

Documentation

Name – Taral Jain

Roll Number – 2019392

- The code uses in high-level language Java, designed to implement the **Multi-level (L1 & L2) Cache Memory** that allows loading and searching using Direct Mapping, Fully Associative Mapping & K-way Set Associative Mapping.

➤ Assumptions


- 1) All the inputs have to be given **Manually** by the user.
- 2) In different mappings, the inputs are:
 - **Direct Mapping:** N (Total number of words in memory)
W (Number of words in a Block/Block Size)
CL (Number of Cache Lines)
 - **Fully Associative:** N (Total number of words in memory)
W (Number of words in a Block/ Block Size)
CL (Number of Cache Lines)
Operation (READ, WRITE, DISPLAY CACHE)
Address (To READ or WRITE)
 - **K-way set Associative:** N (Total number of words in memory)
W (Number of words in a Block/ Block Size)
CL (Number of Cache Lines)
K (Number of cache lines in a Set)
Operation (READ, WRITE, DISPLAY CACHE)
Address (To READ or WRITE)
- 3) The number of bits in Address is assumed to be exactly equal to " **$\log_2(N)$** "
- 4) Main Memory is present **NOT** present in any of the Mapping.
- 5)- In case of miss- "CACHE MISS"
 - In case of replacement - Address by which the block will be replaced

6) For Block replacement, **LRU** (Least Recently Used) policy is used


➤ Searching Algorithm & Replacement Algorithm in Detail

- Whenever the CPU refers to a word, firstly it is searched in the cache. If the reference is found in the cache, it is a **CACHE HIT** else it is a miss. Initially the cache will suffer from **CACHE MISS**.
- A miss in lower level (L1) causes access to the upper level (L2) in the memory hierarchy.
- In case of miss from L1 and L2, the required word is fetched from the main memory.
- As L1 is possessing rich temporal locality, the word which is often used are likely to be used again, which means that **Latest word should remain in L1**. Access to L2 occurs when a reference suffers a miss on L1 thus L2 is having weaker temporal locality in contrast to L1. Latest pages refer to the pages which are referred frequently while old pages refer to the pages which are referred less frequently. Thus, when a miss occurs on L1 and if the cache is saturated the victim chosen from L1 should be the old pages. **Whenever a page is removed from L1, it will be placed on L2.**
- When L2 gets full, we use LRU for L2 and replace accordingly.


➤ Explanation of Code

 **Direct Mapping** – The cache contains multiple sets with a single cache line per set. Based on the address of the memory block, it can only occupy a single cache line.

- The line is determined by the index bits derived from the address of the memory block for Cache L1
- The memory block is placed in the line identified and the tag is stored in the tag field associated with the line
- If the cache line is previously occupied, then the new data replaces the memory block in the Cache L1 and old one is transferred to Cache L2
- To **search** a particular block, we compare tags in Cache L1. If the tag matches, then there is a cache hit and the word is displayed on screen. Otherwise, cache miss occurs in Cache L1 and then Cache L2 is being searched. If cache hit then display the result, otherwise memory block is fetched from Main Memory (**NOT DONE IN CODE AS NO MAIN MEMORY**)

 **Fully Associative Mapping** – The cache contains a single cache set with multiple cache lines. A memory block can occupy any of the cache lines.

- The cache line is selected by Tag of the line. If the Tag is -1, the new memory block can be placed in the cache line, else it has to be placed in another cache line with Tag -1.
- If the cache is full then a block is **removed** using **LRU policy** and new memory block is then placed in that cache line in Cache L1 and removed one is transferred to Cache L2.
- To **search**, we compare Tag of the memory address with Tag associated with all the cache lines using loop traversal. If match occurs, the block is present in the Cache L1 and is a **Cache Hit**, we print the Word on screen. Else, **Cache Miss** and then search in Cache L2. If present then print else block has to be loaded from Main memory.

 **K-way Set Associative Mapping** — It lies in mediocrity of Direct Mapping and Fully Associative Mapping.

- The cache is fragmented as 'CL/K' sets and each set contain 'K' Cache Lines.
- A memory block is first mapped onto a set and then placed into **any** cache line of the set if and only if the particular set has **empty Cache Line(s)**.
- The set number is determined by some particular bits of address.
- The block is placed in empty cache line in the determined set, and Tag is associated with the Cache Line. If all Cache Lines in the set are occupied, then We use **LRU policy** to remove some block and then store new block into that removed Cache Line of Cache L1 and removed one is transferred to Cache L2.
- To **search**, we compare Tag with Tag of all Cache Lines in determined set. If the Tag matches in Cache L1, **Cache Hit** occurs and we print the particular Word in that Cache Line. Else, **Cache Miss** occurs and we search in Cache L2. If it matches, then print else the block is loaded from Main Memory.

