UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Laboratorio Nro. 3: Caché

Autor:	Andrés Muñoz Bravo
Curso:	Organización de computadores
Sección	B-2
Profesores:	Daniel Wladdimiro Cottet
	Alfonso Guzmán
Ayudantes:	Pablo Ulloa
	Ariel Undurraga

7 de Junio de 2017

Tabla de contenidos

1. Introducción **1**

1.1. Contexto 1

1.2. Problema 1

1.3. Motivación 1

1.4. Objetivos 2

1.5. Herramientas 2

1.6. Estructura del informe 2

2. Marco teórico **3**

3. Desarrollo **5**

3.1. Elaboración del programa 5

3.1.1. Estructuras 5

3.1.2. Explicación de algoritmos empleados 6

3.2. Experimentos a realizar 9

3.3. Resultados obtenidos 9

3.4. Análisis de resultados 9

4. Conclusiones **10**

5. Referencias **10**

1. Introducción

1.1. Contexto

Como lo demuestra a teoría vista en cátedra, el rendimiento de los computadores se ve afectado por diferentes cosas, entre ellas, los algoritmos empleados, el compilador utilizado, el lenguaje de programación, en tiempo que demora en obtener un valor desde memoria, entre otras.

En entre trabajo, se abordará el concepto de memoria caché.Esta memoria puede tener diferentes configuraciones que se clasifican según: mapeo directo, n-vías y full asociativo. Además, existen políticas de reemplazo cuando la memoria caché está llena y se necesita guardar un nuevo valor, estas son: FIFO, MRU, LRU.

1.2. Problema

Se ha pedido realizar un programa que simule el funcionamiento de una memoria caché. Este programa, debe recibir la configuración de la memoria, es decir, se debe entregar la cantidad de vías, palabras por bloque, bloques totales, la política de reemplazo y el archivo que contiene la información que se irá guardando. Los resultados de ejecutar el programa deben ser dos archivos; El primero, debe contener las estadísticas, es decir, la tasa de hit y miss. El segundo, debe contener los datos alojados en la memoria posterior a la última consulta realizada.

1.3. Motivación

La principal motivación es aprender de manera práctica los conceptos vistos en cátedra sobre memoria caché. Además, realizar un análisis sobre que configuración tiene mejor tasa de hit.Finalmente, con el programa realizado se podrán comprobar ejercicios propuestos sobre memoria de caché, lo cual puede ayudar a las futuras generaciones para estudiar y entender de mejor manera estos conceptos sobre organización de computadores.

1.4. Objetivos

- Objetivo general: Realizar un programa que simule una memoria caché, que entregue la tasa de hit y miss, así como también los datos, posterior a la última consulta de la ejecución.
- Objetivos específicos:
  - Implementar una estructura caché que soporte las diferentes configuraciones que se pueden seleccionar.
  - Implementar la política de reemplazo FIFO.
  - Implementar la política de reemplazo MRU.
  - Implementar la política de reemplazo LRU.

1.5. Herramientas

- Sistema operativo: Windows 10 Education de 64 bits.
- Editor de texto: Sublime Text 3, build: 3126.
- Lenguaje de programación "C ".
- Compilador GCC, versión 5.3.0.

1.6. Estructura del informe

En la primera parte del informe se encuentra el marco teórico donde se explicarán conceptos claves para entender lo que se explica en el informe. Luego en el capítulo de desarrollo se describe como es elaborado el programa pedido, así como también se presentan experimentos realizados y un posterior análisis de estos resultados obtenidos. Finalmente, en el último capítulo se encuentran las conclusiones a partir de todo el proceso realizado en este trabajo.

2. Marco teórico

- **Memoria caché:** Es una área de almacenamiento relativamente pequeña compara con la memoria principal. Esta memoria es utilizada para datos que son constantemente consultados, ya que su tiempo de acceso es menor respecto a las otras memorias del computador. Cabe recalcar que es muy volátil.

