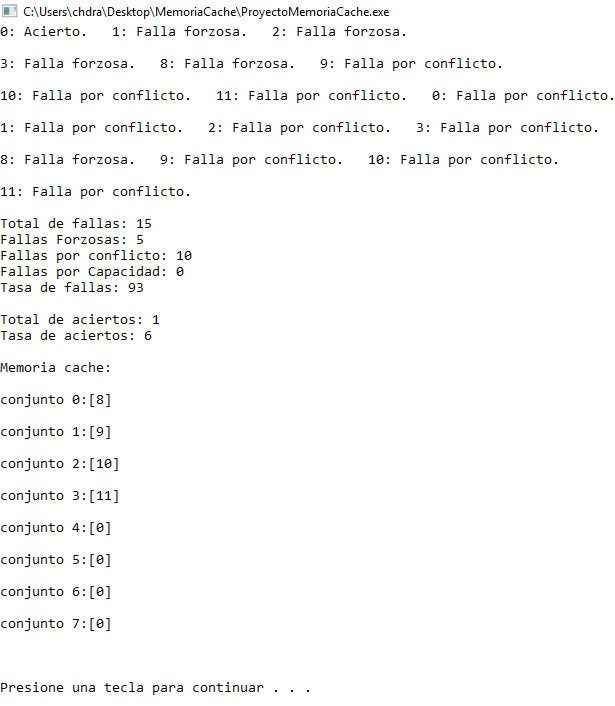
****

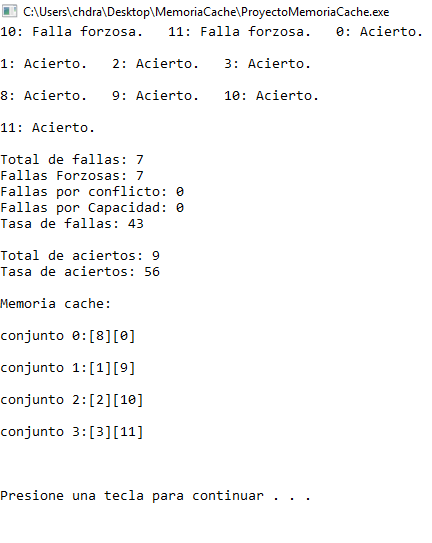
1. Numero de bloques en la memoria caché de 8 y cantidad de conjuntos 2:
   1. Utilice algoritmo de reemplazo LRU con la secuencia de direcciones de 0 al 10:

****

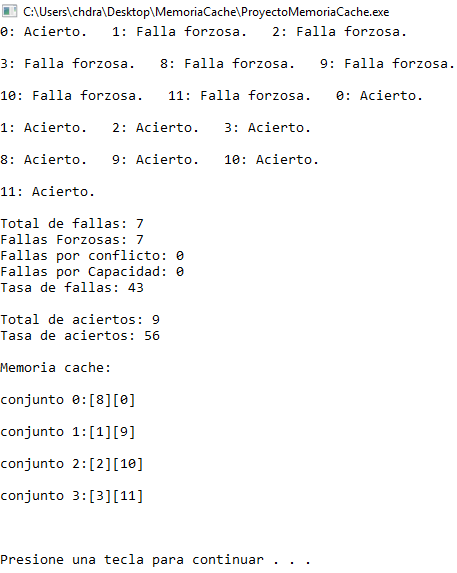
* 1. ****Utilice algoritmo de reemplazo LRU con la secuencia de direcciones 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9:
  2. Utilice algoritmo de reemplazo FIFO con la secuencia de direcciones 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9:

****

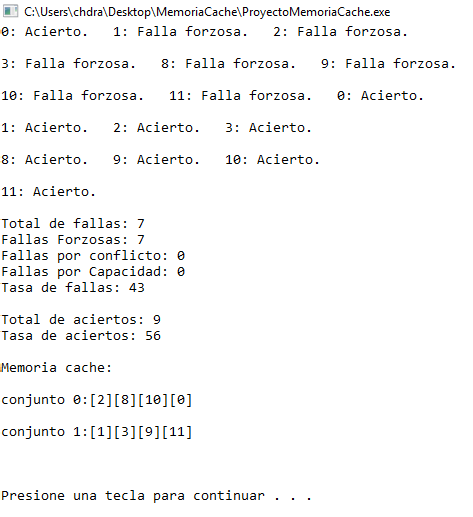
1. **Realice lo siguiente:**
   1. ****Numero de bloques en la memoria caché de 8 y cantidad de conjuntos 8 (Mapeo directo). Utilice algoritmo de remplazo LRU con la secuencia de direcciones 0 1 2 3 8 9 10 11 0 1 2 3 8 9 10 11:
   2. ****Numero de bloques en la memoria caché de 8 y cantidad de conjuntos 8 (Mapeo directo). Utilice algoritmo de remplazo FIFO con la secuencia de direcciones 0 1 2 3 8 9 10 11 0 1 2 3 8 9 10 11:
   3. Numero de bloques en la memoria caché de 8 y cantidad de conjuntos 4. Utilice algoritmo de remplazo LRU con la secuencia de direcciones 0 1 2 3 8 9 10 11 0 1 2 3 8 9 10 11:

