TP2 - Memorias caché

Lucio Lopez Lecube, *Padrón Nro. 96583*
luciolopezlecube@gmail.com
Santiago Alvarez Juliá, *Padrón Nro. 99522*
santiago.alvarezjulia@gmail.com
Bautista Canavese, *Padrón Nro. 99714*
bauti-canavese@hotmail.com

Grupo Nro. - 1er. Cuatrimestre de 2018
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

29 de mayo de 2018

Índice

1. Introducción

Simulación de una memoria caché asociativa por conjuntos de dos vías, de 1KB de capacidad, bloques de 32 bytes, política de reemplazo LRU y política de escritura WT/?WA. Se asume que el espacio de direcciones es de 12 bits, y hay entonces una memoria principal a simular con un tamaño de 4KB.

2. Problemática a desarrollar

El programa a escribir, en lenguaje C, recibirá un archivo de texto plano, en donde se encuentran las instrucciones a realizar de la siguiente manera:

```
W dddd, vvv
R dddd
MR
```

- Los comandos de la forma `R dddd` lee el dato dddd y lo imprime.
- Los comandos de la forma `W dddd, vvv` escriben en la dirección dddd el valor vvv.
- Los comandos de la forma `MR` imprime el miss rate actual del programa ejecutado sobre la cache.

Además, implementando las siguientes primitivas:

```
void init()
unsigned char read_byte(int address)
int write_byte(int address, unsigned char value)
unsigned int get_miss_rate()
```

3. Detalles de implementación

3.1. Estructura Cache

3.1.1. Division de address

- Offset de palabra de 3 bits, ya en cada bloque se encuentran 8 lineas.
- Offset de bloque de 2 bits, al ser de 4 bytes la palabra.
- Index formado por otros 4 bits para seleccionar entre los 16 bloques.
- Tag formado por 3 bits, contiene los bits resantes de la dirección.

Por lo tanto, la dirección es interpretada con un offset formado por los primeros 5 bits menos significativos y los 7 bits subsiguientes conforman el index y el tag.

3.1.2. LRU

Para la implementación de la politica de reemplazo LRU dentro de la cache, se opto por colocar una varaible dentro de cada bloque, que se inicializa en cero; en caso de ser el ultimo recientemente es seteado como uno. Por lo tanto al cargar un bloque solo se verifica si ese flag esta en 0 o en 1.

3.2. Salida estándar

Se muestra por salida estándar según el comando solicitado por el usuario. En la siguiente sección *Ejemplos* se podrán apreciar los flags implementados del programa y como ejecutarlos.

4. Ejemplos

Con la opción -h se vera los distintos falgs disponibles para ejecutar el programa.

```
$
Usage:
  cache -h
  cache -V
  cache [options] filename
Options:
  -h, --help    Prints usage information.
  -V, --version Prints version information.
  -i, --input   Path to input file.

Example:
./cache  input_file
./cache -i input_file
```

5. Casos de Prueba

1. En el caso del primer archivo:

   `$cat prueba1.mem`

   ```
   1  W 0, 16
   2  R 0
   3  R 1024
   4  R 8
   5  R 2050
   6  R 3074
   7  W 8, 12
   8  R 8
   9  R 8
   10 W 3072, 255
   11 W 2048, 10
   12 R 0
   13 MR
   ```

   linea 1: Cache vacia, es un *miss*, se escribe 16 en la posición 0, por la politica $WT/notWA$ solo se escribe en la memoria principal.

   linea 2: Se quiere leer la posicion cero, es un *miss*, se carga en cache al conjunto 0.

   linea 3: Se quiere leer la posición 1024, es un *miss* compulsivo, se carga en cache al conjunto 0.

   linea 4: Se quiere leer la posicion 8, como anteriormente se cargo el bloque de 0 a 31, es un *hit*.

   linea 5/6: *misses* compulsivos, se cargan en el conjunto 0 (por ser de index 0000).

   linea 7/8: *miss* porque el bloque fue anteriormente reemplazado por la politica LRU.

   linea 9: *hit* el bloque fue cargado en la linea anterior, devuelve 12.

