Trabajo Práctico 2
Memorias cache

Ailín Braccelarghe, *Padrón 99366*
abraccelarghe@fi.uba.ar

Luis Waldman, *Padrón 79279*
lwaldman@fi.uba.ar

2do. Cuatrimestre de 2021
66.20 Organización de Computadoras − Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

Simularemos 2 memorias cache en distintos escenarios, con una aplicación desarrollada en C. En los siguientes capitulos explicaremos las desiciones de diseño y de implementacion que tomamos, lo que esperamos de resultado y lo que observamos. Finalmente en las conclusiones argumentaremos sobre las memorias que simulamos.

1. Introducción

Las Memoria cache son memorias rápidas que se utilizan para poder simular una memoria mas grande pero mas lenta. Las características importantes de las caches son la capacidad y la capacidad de cada bloque. También, el numero de vías y la política de reemplazo.

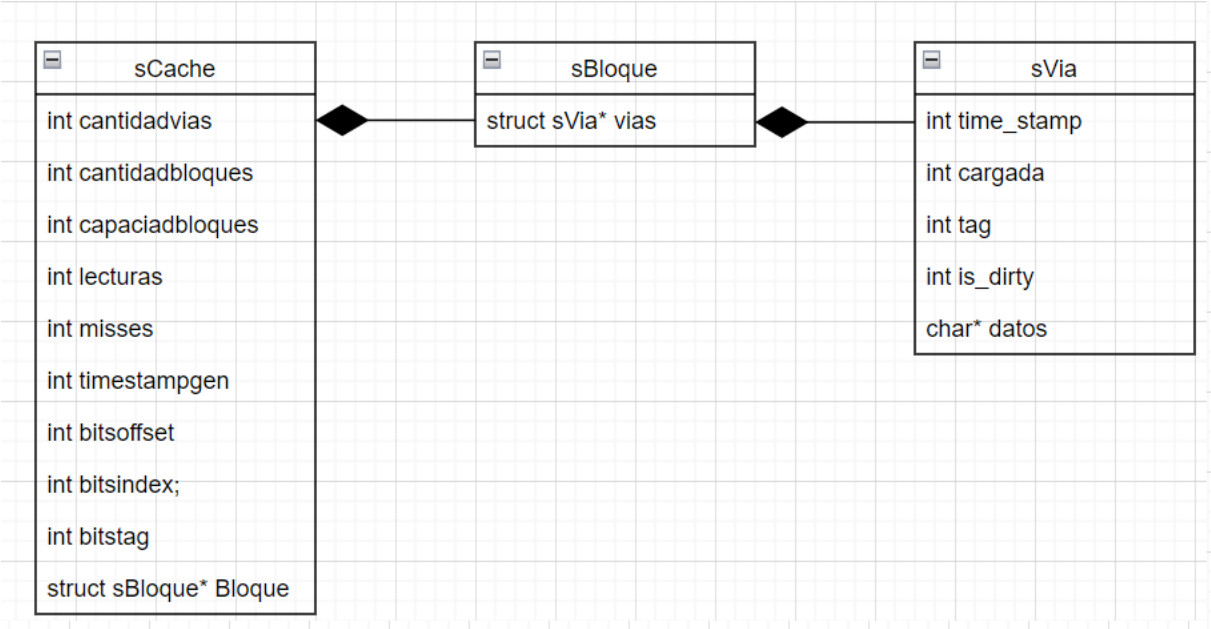
En nuestro programa, las tres primeras se pueden ingresar como parámetros y tenemos una política de reemplazo FIFO. Nuestra política de escritura sera Write Back - Writa Allocate, es decir que cuando se produce un MISS de escritura la cache busca ese bloque de datos en la memoria, y que se escribe en memoria cuando ese bloque es reemplazado.

2. Diseño e Implementación

Modelamos la memoria principal como un vector de char:

```
char * Memoria_principal;
```

Y a la memoria cache con la siguiente estructura:



toda la cache va a ser representada con la estructura sCache, que guarda en sus propiedades la cantidad de vias, bloques y la capacidad de cada bloque.

en 'lecturas' y 'misses' vamos a ir almacenando los estados para finalmente calcular la tasa de miss.

en bitsoffset, bitsIndex y bitsTag guardamos la cantidad de bits que necesitamos para cada bloque. Se calcula en base a cantidadVias, cantidadBloques y capacidadBloques

Por ultimo tenemos nuestros bloques en el vector de sBloque Bloque.

sBloque es un vector de tantas sVia como vias tenga la cache.

sVia tiene una propiedad timestamp en la que guardamos un entero que vamos incrementando cada vez que se asigna un bloque. Esta propiedad es la que nos permite luego implementar la política de FIFO.

Además, se guarda si esta cargada de memoria y el tag que guarda y en ïs$_d$*irty*" *si*, *cuandoesdesplazada*, *elbloquetienequegu*

La ultima propiedad es "datos", allí se guardan los bytes de esta parte de la cache.

Para el manejo de estas estructuras utilizamos las primitivas solicitadas en el enunciado:

```
void init()
unsigned int find_set(unsigned int address)
unsigned int find_earliest(unsigned int setnum)
unsigned int is_dirty(unsigned int way, unsigned int setnum)
unsigned int find_block (unsigned int address)
void read_block(unsigned int blocknum)
void write_block(unsigned int way, unsigned int setnum)
char read_byte(unsigned int address, unsigned char *hit)
```

y agregamos,

```
/* Devuelve la cantidad de bloques */
int blocks_count (int block_size, int ways, int cache_size) {
    return (cache_size / block_size) / ways;
}
/* Devuelve la cantidad de bits que hacen falta para representar las
direcciones de un tamaño = block_size */
int bits_offset (int block_size) {
    return log((double)block_size)/log((double)2);
```

```
}

/* Devuelve la cantidad de bits que hacen falta para representar un
numero de conjuntos  = blocks_count */
int bits_index (int blocks_count);

// La funcion end() debe desalocar los bloques de la cache.
void end()
```

En la funcion main() controlamos los parametros y leemos el archivo de entrada.

Los parametros de entrada se parsean con la funcion getopt$_l ong()$.$Lue f oelarchivodeentradaseparseacadalíne$

2.1. Análisis de la línea de comandos

Las formas de invocar a la aplicación son: para obtener información del programa

```
$ ./tp2 -h
$ ./tp2 -V
```

ejecutar la funcionalidad del programa

```
$ ./tp2 -w {#vias} -b {tamaño bloque} -c {tamaño cache} -o {archivo de salida} archivo_de_p
```

en donde únicamente la opción -o es optativa, y los argumentos de todas las opciones indicadas son necesarios, al igual que lo es el archivo de pruebas al final.