  - **Número de vías:** Es el número de bloques que tiene un conjunto.
  - **Número de bloques:** Es el número total de bloques en la memoria caché.
  - **Número de conjuntos:** Es el número de conjuntos que tiene la memoria. Los conjuntos contienen bloques, es decir, en un conjunto existen bloques que depende del número de bloques y número de vías. Se puede calcular con la formula 1:

$$NroConjuntos = NroBloques/NNroVias \tag{1}$$

  - **Número de palabras por bloque:** Es la cantidad de palabras que existe en un bloque, cada palabra tiene un tamaño dependiendo de la arquitectura del computador, es decir, si es de 32 bits, el tamaño de la palabra es de 32 bits.

G0				G1			
B0		B1		B0		B1	
W0	W1	W0	W1	W0	W1	W0	W1
88	89	60	61	34	35	46	47

Figura 1: Ejemplo de cache.

En la figura 1, se muestra un ejemplo de una memoria cache de 2 vías, 2 palabras por bloque y 8 bloques, donde las çajas"de color negro, azul, rojo y amarillo representan los conjuntos (G0,G1),los bloques (B0, B1, B2, B3), las palabras (W0, W1.) y los datos respectivamente.

- **Política de reemplazo:** Existen diferentes políticas de reemplazo, entre ellas: FIFO, MRU, LRU. Estas se utilizan cuando se necesita guardar un dato en la memoria caché, pero ésta, está en su capacidad máxima.
  - **FIFO:** El primer dato que entra es el primero en salir.
  - **MRU:** El dato que se utiliza por última vez es el que se reemplaza.
  - **LRU:** El dato que más tiempo lleva en la memoria cache es el que se reemplaza.
- **Principio de localidad espacial:** Si un ítem es referenciado, los ítems cercanos tienden a ser referenciados pronto.
- **Principio de localidad temporal:** Si un ítem es referenciado, tiende a ser referenciado pronto.
- **Hit:** Se dice que ocurre un 'Hit', cuando se encuentra un dato en la memoria de caché al momento de ser consultado.
- **Miss:** Se dice que ocurre un 'Miss', cuando **NO** se encuentra un dato en la memoria de caché al momento de ser consultado.
- **Tasa de Hit:** Es un porcentaje de la cantidad total de hit ocurridos. Se calcula con la formula 2:

$$Tasadehit = (Cantidaddehit/(Cantidaddehit + Cantidaddemiss)) * 100 \quad (2)$$

- **Tasa de Miss:**

  Es un porcentaje de la cantidad total de hit ocurridos. Se calcula con la formula 3:

$$Tasadehit = (Cantidaddemiss/(Cantidaddehit + Cantidaddemiss)) * 100 \quad (3)$$

3. Desarrollo

3.1. Elaboración del programa

3.1.1. Estructuras

Para el desarrollo del programa se utilizan tres estructuras: Cache, Group y Block. En base a estas estructuras se lleva a cabo la implementación del programa para entregar la solución pedida implementada la solución.

- **Block**: Representa un bloque de la memoria cache.
  - **words**: Es un arreglo de números que representa las palabras de un bloque. Es aquí donde se guardarán los datos de la caché.
  - **lru**: Es un número, que representa hace cuánto tiempo es utilizado el bloque, es decir, mientras más alto es el número quiere decir que se utilizó hace más tiempo.
- **Group**: Representa los grupos o conjuntos dentro de una memoria caché.
  - **Blocks**: Es un arreglo de la estructura Block, es decir, representa los bloques que están contenidos en dicho grupo o conjunto.
  - **counter**: Es un número que representa cuando datos se han insertado en un grupo.
  - **mru**: Es un número que indica el bloque en el cual se debe insertar un dato cuando se utiliza la política MRU, es decir, debe contener el índice del último bloque con el cual se ha interactuado.
- **Cache**: Representa la memoria cache.
  - **Grupos**: Arreglo de estructura Group, es decir representa los grupos que están contenidos en la memoria caché.
  - **wordsPerBlock**: Es un número que representa la cantidad de palabras por bloque.
  - **blocksPerGroup**: Es un número que representa la cantidad de bloques por grupo o conjunto.

- **numberGroups**: Es un número que representa la cantidad de grupos que existen en la memoria caché.
- **numberBlocks**: Es un número que representa la cantidad de bloques totales existentes en la memoria caché.
- **hit**: Es un número que representa la cantidad de hit ocurridos.
- **miss**: Es un número que representa la cantidad de miss ocurridos.
- **politics**: Es un string o cadena de caracteres, en el cual se guardará la política de reemplazo a utilizar.