   linea 10: *hit*, bloque fue cargado anteriormente. *linea 11*: *miss*. *linea 12*: *hit*.

   $$\boxed{MR = \frac{8}{12} \times 100 = 66}$$

   `$./cache prueba1.mem` Cuya salida por salida estándar mediante la pruebas es:

   ```
   Instruccion: W
   Resultado: -1

   Instruccion: R
   ```

```
Resultado: -1

Instruccion: R
Resultado: -1

Instruccion: R
Resultado: 0

Instruccion: R
Resultado: -1

Instruccion: R
Resultado: -1

Instruccion: W
Resultado: -1

Instruccion: R
Resultado: -1

Instruccion: R
Resultado: 12

Instruccion: W
Resultado: 0

Instruccion: W
Resultado: -1

Instruccion: R
Resultado: 16

Instruccion: MR

Resultado: 66
```

2. En el caso del segundo archivo:

```
$cat prueba2.mem

1  R 0
2  R 31
3  W 32, 10
4  R 32
5  W 32, 20
6  R 32
7  R 1040
8  R 2064
9  R 32
10 R 32
11 MR
```

linea 1: Cache vacia, es un *miss* compulsivo, se carga el bloque (0,31) sobre el conjunto 0 de cache.

linea 2: Es un *hit*, fue cargado en la linea anterior.

linea 3: Es un *miss*, se esccribe solo sobre memoria principal.

linea 4: Es un *miss*, se carga el bloque (32, 63) al conjunto 1 del cache por tener un index de 0001.

linea 5: *hit*.

linea 6: *hit* devuelve 20.

linea 7/8: *misses* compulsivos, el bloque que contiene 1024 va a parar al conjunto 0 del cache y el 2064 al 1.

linea 9/10: *hit*, bloque fue cargado anteriormente en la via 0 del conjunto 1.

$$\boxed{MR = \frac{5}{10} \times 100 = 50}$$

`./cache prueba2.mem`

Cuya salida por salida estándar mediante la pruebas es:

```
Instruccion: R
Resultado: -1

Instruccion: R
Resultado: 0

Instruccion: W
Resultado: -1

Instruccion: R
Resultado: -1

Instruccion: W
Resultado: 0

Instruccion: R
Resultado: 20

Instruccion: R
Resultado: -1

Instruccion: R
Resultado: -1

Instruccion: R
Resultado: 20

Instruccion: R
Resultado: 20

Instruccion: MR

Resultado: 50
```

3. En el caso del tercer archivo:

```
$cat prueba3.mem

1  W 0, 1
2  W 1, 2
3  W 2, 3
4  W 3, 4
5  W 4, 5
6  R 0
7  R 1
8  R 2
9  R 3
10 R 4
11 MR
```

linea 1-5: Cache vacia, son *misses*, se escribe sobre memoria principal.

linea 6: Es un *miss*, se carga el bloque (0,31) sobre el conjunto 0 de la cache.

linea 7-10: Son *hits*, bloque fue cargado a la cache en la linea anterior.

$$\boxed{MR = \frac{6}{10} \times 100 = 60}$$

`./cache prueba3.mem`

Cuya salida por salida estándar mediante la pruebas es:

```
Instruccion: W
Resultado: -1

Instruccion: W
Resultado: -1

Instruccion: W
Resultado: -1

Instruccion: W
Resultado: -1

Instruccion: W
Resultado: -1

Instruccion: R
Resultado: -1

Instruccion: R
Resultado: 2

Instruccion: R
Resultado: 3

Instruccion: R
Resultado: 4

Instruccion: R
Resultado: 5

Instruccion: MR

Resultado: 60
```

4. En el caso del cuarto archivo:

```
$cat prueba4.mem

1  W 0, 1
2  W 1, 2
3  W 2, 3
4  W 3, 4
5  W 4, 5
6  R 0
7  R 1
8  R 2
9  R 3
10 R 4
11 R 1024
12 R 2048
```

```
13 R 0
14 R 1
15 R 2
16 R 3
17 R 4
18 MR
```

linea 1-5: Cache vacia, son *misses*, se escribe sobre memoria principal.