En caso de no aclararse el archivo de salida, el resultado del programa se imprimirá por pantalla.

Las opciones válidas ingresadas por línea de comandos, como indica en el enunciado, pueden ser:

```
-h, --help  ~Imprime ayuda.
-V, --version ~Version del programa.
-o, --output ~Archivo de salida.
-w, --ways ~Cantidad de vias.
-c, --cachesize ~Cantidad de kilobytes del cache.
-b, --blocksize ~Cantidad de bytes del bloque.
```

2.2. Desarrollo del Código Fuente

Dividimos el programa en los archivos main.c cache.h y cache.c En el archivo main se encuentra el funcionamiento general como el procesamiento de los argumentos de la línea de comandos, el parseo el archivo de entrada y el comportamiento que cada línea desata. En el archivo cache.h se declaran las estructuras necesarias para poder implementar la cache y las funciones requeridas en el enunciado de este trabajo, junto con algunas otras. Y por último en el archivo cache.c se encuentra el comportamiento de las funciones declaradas anteriormente, entre otras.

Hicimos uso de las siguientes librerías:

1. **getopt.h**: Funciones para el manejo de las opciones en la línea de comandos.
2. **math.h**: Funciones necesarias para el cálculo de potencias y logaritmos, utilizadas para el cálculo de bits.
3. **string.h**, para el manejo de cadenas de caracteres, durante el parseo de el archivo de entrada.
4. **stdlib.h**, para el manejo de memoria.
5. **stdio.h**, para el manejo de archivos y las salidas de datos stdout y stderr.
6. **cache.h**: Creamos este archivo que contiene las estructuras y funciones necesarias para simular el comportamiento de una cache y la memoria principal.

3. Proceso de Compilación

Se compilan los archivos con la siguiente linea de comandos:

```
$ gcc -Wall -g main.c cache.c -o tp2 -lm
```

Los archivos son main.c cache.c y cache.h El comando -lm al final fue necesario para poder linkear la librería math.h, utilizada en el programa para funciones como el cálculo del logaritmo y la potencia, ya que sin este comando no reconocía estas funciones.

4. Portabilidad

Dado que se utilizó el lenguaje de programación C, sin hacer uso de funciones específicas de sistemas operativos, o de bibliotecas comerciales, los programas pueden ser compilados en varios de ellos. De todas formas, el programa fue hecho y probado particularmente en Ubuntu-Linux .

5. Casos de Prueba

Tenemos que simular 2 caches con estas caracteristicas

1) memoria de 4KB, con 4 vias y 32bytes por grupo

2) memoria de 16KB, con 1 via y 128bytes por grupo

y tenemos 5 casos de prueba. Vamos a mostrar como se divide cada direccion en tag, index y offset para cada caso, y si esperamos MISS o HIT.

Calcularemos la tasa de miss que esparamos y correremos la aplicacion para ver si coincide.

Caso 1 - Cache 1

Direccion: 0	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 16384	Tag: 16 0 1 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 32768	Tag: 32 1 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 49152	Tag: 48 1 1 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 0	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	HIT
Direccion: 16384	Tag: 16 0 1 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	HIT
Direccion: 32768	Tag: 32 1 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	HIT
Direccion: 49152	Tag: 48 1 1 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	HIT

Tasa de miss: 50

Caso 1 - Cache 2

Direccion: 0	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 16384	Tag: 1 0 1	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 32768	Tag: 2 1 0	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 49152	Tag: 3 1 1	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 0	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 16384	Tag: 1 0 1	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 32768	Tag: 2 1 0	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 49152	Tag: 3 1 1	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS

Tasa de miss: 100

Caso 2 - Cache 1

Direccion: 0	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 1024	Tag: 0 0 0	Index: 8 0 0 0 1 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 2048	Tag: 0 0 0	Index: 16 0 0 1 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 3072	Tag: 0 0 0	Index: 24 0 0 1 1 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 4096	Tag: 0 0 0	Index: 32 0 1 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 0	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 1024	Tag: 0 0 0	Index: 8 0 0 0 1 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 2048	Tag: 0 0 0	Index: 16 0 0 1 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 3072	Tag: 0 0 0	Index: 24 0 0 1 1 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 4096	Tag: 0 0 0	Index: 32 0 1 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS

Tasa de miss: 100

Caso 2 - Cache 2

Direccion: 0	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 1024	Tag: 0 0 0	Index: 8 0 0 0 1 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 2048	Tag: 0 0 0	Index: 16 0 0 1 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 3072	Tag: 0 0 0	Index: 24 0 0 1 1 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 4096	Tag: 0 0 0	Index: 32 0 1 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	MISS
Direccion: 0	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	HIT
Direccion: 1024	Tag: 0 0 0	Index: 8 0 0 0 1 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	HIT
Direccion: 2048	Tag: 0 0 0	Index: 16 0 0 1 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	HIT
Direccion: 3072	Tag: 0 0 0	Index: 24 0 0 1 1 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	HIT
Direccion: 4096	Tag: 0 0 0	Index: 32 0 1 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0 0	HIT

Tasa de miss: 50

Caso 3 - Cache 1

Direccion: 128	Tag: 0 0 0 0 0 0 0	Index: 4 0 0 1 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 129	Tag: 0 0 0 0 0 0 0	Index: 4 0 0 1 0 0	Offset: 1 0 0 0 0 1	HIT
Direccion: 130	Tag: 0 0 0 0 0 0 0	Index: 4 0 0 1 0 0	Offset: 2 0 0 0 1 0	HIT
Direccion: 131	Tag: 0 0 0 0 0 0 0	Index: 4 0 0 1 0 0	Offset: 3 0 0 0 1 1	HIT
Direccion: 1152	Tag: 1 0 0 0 0 0 1	Index: 4 0 0 1 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 2176	Tag: 2 0 0 0 0 1 0	Index: 4 0 0 1 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 3200	Tag: 3 0 0 0 0 1 1	Index: 4 0 0 1 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 4224	Tag: 4 0 0 0 1 0 0	Index: 4 0 0 1 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 128	Tag: 0 0 0 0 0 0 0	Index: 4 0 0 1 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 129	Tag: 0 0 0 0 0 0 0	Index: 4 0 0 1 0 0	Offset: 1 0 0 0 0 1	HIT
Direccion: 130	Tag: 0 0 0 0 0 0 0	Index: 4 0 0 1 0 0	Offset: 2 0 0 0 1 0	HIT
Direccion: 131	Tag: 0 0 0 0 0 0 0	Index: 4 0 0 1 0 0	Offset: 3 0 0 0 1 1	HIT