3.1.2. Explicación de algoritmos empleados

Dentro del programa existen 3 principales algoritmos, donde cada uno de estos representa una política de reemplazo. En cada uno de estos algoritmos se recibe la estructura de dato Cache, y un número que es el dato el cual se consulta.

- FIFO:

```
1  void FIFO(Cache* cache, int dato)
2  {
3      int group = dato / cache->wordsPerBlock % cache->numberGroups;
4      int block = cache->groups[group].counter % cache->blocksPerGroup;
5      if (isInCache(cache,dato)){
6          cache->hit++;
7      }
8      else{
9          cache->miss++;
10         inputWords(cache,group,block, dato);
11         cache->groups[group].counter++;
12     }
13 }
```

Figura 2: Algoritmo de política de reemplazo FIFO.

En la figura 2, se puede apreciar el algoritmo implementado para realizar la inserción de datos cuando se utiliza la política FIFO. En las líneas 3 y 4 se obtiene los valores necesarios para saber en cual bloque se debe insertar el dato. Luego si el dato está en la cache, se suma un hit. En caso contrario se debe insertar en el bloque y grupo obtenido anteriormente, y sumar al contador.

- MRU:

```
1  void MRU(Cache* cache, int dato)
2  {
3      int group;
4      group = dato / cache->wordsPerBlock % cache->numberGroups;
5      if (!isComplete(cache,group))
6      {
7          FIFO(cache,dato);
8      }
9      else
10     {
11         if(cache->groups[group].counter == cache->blocksPerGroup )
12         {
13             cache->groups[group].mru = cache->groups[group].counter - 1;
14             cache->groups[group].counter++;
15         }
16
17         if (isInCache(cache,dato)){
18             cache->hit++;
19             cache->groups[group].mru = indexBlock(cache,group,dato);
20         }
21         else{
22             cache->miss++;
23             inputWords(cache,group,cache->groups[group].mru, dato);
24         }
25     }
26 }
27
```

Figura 3: Algoritmo de política de reemplazo MRU.

En la figura 3, se puede apreciar el algoritmo implementado para realizar la inserción cuando se utiliza la política de reemplazo MRU. En este algoritmo es necesario identificar si el grupo del cual se consulta un dato está a su máxima capacidad. En caso de que sea verdadero, se realiza el mismo procedimiento que se realiza para la política FIFO. En caso contrario, se verifica si la consulta realizada es la siguiente después de completar el grupo, en caso de serlo, se actualiza la variable mru del grupo la cual indica cual fue el último bloque con el cual se interactuó. Luego se debe consultar si el dato está en la memoria caché. En caso de que sea verdadero, se suma un hit y luego se actualiza la variable mru, indicando el bloque del cual se produjo el hit. En caso contrario, se suma un hit y se inserta el dato en el bloque que indique la variable mru del grupo.

- LRU:

```
1  void LRU(Cache* cache, int dato)
2  {
3      int group;
4      int block;
5      group = dato / cache->wordsPerBlock % cache->numberGroups;
6
7      if (!isComplete(cache,group))
8      {
9          if (isInCache(cache,dato)){
10             cache->hit++;
11             block = indexBlock(cache,group,dato);
12             interactionBlockLRU(cache,group,block);
13         }
14         else{
15             block = cache->groups[group].counter % cache->blocksPerGroup;
16             cache->miss++;
17             inputWords(cache,group,block, dato);
18             interactionBlockLRU(cache,group,block);
19             cache->groups[group].counter++;
20         }
21     }
22     else
23     {
24         if (isInCache(cache,dato)){
25             cache->hit++;
26             block = indexBlock(cache,group,dato);
27             interactionBlockLRU(cache,group,block);
28         }
29         else{
30             cache->miss++;
31             block = indexBlockLRU(cache,group);
32             inputWords(cache,group,block, dato);
33             interactionBlockLRU(cache,group,block);
34         }
35     }
36 }
```

Figura 4: Algoritmo de política de reemplazo LRU.

