

# Computer Architecture Lab-6: Cache Miss Simulator

Pradeep Mundlik

November 23, 2023

## 1 Introduction

The objective of this lab exercise is to implement a Cache Miss Simulator, which upon taking configuration and sequence of addresses which to be access, will give Hit or Miss for each address accessed.

The cache configuration is provided in a file `cache.config` in the following format:

```
SIZE_OF_CACHE (number)
BLOCK_SIZE (number)
ASSOCIATIVITY (number)
REPLACEMENT_POLICY (FIFO or LRU or RANDOM)
WRITEBACK_POLICY (WB or WT)
```

The access sequence is provided in another file (`cache.access`) in the following format:

```
Mode: Address
Mode: Address
```

Mode refers to access mode: R: Read or W: Write

Address refers to the memory address that is accessed

## 2 Code Explanation

### 2.1 Input Reading

The code starts by reading configuration parameters from "cache.config" file and reads addresses that to be accessed from "cache.access" file.

### 2.2 Classes

There are different classes implemented for each replacement policy i.e. FIFO, LRU, RANDOM that will take all configuration parameters as arguments. Here, `associativity` is from 1 (Directly Mapped) upto 16 (Max Set Associativity). For `associativity=0` (Fully Associative) case, special classes are implemented i.e. `FIFO_FULLY_ASSOCIATIVE`, `LRU_FULLY_ASSOCIATIVE`, `RANDOM_FULLY_ASSOCIATIVE`

## 2.3 FIFO

Implementation of FIFO policy (`FIFO` and `FIFO_FULLY_ASSOCIATIVE`) is done using queue for each set (single set in case of Fully Associative) in cache. Queue will insert elements (blocks) until its size reaches max number of blocks per set. After that, for every insert it will pop one element out. Vector of queue is used, where  $i^{th}$  queue in vector is for  $i^{th}$  block.

## 2.4 LRU

For LRU policy (`LRU` and `LRU_FULLY_ASSOCIATIVE`), a vector of pair (`vector<pair<uint32_t, int>`) is maintained for each set, where each pair stores Tag and a integer number (initially 0) that will be incremented every time as address will be accessed. During replacement, block with smallest value of the number will represent least recently used block. So, we will replace that pair with new one (with number as max of all present numbers + 1).

## 2.5 RANDOM

For LRU policy (`LRU` and `LRU_FULLY_ASSOCIATIVE`), a vector of tags is maintained and when it will reach its max size i.e. number of blocks per set (or total number of blocks in cache for fully associative) then code will generate random index and will replace tag value at generated index.

## 2.6 WB and WT

Write back is handled same way as reads (assumed write allocate). For write through, when there is miss it is neglected (assumed no write allocate) and hit is handled same way as write back.

### 3 Testing Code

#### 3.1 Test Case 1: Direct Mapped Cache + Write Through

```
cache.config
1 512
2 32
3 1
4 FIFO
5 WT

cache.access
1 W: 0x10000000
2 R: 0x10000000
3 W: 0x20000004
4 W: 0x1000000C
5 R: 0x20000010
6 W: 0x30000014
7 R: 0x10000018
8 W: 0x2000001C
9 R: 0x50000000
10 W: 0x60000004
11 W: 0x30000008

che Miss Simulator/" && g++ main.cpp -o main && "/hom
Address: 0x10000000, Set: 0x0, Tag: 0x80000, Miss
Address: 0x10000000, Set: 0x0, Tag: 0x80000, Miss
Address: 0x20000004, Set: 0x0, Tag: 0x100000, Miss
Address: 0x1000000c, Set: 0x0, Tag: 0x80000, Hit
Address: 0x20000010, Set: 0x0, Tag: 0x100000, Miss
Address: 0x30000014, Set: 0x0, Tag: 0x180000, Miss
Address: 0x10000018, Set: 0x0, Tag: 0x80000, Miss
Address: 0x2000001c, Set: 0x0, Tag: 0x100000, Miss
Address: 0x50000000, Set: 0x0, Tag: 0x280000, Miss
Address: 0x60000004, Set: 0x0, Tag: 0x300000, Miss
Address: 0x30000008, Set: 0x0, Tag: 0x180000, Miss
pradeep@pradeep-IdeaPad-5-15ITL05-Ua:~/Documents/IITH
```

Figure 1: configuration + Sequence + Result

### 3.2 Test Case 2: FIFO + Write Through + Fully Associative

The image shows a terminal window with three sections of text. The top section, titled 'cache.config', lists five parameters: 1. 512, 2. 32, 3. 0, 4. FIFO, and 5. WT. The middle section, titled 'cache.access', shows a sequence of 11 memory accesses, each with a number, a type (W for write, R for read), and a hexadecimal address. The bottom section shows the output of the simulator, displaying the address, set, tag, and hit/miss status for each access.

```
cache.config
1 512
2 32
3 0
4 FIFO
5 WT

cache.access
1 W: 0x10000000
2 R: 0x10000000
3 W: 0x20000004
4 W: 0x1000000C
5 R: 0x20000010
6 W: 0x30000014
7 R: 0x10000018
8 W: 0x2000001C
9 R: 0x50000000
10 W: 0x60000004
11 W: 0x30000008

che Miss Simulator/" && g++ main.cpp -o main && "/hon
Address: 0x10000000, Set: 0x0, Tag: 0x800000, Miss
Address: 0x10000000, Set: 0x0, Tag: 0x800000, Miss
Address: 0x20000004, Set: 0x0, Tag: 0x1000000, Miss
Address: 0x1000000c, Set: 0x0, Tag: 0x800000, Hit
Address: 0x20000010, Set: 0x0, Tag: 0x1000000, Miss
Address: 0x30000014, Set: 0x0, Tag: 0x1800000, Miss
Address: 0x10000018, Set: 0x0, Tag: 0x800000, Hit
Address: 0x2000001c, Set: 0x0, Tag: 0x1000000, Hit
Address: 0x50000000, Set: 0x0, Tag: 0x2800000, Miss
Address: 0x60000004, Set: 0x0, Tag: 0x3000000, Miss
Address: 0x30000008, Set: 0x0, Tag: 0x1800000, Miss
pradeep@pradeep-IdeaPad-5-15ITL05-Ua:~/Documents/IITH
```