Tasa de miss: 50

Caso 3 - Cache 2

Direccion: 128	Tag: 0 0 0	Index: 1 0 0 0 0 0 0 1	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 129	Tag: 0 0 0	Index: 1 0 0 0 0 0 0 1	Offset: 1 0 0 0 0 0 0 1	HIT
Direccion: 130	Tag: 0 0 0	Index: 1 0 0 0 0 0 0 1	Offset: 2 0 0 0 0 0 1 0	HIT
Direccion: 131	Tag: 0 0 0	Index: 1 0 0 0 0 0 0 1	Offset: 3 0 0 0 0 0 1 1	HIT
Direccion: 1152	Tag: 0 0 0	Index: 9 0 0 0 1 0 0 1	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 2176	Tag: 0 0 0	Index: 17 0 0 1 0 0 0 1	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 3200	Tag: 0 0 0	Index: 25 0 0 1 1 0 0 1	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 4224	Tag: 0 0 0	Index: 33 0 1 0 0 0 0 1	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 128	Tag: 0 0 0	Index: 1 0 0 0 0 0 0 1	Offset: 0 0 0 0 0 0 0 0	HIT
Direccion: 129	Tag: 0 0 0	Index: 1 0 0 0 0 0 0 1	Offset: 1 0 0 0 0 0 0 1	HIT
Direccion: 130	Tag: 0 0 0	Index: 1 0 0 0 0 0 0 1	Offset: 2 0 0 0 0 0 1 0	HIT
Direccion: 131	Tag: 0 0 0	Index: 1 0 0 0 0 0 0 1	Offset: 3 0 0 0 0 0 1 1	HIT

Tasa de miss: 41

Caso 4 - Cache 1

Direccion: 0	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 1	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 1 0 0 0 0 1	HIT
Direccion: 2	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 2 0 0 0 1 0	HIT
Direccion: 3	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 3 0 0 0 1 1	HIT
Direccion: 4	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 4 0 0 1 0 0	HIT
Direccion: 0	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	HIT
Direccion: 1	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 1 0 0 0 0 1	HIT
Direccion: 2	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 2 0 0 0 1 0	HIT
Direccion: 3	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 3 0 0 0 1 1	HIT
Direccion: 4	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 4 0 0 1 0 0	HIT
Direccion: 4096	Tag: 4 0 0 0 1 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 8192	Tag: 8 0 0 1 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 0	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	HIT
Direccion: 1	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 1 0 0 0 0 1	HIT
Direccion: 2	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 2 0 0 0 1 0	HIT
Direccion: 3	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 3 0 0 0 1 1	HIT
Direccion: 4	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 4 0 0 1 0 0	HIT

Tasa de miss: 17

Caso 4 - Cache 2

Direccion: 0	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 1	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 1 0 0 0 0 0 0 1	HIT
Direccion: 2	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 2 0 0 0 0 0 1 0	HIT
Direccion: 3	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 3 0 0 0 0 0 1 1	HIT
Direccion: 4	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 4 0 0 0 0 1 0 0	HIT
Direccion: 0	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	HIT
Direccion: 1	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 1 0 0 0 0 0 0 1	HIT
Direccion: 2	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 2 0 0 0 0 0 1 0	HIT
Direccion: 3	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 3 0 0 0 0 0 1 1	HIT
Direccion: 4	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 4 0 0 0 0 1 0 0	HIT
Direccion: 4096	Tag: 0 0 0	Index: 32 0 1 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 8192	Tag: 0 0 0	Index: 64 1 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 0	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	HIT
Direccion: 1	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 1 0 0 0 0 0 0 1	HIT
Direccion: 2	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 2 0 0 0 0 0 1 0	HIT
Direccion: 3	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 3 0 0 0 0 0 1 1	HIT
Direccion: 4	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 4 0 0 0 0 1 0 0	HIT

Tasa de miss: 17

Caso 5 - Cache 1

Direccion: 4096	Tag: 4 0 0 0 1 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 8192	Tag: 8 0 0 1 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 4096	Tag: 4 0 0 0 1 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	HIT
Direccion: 0	Tag: 0 0 0 0 0 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	MISS
Direccion: 4096	Tag: 4 0 0 0 1 0 0	Index: 0 0 0 0 0 0	Offset: 0 0 0 0 0 0	HIT

Tasa de miss: 60

Caso 5 - Cache 2

Direccion: 4096	Tag: 0 0 0	Index: 32 0 1 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 8192	Tag: 0 0 0	Index: 64 1 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 4096	Tag: 0 0 0	Index: 32 0 1 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	HIT
Direccion: 0	Tag: 0 0 0	Index: 0 0 0 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	MISS
Direccion: 4096	Tag: 0 0 0	Index: 32 0 1 0 0 0 0 0	Offset: 0 0 0 0 0 0 0 0	HIT

Tasa de miss: 60

Ahora, Ejecutamos las pruebas ....

5.1. Línea de comandos

Se realizaron las siguientes pruebas para garantizar el funcionamiento de la aplicación en diferentes escenarios:

Sin parámetros

```
$ ./tp2
ERROR: missing arguments. Use -h for help.
```

Sin algun parametro necesario

```
$ ./tp2 -w 2 -b 8 test.txt
ERROR: missing arguments. Use -h for help.
```

Con una opción inválida.

```
$ ./tp2 -a
./tp2: invalid option -- 'a'
Use -h or --help if needed.
```

Sin los argumentos requeridos

```
 $ ./tp2 -w 2 -b -c 7 test.txt
ERROR: missing arguments. Use -h for help.
```

Sin archivo de entrada

```
$ ./tp2 -w 2 -b -c 16
ERROR: missing arguments. Use -h for help.
```

Con archivo de entrada inexistente

```
$ ./tp2 -w 2 -b 8 -c 16 not_existent_file.txt
Error while opening input file not_existent_file.txt
```

Si la relación cantidad de vías - tamaño de bloque no es coherente

```
$ ./tp2 -w 3 -b 16 -c 8 test.txt
ERROR: bad argument input, relation between block size and ways count can not be applied
```

Si el tamaño de bloque es mayor al tamño de cache

```
$ ./tp2 -w 2 -b 2000 -c 1 test.txt
ERROR: bad argument input, block size can not be larger than cache size
```

Caso exitoso

```
$ ./tp2 -w 2 -b 16 -c 8 test.txt
WRITE address 34567 data 86 - MISS
READ address 45352, data 0 - MISS
READ address 34564, data 0 - HIT
MISS RATE: 66
```

Caso exitoso

```
$ ./tp2 -w 2 -b 16 -c 8 test.txt
WRITE address 34567 data 86 - MISS
READ address 45352, data 0 - MISS
READ address 34564, data 0 - HIT
MISS RATE: 66
```

Si la direccion de memoria a Leer o Escribir es mayor a la capacidad de la Memoria Principal

```
$ ./tp2 -w 2 -b 16 -c 8 test.txt
WRITE: error pos 73254 dont exists
READ: error pos 82132 dont exists
READ address 34564, data 0 - MISS
MISS RATE: 100
```

Si no hay datos para calcular el miss rate

```
$ ./tp2 -w 2 -b 16 -c 8 test.txt
MISS RATE: NO DATA
```

5.2. Funcionamiento de la aplicación

A continuación se ven los casos de prueba proporcionados en el enunciado y los resultado obtenidos.