inserción cuando se utiliza la política de reemplazo LRU. En este algoritmo se realizan prácticamente los mismos pasos que en la política anterior, la diferencia radica en que el dato que se reemplaza es el que se utilizó hace mayor tiempo. Por lo tanto, cada bloque tiene una variable de tipo número que indicara su "vejes". Con la función intactionLRU, se incrementa a todas las variables lru de los bloques en 1 y se deja en 0 la variable del bloque con el cual se interactuó, esto garantiza que el más "viejo"tenga un valor más alto respecto a los otros. Con la función indexBlockLRU se obtiene el índice del bloque con el cual se interactuó hace mayor tiempo. Así, se conoce cual es el bloque en el cual se debe insertar el dato.

3.2. Experimentos a realizar

Se realizarán pruebas con diferentes configuraciones de caché y se entregará un archivo que contiene 1000 datos. Las configuraciones son las siguientes:

1. FIFO, 4 vias, 2 palabras por bloque, 8 bloques.
2. MRU, 4 vias, 2 palabras por bloque, 8 bloques.
3. LRU, 4 vias, 2 palabras por bloque, 8 bloques.
4. FIFO, 8 vias, 2 palabras por bloque, 16 bloques.
5. MRU, 8 vias, 2 palabras por bloque, 16 bloques.
6. LRU, 8 vias, 2 palabras por bloque, 16 bloques.

3.3. Resultados obtenidos

Los resultados obtenidos para las pruebas anteriores son:

1. tasa de hit: 15.400 % // tasa de miss: 84.600 %
2. tasa de hit: 17.100 % // tasa de miss: 82.900 %
3. tasa de hit: 15.900 % // tasa de miss: 84.100 %
4. tasa de hit: 31.800 % // tasa de miss: 68.200 %
5. tasa de hit: 31.100 % // tasa de miss: 68.900 %
6. tasa de hit: 33.200 % // tasa de miss: 66.800 %

3.4. Análisis de resultados

Al comparar la tasa de hit, se puede observar que aumenta a medida que aumenta la cantidad de bloques, esto es debido a que existe mayor espacio para guardar informacion en la memoria caché, cabe recalcar que las diferentes politicas entregan una tasa de hit muy cercanas entre ellas. Cabe recalcar que no se puede asegurar que una configuracion en mejor que otra, ya que la cantidad de hit depende del orden que se consulten los datos.

4. Conclusiones

En el presente trabajo se logró completar exitosamente la elaboración del programa que consiste en simular el funcionamiento de una memoria caché en sus diferentes configuraciones, logrando generar de manera correcta los archivos de salida solicitados, los cuales entregan la información de las estadísticas y los datos que se encuentran en la caché al terminar la ejecución del programa. Cabe recalcar que estos archivos son entregados en un archivo HTML que ayuda a la comprensión de los resultados obtenidos.

En este laboratorio se aprendió de manera práctica y a profundidad, los diferentes conceptos vistos en cátedra sobre caché. Haciendo mención al análisis realizado en este informe, se puede rescatar que la tasa de hit aumenta cuando se incrementa la cantidad de bloques en una caché ya que se demostró con los resultados obtenidos. Es necesario mencionar también que para realizar análisis más profundos sobre la memoria caché en sus diferentes configuraciones es necesario realizar muchos más experimentos, los cuales no se podrían explican con un mínimo de 10 páginas en el informe, ya que existiría una enorme cantidad de información la cual se debe analizar.

Finalizando el trabajo realizado, se espera que se saque provecho este y los anteriores laboratorios realizados por los estudiantes durante este semestre, de manera que sirvan de aprendizaje para futuras generaciones, ya que es mucho más didáctico aprender y verificar ejercicios con un programa que entregue el resultado correcto. Más aún, mencionar que laboratorios realizados por otros compañeros realizan una traza detallada del paso a paso que ocurre en el programa.

5. Referencias

- CACHE-CACHE COMPARISON FOR SUPPORTING MEANINGFUL LEARNING, Jingyun Wang and Seiji Fujino, 2015-Oct.
- REDUCING ROUTER FORWARDING TABLE SIZE USING AGGREGATION AND CACHING, Yaoqing Liu, 2013-August.