linea 6: Es un *miss*, se carga el bloque (0,31) sobre el conjunto 0 de la cache.

linea 7-10: Son *hits*, bloque fue cargado a la cache en la linea anterior.

linea 11/12: *Misses* compulsivos, el bloque que contiene 1024 va a parar al conjunto 0 del cache sobre la vía 1 y el 2048 en el conjunto 0, pero sobre la via 0 (aplicando política de reemplazo LRU).

linea 13: *miss*, bloque (0,31) fue reemplazado anteriormente, se vuelve a cargar en la vía 1 del conjunto 0.

linea 14-17: Son *hits*, bloque fue cargado a la cache en la linea anterior.

$$\boxed{MR = \frac{9}{17} \times 100 = 52}$$

`./cache prueba4.mem`

Cuya salida por salida estándar mediante la pruebas es:

```
Instruccion: W
Resultado: -1

Instruccion: W
Resultado: -1

Instruccion: W
Resultado: -1

Instruccion: W
Resultado: -1

Instruccion: W
Resultado: -1

Instruccion: R
Resultado: -1

Instruccion: R
Resultado: 2

Instruccion: R
Resultado: 3

Instruccion: R
Resultado: 4

Instruccion: R
Resultado: 5

Instruccion: R
Resultado: -1

Instruccion: R
Resultado: -1
```

```
Instruccion: R
Resultado: -1

Instruccion: R
Resultado: 2

Instruccion: R
Resultado: 3

Instruccion: R
Resultado: 4

Instruccion: R
Resultado: 5

Instruccion: MR

Resultado: 52
```

5. En el caso del quinto archivo:

```
$cat prueba5.mem

R 0
R 1024
R 3072
R 2048
R 0
R 3072
MR
```

linea 1: se quiere leer la posición 0, cache vacia, es un *miss*, se carga el bloque (0,31) sobre la via 0 del conjunto 0.

linea 2: se quiere leer la pos 1024, es un *miss*, se carga el bloque sobre la vía 1 del conjunto 0.

linea 3: Se quiere leer la pos 3072, es un *miss*, se carga el bloque sobre la via 0 del conjunto 0. Ya que 3072 en binario es 110000000000, por lo tanto tiene index 0000.

linea 4: Se quiere leer la pos 2048, es un *miss*, se carga el bloque sobre la via 1 del conjunto 0. Ya que tiene index 0000. *linea 5*: Se quiere leer la pos 0, es un *miss*, fue reemplazada anteriormente, se carga el bloque a la via 0 del conjunto 0.

linea 6: Se quiere leer la pos 3072, es un *miss*, fue reemplazada anteriormente, se carga el bloque a la via 1 del conjunto 0.

$$\boxed{MR = \frac{6}{6} \times 100 = 100}$$

```
./cache prueba5.mem
```

Cuya salida por salida estándar mediante la pruebas es:

```
Instruccion: R
Resultado: -1

Instruccion: R
Resultado: -1

Instruccion: R
Resultado: -1

Instruccion: R
Resultado: -1
```

```
Instruccion: R
Resultado: -1

Instruccion: R
Resultado: -1

Instruccion: MR

Resultado: 100
```

6. Compilación

Realizar `make` sobre la carpeta en donde se encuentran los archivos fuente, generara un ejecutable `cache`.

Github: `https://github.com/lucioll/OrgaDeCompus.git` dentro de la carpeta `TP2`.

7. Conclusión

Mediante el estudio y simulación de una cache en C, pudimos observar y comprender su funcionamiento dentro de la ejecución de un programa, como se relaciona con el CPU y la memoria principal.

Asimismo entendimos la los principios de localidad espacial y temporal, mediante las reiteradas pruebas realizadas sobre distintas direcciones de memoria; además como influye la política de reemplazo utilizada en estos aspectos.

Finalmente concluimos que no necesariamente colocando una cache a un programa este se va a ejecutar mas eficientemente, influye fuertemente como este implementada, es decir, cuantas vías utiliza, que política de escritura utiliza y otros factores mencionados anteriormente.

Referencias

[1] Hennessy, John L. and Patterson, David A., Computer Architecture: A Quantitative Approach, Third Edition, 2002.