Caso 1 - Cache 1

```
$ ./tp2 -w 4 -c 4 -b 32 prueba1.mem
WRITE address 0 data 255 - MISS
WRITE address 16384 data 254 - MISS
WRITE address 32768 data 248 - MISS
WRITE address 49152 data 96 - MISS
READ address 0, data -1 - HIT
READ address 16384, data -2 - HIT
READ address 32768, data -8 - HIT
READ address 49152, data 96 - HIT
MISS RATE: 50
```

Caso 2 - Cache 1

```
$ ./tp2 -w 4 -c 4 -b 32 prueba2.mem
WRITE address 0 data 123 - MISS
WRITE address 1024 data 234 - MISS
WRITE address 2048 data 33 - MISS
WRITE address 3072 data 44 - MISS
WRITE address 4096 data 55 - MISS
READ address 0, data 123 - MISS
READ address 1024, data -22 - MISS
READ address 2048, data 33 - MISS
READ address 3072, data 44 - MISS
READ address 4096, data 55 - MISS
MISS RATE: 100
```

Caso 3 - Cache 1

```
$ ./tp2 -w 4 -c 4 -b 32 prueba3.mem
WRITE address 128 data 1 - MISS
WRITE address 129 data 2 - HIT
WRITE address 130 data 3 - HIT
WRITE address 131 data 4 - HIT
READ address 1152, data 0 - MISS
READ address 2176, data 0 - MISS
READ address 3200, data 0 - MISS
```

```
READ address 4224, data 0 - MISS
READ address 128, data 1 - MISS
READ address 129, data 2 - HIT
READ address 130, data 3 - HIT
READ address 131, data 4 - HIT
MISS RATE: 50
```

Caso 4 - Cache 1

```
$ ./tp2 -w 4 -c 4 -b 32 prueba4.mem
WRITE address 0 data 256 - MISS
WRITE address 1 data 2 - HIT
WRITE address 2 data 3 - HIT
WRITE address 3 data 4 - HIT
WRITE address 4 data 5 - HIT
READ address 0, data 0 - HIT
READ address 1, data 2 - HIT
READ address 2, data 3 - HIT
READ address 3, data 4 - HIT
READ address 4, data 5 - HIT
READ address 4096, data 0 - MISS
READ address 8192, data 0 - MISS
READ address 0, data 0 - HIT
READ address 1, data 2 - HIT
READ address 2, data 3 - HIT
READ address 3, data 4 - HIT
READ address 4, data 5 - HIT
MISS RATE: 17
```

Caso 5 - Cache 1

```
$ ./tp2 -w 4 -c 4 -b 32 prueba5.mem
READ: error pos 131072 dont exists
READ address 4096, data 0 - MISS
READ address 8192, data 0 - MISS
READ address 4096, data 0 - HIT
READ address 0, data 0 - MISS
READ address 4096, data 0 - HIT
MISS RATE: 60
```

Caso 1 - Cache 2

```
$ ./tp2 -w 1 -c 16 -b 128 prueba1.mem
WRITE address 0 data 255 - MISS
WRITE address 16384 data 254 - MISS
WRITE address 32768 data 248 - MISS
WRITE address 49152 data 96 - MISS
READ address 0, data -1 - MISS
READ address 16384, data -2 - MISS
READ address 32768, data -8 - MISS
READ address 49152, data 96 - MISS
MISS RATE: 100
```

Caso 2 - Cache 2

```
$ ./tp2 -w 1 -c 16 -b 128 prueba2.mem
WRITE address 0 data 123 - MISS
WRITE address 1024 data 234 - MISS
WRITE address 2048 data 33 - MISS
WRITE address 3072 data 44 - MISS
WRITE address 4096 data 55 - MISS
READ address 0, data 123 - HIT
READ address 1024, data -22 - HIT
READ address 2048, data 33 - HIT
READ address 3072, data 44 - HIT
READ address 4096, data 55 - HIT
MISS RATE: 50
```

Caso 3 - Cache 2

```
$ ./tp2 -w 1 -c 16 -b 128 prueba3.mem
WRITE address 128 data 1 - MISS
WRITE address 129 data 2 - HIT
WRITE address 130 data 3 - HIT
WRITE address 131 data 4 - HIT
READ address 1152, data 0 - MISS
READ address 2176, data 0 - MISS
READ address 3200, data 0 - MISS
READ address 4224, data 0 - MISS
READ address 128, data 1 - HIT
READ address 129, data 2 - HIT
READ address 130, data 3 - HIT
READ address 131, data 4 - HIT
MISS RATE: 41
```

Caso 4 - Cache 2

```
$ ./tp2 -w 1 -c 16 -b 128 prueba4.mem
WRITE address 0 data 256 - MISS
WRITE address 1 data 2 - HIT
WRITE address 2 data 3 - HIT
WRITE address 3 data 4 - HIT
WRITE address 4 data 5 - HIT
READ address 0, data 0 - HIT
READ address 1, data 2 - HIT
READ address 2, data 3 - HIT
READ address 3, data 4 - HIT
READ address 4, data 5 - HIT
READ address 4096, data 0 - MISS
READ address 8192, data 0 - MISS
READ address 0, data 0 - HIT
READ address 1, data 2 - HIT
READ address 2, data 3 - HIT
READ address 3, data 4 - HIT
READ address 4, data 5 - HIT
MISS RATE: 17
```

Caso 5 - Cache 2

```
$ ./tp2 -w 1 -c 16 -b 128 prueba5.mem
READ: error pos 131072 dont exists
READ address 4096, data 0 - MISS
READ address 8192, data 0 - MISS
READ address 4096, data 0 - HIT
READ address 0, data 0 - MISS
READ address 4096, data 0 - HIT
MISS RATE: 60
```