➤ Sample I/O of Code

Direct Mapping

```
Enter the value of N (MEMORY SIZE: EXPONENT OF 2)
5
Enter the value of Cache Lines (EXPONENT OF 2)
2
Enter the value of W (EXPONENT OF 2)
1
Enter the Operation (R)ead or (W)rite or (D)isplay
W
Enter the Address To Read/Write
11000
Enter The Data To Write At 11000
32
Enter the Operation (R)ead or (W)rite or (D)isplay
W
Enter the Address To Read/Write
10000
Enter The Data To Write At 10000
64
Tag of Block for replacement using LRU is 11 in Cache Line 0 in Cache L1
Enter the Operation (R)ead or (W)rite or (D)isplay
D
LEVEL-1 CACHE MEMORY
LEVEL-2 CACHE MEMORY
0 0 TAG = -1
0 0 TAG = -1
0 0 TAG = -1
0 0 TAG = -1
32 1 TAG = 1
0 0 TAG = -1
0 0 TAG = -1
0 0 TAG = -1
```

Full Associative Mapping

```

Enter the value of N (MEMORY SIZE: EXPONENT OF 2)
5
Enter the value of Cache Lines (EXPONENT OF 2)
1
Enter the value of W (EXPONENT OF 2)
3
Enter the Operation (R)ead or (W)rite or (D)isplay
W
Enter the Address To Read/Write
11000
Enter The Data To Write At 11000
32
Enter the Operation (R)ead or (W)rite or (D)isplay
W
Enter the Address To Read/Write
10000
Enter The Data To Write At 10000
64
Enter the Operation (R)ead or (W)rite or (D)isplay
W
Enter the Address To Read/Write
01000
Enter The Data To Write At 01000
128
Tag of Block for replacement using LRU is 11 in SET 0 in Cache L1
Enter the Operation (R)ead or (W)rite or (D)isplay
D
LEVEL-1 CACHE MEMORY
128 1 1 1 1 1 1 TAG = 1
64 1 1 1 1 1 1 TAG = 2

LEVEL-2 CACHE MEMORY
32 1 1 1 1 1 1 TAG = 3
0 0 0 0 0 0 0 TAG = -1
0 0 0 0 0 0 0 TAG = -1
0 0 0 0 0 0 0 TAG = -1
Enter the Operation (R)ead or (W)rite or (D)isplay
R
Enter the Address To Read/Write
01000
CACHE HIT
DATA at 01000 is 128 and is found in Cache_L1

```

K-Way Set Associative Mapping

```

Enter the value of N (MEMORY SIZE: EXPONENT OF 2)
6
Enter the value of Cache Lines (EXPONENT OF 2)
3
Enter the value of K
2
Enter the value of W (EXPONENT OF 2)
1
Enter the Operation (R)ead or (W)rite or (D)isplay
W
Enter the Address To Read/Write
011010
Enter The Data To Write At 011010
32
Enter the Operation (R)ead or (W)rite or (D)isplay
W
Enter the Address To Read/Write
010010
Enter The Data To Write At 010010
64
Enter the Operation (R)ead or (W)rite or (D)isplay
W
Enter the Address To Read/Write
000010
Enter The Data To Write At 000010
128
Tag of Block for replacement using LRU is 11 in SET 1 in Cache L1
Enter the Operation (R)ead or (W)rite or (D)isplay
D
LEVEL-1 CACHE MEMORY
0 0 - SET = 0, TAG = -1
0 0 - SET = 0, TAG = -1
128 1 - SET = 1, TAG = 0
64 1 - SET = 1, TAG = 2
0 0 - SET = 2, TAG = -1
0 0 - SET = 2, TAG = -1
0 0 - SET = 3, TAG = -1
0 0 - SET = 3, TAG = -1

LEVEL-2 CACHE MEMORY
0 0 - SET = 0, TAG = -1
0 0 - SET = 0, TAG = -1
0 0 - SET = 1, TAG = -1
0 0 - SET = 1, TAG = -1
0 0 - SET = 2, TAG = -1
0 0 - SET = 2, TAG = -1
0 0 - SET = 3, TAG = -1
0 0 - SET = 3, TAG = -1
0 0 - SET = 4, TAG = -1
0 0 - SET = 4, TAG = -1
32 1 - SET = 5, TAG = 1
0 0 - SET = 5, TAG = -1
0 0 - SET = 6, TAG = -1
0 0 - SET = 6, TAG = -1
0 0 - SET = 7, TAG = -1
0 0 - SET = 7, TAG = -1

```