Figure 2: configuration + Sequence + Result

### 3.3 Test Case 3: FIFO + Write Through + 4-way Set Associative

The image shows a terminal window with three sections of text. The first section, 'cache.config', lists five parameters: 1. 512, 2. 32, 3. 4, 4. FIFO, and 5. WT. The second section, 'cache.access', lists 11 memory access operations with their type (W for Write, R for Read) and address. The third section shows the results of these operations, including the address, set number, tag, and whether it was a hit or miss.

```

cache.config
1 512
2 32
3 4
4 FIFO
5 WT

cache.access
1 W: 0x10000000
2 R: 0x10000000
3 W: 0x20000004
4 W: 0x1000000C
5 R: 0x20000010
6 W: 0x30000014
7 R: 0x10000018
8 W: 0x2000001C
9 R: 0x50000000
10 W: 0x60000004
11 W: 0x30000008

Cache MISS Simulator / && g++ main.cpp -o main && ./main
Address: 0x10000000, Set: 0x0, Tag: 0x200000, Miss
Address: 0x10000000, Set: 0x0, Tag: 0x200000, Miss
Address: 0x20000004, Set: 0x0, Tag: 0x400000, Miss
Address: 0x1000000c, Set: 0x0, Tag: 0x200000, Hit
Address: 0x20000010, Set: 0x0, Tag: 0x400000, Miss
Address: 0x30000014, Set: 0x0, Tag: 0x600000, Miss
Address: 0x10000018, Set: 0x0, Tag: 0x200000, Hit
Address: 0x2000001c, Set: 0x0, Tag: 0x400000, Hit
Address: 0x50000000, Set: 0x0, Tag: 0xa00000, Miss
Address: 0x60000004, Set: 0x0, Tag: 0xc00000, Miss
Address: 0x30000008, Set: 0x0, Tag: 0x600000, Miss
pradeep@pradeep-IdeaPad-5-15TTL05-Ua: ~/Documents/TTH

```

Figure 3: configuration + Sequence + Result

### 3.4 Test Case 4: LRU + Write Through + 4-way Set Associative

```
main.cpp  cache.c
cache.config
1 512
2 32
3 4
4 LRU
5 WT

cache.access
1 R: 0x10000000
2 W: 0x20000004
3 R: 0x30000008
4 W: 0x1000000C
5 R: 0x20000010
6 W: 0x30000014
7 R: 0x10000018
8 W: 0x2000001C
9 R: 0x30000020
10 W: 0x10000024

Cache Miss Simulator/ && g++ main.cpp -o main && ./main
Address: 0x10000000, Set: 0x0, Tag: 0x200000, Miss
Address: 0x20000004, Set: 0x0, Tag: 0x400000, Miss
Address: 0x30000008, Set: 0x0, Tag: 0x600000, Miss
Address: 0x1000000c, Set: 0x0, Tag: 0x200000, Hit
Address: 0x20000010, Set: 0x0, Tag: 0x400000, Miss
Address: 0x30000014, Set: 0x0, Tag: 0x600000, Hit
Address: 0x10000018, Set: 0x0, Tag: 0x200000, Hit
Address: 0x2000001c, Set: 0x0, Tag: 0x400000, Hit
Address: 0x30000020, Set: 0x1, Tag: 0x600000, Miss
Address: 0x10000024, Set: 0x1, Tag: 0x200000, Miss
pradeep@pradeep-IdeaPad-5-15ITL05-Ua:~/Documents/IITH
```

Figure 4: configuration + Sequence + Result

### 3.5 Test Case 5: LRU + Write Back + 4-way Set Associative

The screenshot displays a cache simulator interface with three main sections: configuration, access sequence, and results.

**Configuration (cache.config):**

Index	Value
1	512
2	32
3	4
4	LRU
5	WB

**Access Sequence (cache.access):**

Index	Operation	Address
1	R	0x10000000
2	W	0x20000004
3	R	0x30000008
4	W	0x1000000C
5	R	0x20000010
6	W	0x30000014
7	R	0x10000018
8	W	0x2000001C
9	R	0x30000020
10	W	0x10000024

**Results:**

Address	Set	Tag	Result
0x10000000	0x0	0x200000	Miss
0x20000004	0x0	0x400000	Miss
0x30000008	0x0	0x600000	Miss
0x1000000c	0x0	0x200000	Hit
0x20000010	0x0	0x400000	Hit
0x30000014	0x0	0x600000	Hit
0x10000018	0x0	0x200000	Hit
0x2000001c	0x0	0x400000	Hit
0x30000020	0x1	0x600000	Miss
0x10000024	0x1	0x200000	Miss

Figure 5: configuration + Sequence + Result