5.3. Control del manejo de memoria

Dado que trabajamos con memoria dinámica, tenemos que verificar que no haya fugas de memoria, ni errores. Para ello utilizamos la herramienta Valgrind en la máquina Host, en este caso es una máquina Linux con Ubuntu.

```
$ valgrind -v --tool=memcheck --leak-check=yes ./main -w 1 -c 16 -b 128 prueba4.mem
==1933== HEAP SUMMARY:
==1933==     in use at exit: 0 bytes in 0 blocks
==1933==   total heap usage: 261 allocs, 261 frees, 91,608 bytes allocated
==1933==
==1933== All heap blocks were freed -- no leaks are possible
==1933==
==1933== Use --track-origins=yes to see where uninitialised values come from
==1933== ERROR SUMMARY: 10 errors from 5 contexts (suppressed: 0 from 0)
==1933==
==1933== 2 errors in context 1 of 5:
==1933== Conditional jump or move depends on uninitialised value(s)
==1933==    at 0x4A2686E: __vfprintf_internal (vfprintf-internal.c:1687)
==1933==    by 0x4A10DE9: fprintf (fprintf.c:32)
==1933==    by 0x10985E: main (main.c:157)
==1933==
==1933==
==1933== 2 errors in context 2 of 5:
==1933== Conditional jump or move depends on uninitialised value(s)
==1933==    at 0x4A273A8: __vfprintf_internal (vfprintf-internal.c:1687)
==1933==    by 0x4A10DE9: fprintf (fprintf.c:32)
==1933==    by 0x10985E: main (main.c:157)
==1933==
==1933==
==1933== 2 errors in context 3 of 5:
==1933== Conditional jump or move depends on uninitialised value(s)
==1933==    at 0x4A0A82D: _itoa_word (_itoa.c:179)
==1933==    by 0x4A266F4: __vfprintf_internal (vfprintf-internal.c:1687)
==1933==    by 0x4A10DE9: fprintf (fprintf.c:32)
==1933==    by 0x10985E: main (main.c:157)
==1933==
==1933==
==1933== 2 errors in context 4 of 5:
==1933== Use of uninitialised value of size 8
==1933==    at 0x4A0A81B: _itoa_word (_itoa.c:179)
==1933==    by 0x4A266F4: __vfprintf_internal (vfprintf-internal.c:1687)
==1933==    by 0x4A10DE9: fprintf (fprintf.c:32)
==1933==    by 0x10985E: main (main.c:157)
==1933==
==1933==
==1933== 2 errors in context 5 of 5:
==1933== Conditional jump or move depends on uninitialised value(s)
==1933==    at 0x4A26AD8: __vfprintf_internal (vfprintf-internal.c:1687)
==1933==    by 0x4A10DE9: fprintf (fprintf.c:32)
```

```
==1933==    by 0x10985E: main (main.c:157)
==1933==
==1933== ERROR SUMMARY: 10 errors from 5 contexts (suppressed: 0 from 0)
```

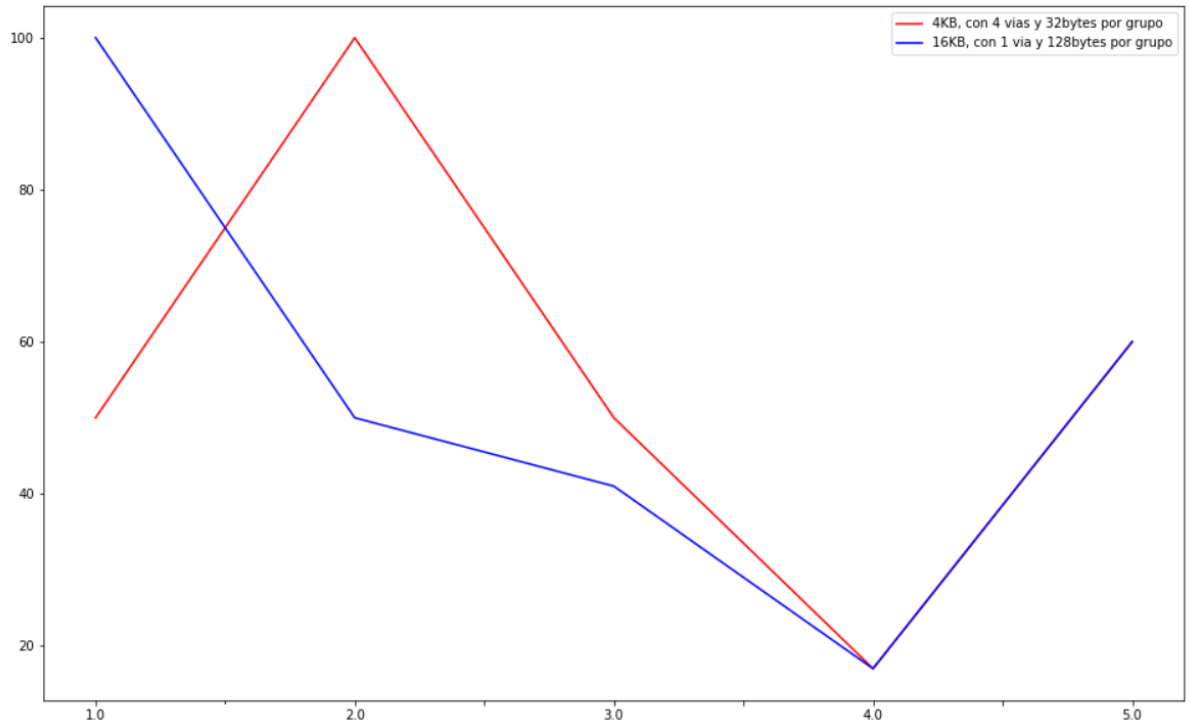
Podemos ver que no hay perdida de memoria, aunque si hay errores ocasionados por un valor no inicializado en la linea 157 de main.c

```
157        fprintf (stdout, "READ address %d, data %d", memory_pos, data);
```

que no pudimos solucionar, ya que las variables en cuestion en esa linea fueron inicializadas.

6. Conclusión

Para finalizar, graficamos como le fue a cada memoria con cada uno de estos casos de prueba.