****

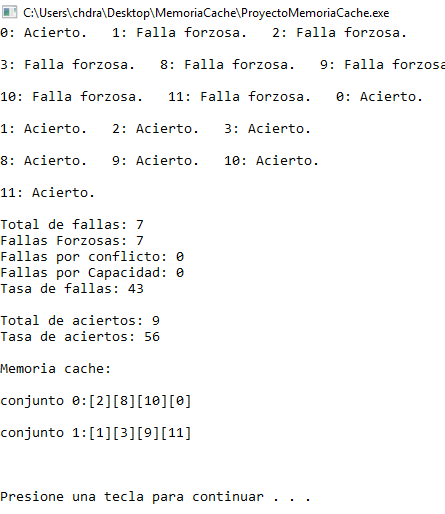
* 1. Numero de bloques en la memoria caché de 8 y cantidad de conjuntos 4. Utilice algoritmo de remplazo FIFO con la secuencia de direcciones 0 1 2 3 8 9 10 11 0 1 2 3 8 9 10 11:



* 1. Numero de bloques en la memoria caché de 8 y cantidad de conjuntos 2. Utilice algoritmo de remplazo LRU con la secuencia de direcciones 0 1 2 3 8 9 10 11 0 1 2 3 8 9 10 11:

****

* 1. Numero de bloques en la memoria caché de 8 y cantidad de conjuntos 2. Utilice algoritmo de remplazo FIFO con la secuencia de direcciones 0 1 2 3 8 9 10 11 0 1 2 3 8 9 10 11:

****

**Análisis**

**Módulo 1:**

Observando los resultados obtenidos a lo largo de los distintos ejercicios planteados para el módulo 1, se pudo ver que tanto los bloques de memoria principal y caché daban los mismos resultados haciéndolo por mapeo directo o asociativo por conjunto. También se pudo ver a simple vista que en el mapeo asociativo por conjunto a medida que incrementábamos el número de bloques por conjunto, los tag se incrementaban en uno y los conjuntos disminuían en uno.

Como se puede comprobar las imágenes muestran la forma en la que quedará dividida la dirección para que pueda ser ubicado en la memoria caché o en la principal, en el caso de mapeo directo se divide en tag, línea y byte, siendo una línea por cada bloque de memoria principal que se llevara a caché, mientras que en el asociativo por conjuntos, se divide en tag, conjunto y byte, siendo el conjunto la representación del conjunto en el cual se ubicará el elemento en la memoria caché, siendo un bloque de memoria dividido en varios conjuntos en la caché.

**Módulo 2:**

Como se puede apreciar dadas las direcciones, en el mapeo directo se puede visualizar en que línea de la memoria caché se ubicará el dato o elemento correspondiente a la dirección, mientras que en el mapeo asociativo por conjunto se puede visualizar el número de conjunto donde se ubicará el elemento, correspondiendo a lo indicado por la sección del conjunto.

**Módulo 3:**

En el caso de los puntos 1 y 2, se puede apreciar que varían los fallos y los aciertos, y además las matrices resultantes varían por solo uno de los elementos, siendo éste el perteneciente al conjunto 2, para la primera secuencia en el conjunto 3 se inserta el dato correspondiente a la dirección 10 y en la segunda secuencia en el mismo conjunto se inserta el dato correspondiente a la dirección 2.

Para la repetición del punto 2 pero utilizando el método de reemplazo FIFO, se obtienen los mismos resultados, dado que en este caso en el LRU no hubo ningún caso en el cual el menos recientemente usado no haya sido el último elemento en entrar en ese conjunto.

Para el punto 4, cada uno de los resultados, en cada caso se obtuvo cada conjunto con 2 bloques cada uno, dado a que se reparten los 8 bloques de memoria caché en 4 conjuntos. En el conjunto 0 y 1 se puede apreciar que los elementos intercambian de lugar, mientras que en el conjunto 2, varía uno de los elementos.

En el punto 5, cada resultado para cada caso tiene 4 bloques por conjunto ya que los 8 bloques de memoria caché se reparten en un total de 2 conjuntos. Y en el FIFO y el LRU se obtiene el mismo resultado.

En el punto 6, se repite cada caso pero con una nueva secuencia, se puede observar que en todos los casos el FIFO y el LRU dan los mismos resultados en cada uno de los casos, esto debido a lo explicado en el punto 2.

**Conclusiones**

En conclusión la memoria caché es mucho más compleja de lo que se suele creer, ya que como pudimos observar, se tienen en cuenta gran cantidad de aspectos para que las direcciones sean ubicadas tanto en la memoria principal como en la caché, sin embargo esto es lo que permite acceder de forma rápida y eficiente a los datos o la información que se encuentre en dicha memoria principal haciendo uso de la jerarquía de memoria y de los principios de localidad para hacer que dichos elementos sean mucho más accesibles y sean localizados en menor tiempo por medio de la memoria caché.

Hay que destacar que para sacar el máximo provecho de esta arquitectura, se deben usar métodos de búsqueda y reemplazo de mayor eficiencia dependiendo del caso, en el caso de LRU que es el método de reemplazo más eficiente, sin embargo dado su complejidad es costoso y poco práctico para implementar, por lo que hay que tener en cuenta distintos factores para utilizar los distintos métodos en los casos que sean pertinentes.