A. Enunciado

Adjunto en la entrega

B. Código fuente

B.1. main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <getopt.h>
#include "cache.h"



void print_help() {
fprintf (stdout, "Usage: \n tp2 -h \n tp2 -V \n tp2 options archivo \nOptions: \n -h, --help  ~Imprime
fprintf (stdout, " -V, --version ~Version del programa.\n -o, --output ~Archivo de salida.\n");
fprintf (stdout, " -w, --ways ~Cantidad de vias.\n -c, --cachesize ~Cantidad de kilobytes del cache.");
fprintf (stdout, "\n -b, --blocksize ~Cantidad de bytes del bloque.\nExamples:\n tp2 -w 4 -cs 8 -bs 16

}

int main(int argc, char *argv[]) {

char* output_file_name = NULL;
FILE* output_file = NULL;
    int ways = -1, block_size = -1, cache_size = -1;

while (1) {

static struct option long_options[] =
{
{"help",     no_argument,       0, 'h'},
{"version",     no_argument,       0, 'V'},
{"output",  required_argument, 0, 'o'},
{"ways",  required_argument, 0, 'w'},
{"cachesize",  required_argument, 0, 'c'},
{"blocksize",  required_argument, 0, 'b'},
{0, 0, 0, 0}
};

int option_index = 0;

int opt = getopt_long (argc, argv, "hVo:w:c:b:", long_options, &option_index);

if (opt == -1) {
break;
}

switch (opt){
case 'h':
print_help();
return OK;
break;

case 'V':
fprintf (stdout, "Version 1.0\n");
return OK;
break;
```

```
case 'o': ;
output_file_name = optarg;
break;

case 'w':
ways = atoi(optarg);
break;

case 'c': ;
int argument = atoi(optarg);
if (argument <= 0) {
fprintf (stderr, "ERROR: bad argument input, cache size can not be equal or less than 0\n");
return ERROR;
}
if (argument > MEMORY_SIZE) {
fprintf (stderr, "ERROR: bad argument input, cache size can not be larger than memory size (
return ERROR;
}

int bits = 10 + (log(argument)/log(2));
cache_size = pow(2, bits);
break;

case 'b':
block_size = atoi(optarg);
break;

default:
fprintf (stderr, "Use -h or --help if needed.\n");
return ERROR;
}

};

    if (ways <= 0 || cache_size <= 0 || block_size <= 0 ) {
        fprintf (stderr, "ERROR: missing arguments. Use -h for help.\n");
        return ERROR;
    }

if (block_size > cache_size) {
fprintf (stderr, "ERROR: bad argument input, block size can not be larger than cache size\n"
return ERROR;
}

int blocks = cache_size / block_size;

//chequeo que no me queden bloques fuera de las vias
if ((blocks % ways != 0)) {
fprintf (stderr, "ERROR: bad argument input, relation between block size and ways count can
return ERROR;
}

if (optind >= argc) {
fprintf(stderr, "ERROR: missing input file\n");
        return ERROR;
}

    char* input_file_name = argv[optind];
    FILE* input_file = fopen(input_file_name, "r");
    if (!input_file) {
        fprintf(stderr, "Error while opening input file %s\n", input_file_name);
```

```
        return ERROR;
    }

if (output_file_name) {
output_file = fopen(output_file_name, "w");
if (!output_file) {
fprintf(stderr, "Error while opening output file %s\n", output_file_name);
fclose(input_file);
return ERROR;
}

stdout = output_file;
stderr = output_file;
}

int error = begin(block_size, ways, cache_size);
if (error) {
fprintf(stderr, "Memory Error\n");
fclose(input_file);
if (output_file) fclose(output_file);
return ERROR;
}


char x[6];
int memory_pos = 0;
unsigned char hit = 0;


    while (fscanf(input_file, "%5s", x) != EOF)
{
        if (strcmp(x, "init") == 0) init();

else if (strcmp(x, "R") == 0)
{
fscanf(input_file, "%d", &memory_pos);
if (memory_pos < MEMORY_SIZE) {

char data = read_byte(memory_pos, &hit);

fprintf (stdout, "READ address %d, data %d", memory_pos, data);

if (hit == 1) fprintf(stdout, " - HIT\n");
else fprintf (stdout, " - MISS\n");
}
else fprintf (stderr, "READ: error pos %d dont exists\n", memory_pos);
}
else if (strcmp(x, "W") == 0)
{
fscanf(input_file, "%7s", x);
char* number = strtok(x, ",");
memory_pos = atoi(number);

if (memory_pos < MEMORY_SIZE) {

int value;
fscanf(input_file, "%d", &value);

fprintf(stdout, "WRITE address %d data %d", memory_pos, value);

write_byte(memory_pos, value, &hit);
```

```
if (hit == 1) fprintf(stdout, " - HIT\n");
else fprintf (stdout, " - MISS\n");
}
else fprintf (stderr, "WRITE: error pos %d dont exists\n", memory_pos);
}
else if (strcmp(x, "MR") == 0)
{
char mr = get_miss_rate();
if (mr == NO_MISS_RATE) fprintf(stdout, "MISS RATE: NO DATA\n");
else fprintf(stdout, "MISS RATE: %d\n", mr);
}
    }

    fclose(input_file);
if (output_file) fclose(output_file);

end();

return OK;
}
```

B.2. cache.h

```
#define BITS_DIR_MEM 16
#define MEMORY_SIZE 65536
#define OK 0
#define ERROR 1
#define NO_MISS_RATE '-'
#define EFES 0xFFFFFFFF


typedef struct sVia
{
    int time_stamp;
    int cargada;
    int tag;
    int is_dirty;
char * datos;
}sVia;


typedef struct sBloque
{

    struct sVia* vias;
}sBloque;

typedef struct sCache
{

    int cantidadvias;
    int cantidadbloques;
    int capaciadbloques;


    int lecturas;
    int misses;
int timestampgen;

int bitsoffset;
    int bitsindex;
    int bitstag;
    struct sBloque* Bloque;
}sCache;

char * Memoria_principal;
sCache cache;


/*
La funcion begin inicializa la estructura segun los parametros. Si hay un error de memoria devuelve NUL
*/
int begin (int block_size, int ways, int cache_size);

/*
La funcion init() debe inicializar los bloques de la cache como invalidos,
la memoria simulada en 0 y la tasa de misses a 0.
*/
void init();
```

```
/*
La funcion end() debe desalocar los bloques de la cache.
*/
void end();

/*
La funcion find set(unsigned int address) debe devolver el conjunto
de cache al que mapea la direccion address.
*/
unsigned int find_offset(unsigned int address);




/*
La funcion find set(unsigned int address) debe devolver el conjunto
de cache al que mapea la direccion address.
*/
unsigned int find_set(unsigned int address);


/*
La funcion find set(unsigned int address) debe devolver el conjunto
de cache al que mapea la direccion address.
*/
unsigned int find_tag(unsigned int address);



/*
La funcion find block(unsigned int address) debe devolver el bloque
de memoria correspondiente a la direccion address.
*/

unsigned int find_block (unsigned int address);



/*
La funcion find earliest(unsigned int setnum) debe devolver el
bloque mas 'antiguo' dentro de un conjunto, utilizando el campo correspondiente
de los metadatos de los bloques del conjunto.
*/
unsigned int find_earliest(unsigned int setnum);


/*
La funcion is dirty(unsigned int way,unsigned int setnum) debe
devolver el estado del bit D del bloque correspondiente.
*/
unsigned int is_dirty(unsigned int way, unsigned int setnum);


/*
La funcion write block(unsigned int way, unsigned int
setnum) debe escribir en las posiciones correspondientes de memoria
el contenido del bloque setnum de la via way.
*/
void write_block(unsigned int way, unsigned int setnum);
```

```
/*
La funcion read block(unsigned int blocknum) debe leer el bloque
blocknum de memoria y guardarlo en el lugar que le corresponda en
la memoria cache.
*/

void read_block(unsigned int blocknum);



/*
La funcion find_via(unsigned int address) busca la via en la que esta guardada address
y devuelve 0xFFFFFFFF si no esta
*/
int find_via(unsigned int address);

/*
La funcion read byte(unsigned int address, char *hit) debe retornar
el valor correspondiente a la posicioon de memoria address,
buscnandolo primero en el cache, y escribir 1 en *hit si es un hit y 0 si
es un miss.
*/
char read_byte(unsigned int address, unsigned char *hit);


/*
La funcion write byte(unsigned int address, char value) debe
escribir el valor value en la posicion correcta del bloque que corresponde
a address, si esta en el cache, y escribir 1 en *hit si es un hit
y 0 si es un miss.
*/
void write_byte(unsigned int address, char value, unsigned char *hit);

/*
La funcion get miss rate() debe devolver el porcentaje de misses
desde que se inicializo el cache.
read byte() y write byte() solo deben interactuar con la memoria
a traves de las otras primitivas.
*/
char get_miss_rate();
```

B.3. cache.c

```
#include <math.h>
#include "cache.h"
#include <stdlib.h>

/* Devuelve la cantidad de bloques */
int blocks_count (int block_size, int ways, int cache_size) {
    return (cache_size / block_size) / ways;
}
/* Devuelve la cantidad de bits que hacen falta para representar las direcciones de un tamañ
int bits_offset (int block_size) {
    return log((double)block_size)/log((double)2);
}

/* Devuelve la cantidad de bits que hacen falta para representar un numero de conjuntos  = b
int bits_index (int blocks_count) {
    return log((double)blocks_count)/log((double)2);
}


int begin (int block_size, int ways, int cache_size) {

    cache.capaciadbloques = block_size;
cache.cantidadvias = ways;
cache.cantidadbloques = blocks_count(block_size, ways, cache_size);
cache.bitsoffset = bits_offset(block_size);
cache.bitsindex = bits_index(cache.cantidadbloques);
    cache.bitstag = BITS_DIR_MEM - cache.bitsindex - cache.bitsoffset;
cache.lecturas = 0;
    cache.misses = 0;

Memoria_principal = malloc(sizeof(char) *  MEMORY_SIZE);
if (Memoria_principal == NULL) return ERROR;

cache.Bloque = malloc(sizeof(struct sBloque) * cache.cantidadbloques);
if (cache.Bloque == NULL) {
free(Memoria_principal);
return ERROR;
}

int cont = 0;
int contvias = 0;

for (cont = 0; cont < cache.cantidadbloques; cont++)
{
cache.Bloque[cont].vias = malloc(sizeof(struct sVia) * cache.cantidadvias);
if (cache.Bloque[cont].vias == NULL) {
for (int i = 0; i < cont; i++) free(cache.Bloque[i].vias);
free(cache.Bloque);
free(Memoria_principal);
return ERROR;
}

for (contvias = 0; contvias < cache.cantidadvias; contvias++)
{
cache.Bloque[cont].vias[contvias].is_dirty = 0;
cache.Bloque[cont].vias[contvias].cargada = 0;
cache.Bloque[cont].vias[contvias].tag = 0;
```

```
cache.Bloque[cont].vias[contvias].time_stamp = -1;
cache.Bloque[cont].vias[contvias].datos = malloc(sizeof(char) * cache.capaciadbloques);

if (cache.Bloque[cont].vias[contvias].datos == NULL) {
for (int i = 0; i < cont; i++) {
for (int j = 0; j < contvias; j++) {
free(cache.Bloque[i].vias[j].datos);
}
free(cache.Bloque[i].vias);
}
free(cache.Bloque);
free(Memoria_principal);
return ERROR;
}
}
}
return OK;
}


/*
funcion init() debe inicializar los bloques de la cache como invalidos,
la memoria simulada en 0 y la tasa de misses a 0.
*/
void init() {

for (int i = 0; i < MEMORY_SIZE; i++) {
Memoria_principal[i] = 0;
}

for (int cont = 0; cont < cache.cantidadbloques; cont++)
{
for (int contvias = 0; contvias < cache.cantidadvias; contvias++)
{
cache.Bloque[cont].vias[contvias].is_dirty = 0;
cache.Bloque[cont].vias[contvias].cargada = 0;
cache.Bloque[cont].vias[contvias].tag = 0;
cache.Bloque[cont].vias[contvias].time_stamp = -1;
}
}
}


/*
La funcion end() debe desalocar los bloques de la cache.
*/
void end()
{

int cont = 0;
int contvias = 0;

for (cont = 0; cont < cache.cantidadbloques; cont++)
{

for (contvias = 0; contvias < cache.cantidadvias; contvias++)
{

free(cache.Bloque[cont].vias[contvias].datos);
}
```

```
free(cache.Bloque[cont].vias);

}
free(cache.Bloque);
    free(Memoria_principal);
}

void free_memory () {

end();
free(Memoria_principal);
return;
}

/*
La funcion find set(unsigned int address) debe devolver el conjunto
de cache al que mapea la direccion address.
*/
unsigned int find_offset(unsigned int address)
{

unsigned int efes = EFES;
efes = (efes >> (32 - cache.bitsoffset));
return address & efes;
}




/*
La funcion find set(unsigned int address) debe devolver el conjunto
de cache al que mapea la direccion address.
*/
unsigned int find_set(unsigned int address)
{
unsigned int sinoffset = (address >> cache.bitsoffset);
unsigned int efes = EFES;
efes = (efes >> (32 - cache.bitsindex));
unsigned int ret = efes & sinoffset;
return ret;
}


/*
La funcion find set(unsigned int address) debe devolver el conjunto
de cache al que mapea la direccion address.
*/
unsigned int find_tag(unsigned int address)
{
return (address >> (cache.bitsoffset + cache.bitsindex));
}



/*
La funcion find block(unsigned int address) debe devolver el bloque
de memoria correspondiente a la direccion address.
*/

unsigned int find_block (unsigned int address)
{
```

```
/// EL INDICE DE BLOQUE EN MEMORIA FISICA? QUE DIFERENCIA HAY CON find_set
unsigned int sinoffset = (address >> cache.bitsoffset);
return sinoffset;
}



/*
La funcion find earliest(unsigned int setnum) debe devolver el
bloque mas 'antiguo' dentro de un conjunto, utilizando el campo correspondiente
de los metadatos de los bloques del conjunto.
*/
unsigned int find_earliest(unsigned int setnum)
{
struct sBloque bloque = cache.Bloque[setnum];
int cont = 0;
struct sVia viaearliest;

// LE ASIGNA EL MAYOR NUMERO POSIBLE
unsigned int ret = EFES;


for (cont = 0; cont < cache.cantidadvias; cont++)
{

struct sVia viacandidata = bloque.vias[cont];

if (viacandidata.cargada == 0)
{
ret = cont;
viaearliest = viacandidata;
}
else
{
if (ret == EFES)
{
ret = cont;
viaearliest = viacandidata;
}
else
{
if
((viaearliest.cargada == 1)
&& (viaearliest.time_stamp > viacandidata.time_stamp))
{
ret = cont;
viaearliest = viacandidata;
}
}

}


}

return ret;

}


/*
```

```
La funcion is dirty(unsigned int way,unsigned int setnum) debe
devolver el estado del bit D del bloque correspondiente.
*/
unsigned int is_dirty(unsigned int way, unsigned int setnum)
{
return cache.Bloque[setnum].vias[way].is_dirty;
}


/*
La funcion write block(unsigned int way, unsigned int
setnum) debe escribir en las posiciones correspondientes de memoria
el contenido del bloque setnum de la via way.
*/
void write_block(unsigned int way, unsigned int setnum)
{
unsigned int cont;

    struct sVia via = cache.Bloque[setnum].vias[way];


// Desplazo los BITS de TAG y le sumo el numero de bloque
unsigned int init_memory = via.tag << cache.bitsindex;
init_memory += setnum;

// Desplazo los BITS de OFFSET
init_memory = init_memory << cache.bitsoffset;

// COPIA LOS DATOS
for (cont = 0; cont < cache.capaciadbloques; cont++)
{
Memoria_principal[init_memory + cont] = via.datos[cont];
}


}




/*
La funcion read block(unsigned int blocknum) debe leer el bloque
blocknum de memoria y guardarlo en el lugar que le corresponda en
la memoria cache.
*/

void read_block(unsigned int blocknum)
{
unsigned int memoria_init = (blocknum << cache.bitsoffset);
unsigned int cont = 0;


// CONJUNTO Y VIA QUE VAMOS A COPIAR
unsigned int setnum = find_set(memoria_init);
unsigned int earliest = find_earliest(setnum);


// VA A COPIAR, SI NO HAY UN ULTIMO LO AGREGO EN EL PRIMERO
if (earliest == EFES)
earliest = 0;
```

```
struct sVia via = cache.Bloque[setnum].vias[earliest];


if (via.is_dirty && via.cargada)
write_block(earliest, setnum);


// COPIA LOS DATOS
for (cont = 0; cont < cache.capaciadbloques; cont++)
{
cache.Bloque[setnum].vias[earliest].datos[cont] = Memoria_principal[memoria_init + cont];
}

cache.Bloque[setnum].vias[earliest].is_dirty = 0;
cache.Bloque[setnum].vias[earliest].tag = blocknum >> cache.bitsindex;
cache.Bloque[setnum].vias[earliest].time_stamp = cache.timestampgen++;
cache.Bloque[setnum].vias[earliest].cargada = 1;
}



/*
La funcion find_via(unsigned int address) busca la via en la que esta guardada address
y devuelve 0xFFFFFFFF si no esta
*/
int find_via(unsigned int address)
{
unsigned int set = find_set(address);
unsigned int tag = find_tag(address);
unsigned int encontradoenvia = EFES;

unsigned int cont = 0;

// Recorro las vias del conjunto
for (cont = 0; cont < cache.cantidadvias; cont++)
{
if (cache.Bloque[set].vias[cont].cargada == 1)
if (cache.Bloque[set].vias[cont].tag == tag)
{
encontradoenvia = cont;
}
}


return encontradoenvia;

}

/*
La funcion read byte(unsigned int address, char *hit) debe retornar
el valor correspondiente a la posicioon de memoria address,
buscnandolo primero en el cache, y escribir 1 en *hit si es un hit y 0 si
es un miss.
*/
char read_byte(unsigned int address, unsigned char *hit)
{
unsigned int offset = find_offset(address);
unsigned int set = find_set(address);
unsigned int block = find_block(address);
unsigned int encontradoenvia = find_via(address);
```

```
cache.lecturas++;


// SI NO LO ENCUENTRA LO VA A LEER
if (encontradoenvia == EFES)
{
encontradoenvia = find_earliest(set);
read_block(block);

// MARCA SALIDA
hit[0] = 0;
cache.misses++;
}
else
{

// MARCA SALIDA
hit[0] = 1;
}
return cache.Bloque[set].vias[encontradoenvia].datos[offset];
}


/*
La funcion write byte(unsigned int address, char value) debe
escribir el valor value en la posicion correcta del bloque que corresponde
a address, si esta en el cache, y escribir 1 en *hit si es un hit
y 0 si es un miss.
*/
void write_byte(unsigned int address, char value, unsigned char *hit)
{
unsigned int offset = find_offset(address);
unsigned int set = find_set(address);
unsigned int block = find_block(address);
unsigned int encontradoenvia = find_via(address);
// SI NO LO ENCUENTRA LO VA A LEER


cache.lecturas++;
if (encontradoenvia == EFES)
{

cache.misses++;
encontradoenvia = find_earliest(set);
read_block(block);

// MARCA SALIDA
hit[0] = 0;
}
else
{
// MARCA SALIDA
hit[0] = 1;
}

cache.Bloque[set].vias[encontradoenvia].is_dirty = 1;
cache.Bloque[set].vias[encontradoenvia].datos[offset] = value;

}

/*
```

```
La funcion get miss rate() debe devolver el porcentaje de misses
desde que se inicializo el cache.
read byte() y write byte() solo deben interactuar con la memoria
a traves de las otras primitivas.
*/
char get_miss_rate()
{
if (cache.lecturas == 0) return NO_MISS_RATE;
return (cache.misses * 100) /  cache.lecturas ;
}
```