**Batch: B-1 Roll No.: 16010122104**

**Experiment / assignment / tutorial No. 8**

|  |
| --- |
| **TITLE:** Implementation of FIFO Page Replacement Algorithm |

**AIM:** To study and implement concept of various mapping techniques designed for

cache memory.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment: (Mention CO/CO’s attained here)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”,

Fifth Edition, TataMcGraw-Hill.

2. Dr. M. Usha, T. S. Srikanth, “Computer System Architecture and Organization”,

First Edition, Wiley-India.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

1. Cache memory: The cache is a smaller, faster memory which stores copies of the data

from the most frequently used main memory locations. As long as most memory

accesses are cached memory locations, the average latency of memory accesses will be

closer to the cache latency than to the latency of main memory.

1. Hit Ratio: You want to increase as much as possible the likelihood of the cache

containing the memory addresses that the processor wants.

**Hit Ratio= No. of hits/ (No. of hits + No. of misses)**

There are only fewer cache lines than the main memory blocks, an algorithm is needed

for mapping main memory blocks into cache lines. Further a means is needed for

determining which main memory block currently occupies in a cache line. The choice

of cache function dictates how the cache is organized. Three techniques can be used.

1. Direct mapping.

2. Associative mapping.

3. Set Associative mapping.

**Direct Mapped Cache:** The direct mapped cache is the simplest form of cache and the

easiest to check for a hit. Since there is only one possible place that any memory

location can be cached, there is nothing to search; the line either contains the memory

information we are looking for, or it doesn’t.

Unfortunately, the direct mapped cache also has the worst performance, because again

there is only one place that any address can be stored. Let’s look again at our 512 KB

level 2 cache and 64 MB of system memory. As you recall this cache has 16,384 lines

(assuming 32-byte cache lines) and so each one is shared by 4,096 memory addresses.

In the absolute worst case, imagine that the processor needs 2 different addresses (call

them X and Y) that both map to the same cache line, in alternating sequence (X, Y, X,

Y). This could happen in a small loop if you were unlucky. The processor will load X

from memory and store it in cache. Then it will look in the cache for Y, but Y uses the

same cache line as X, so it won’t be there. So Y is loaded from memory, and stored in

the cache for future use. But then the processor requests X, and looks in the cache only

to find Y. This conflict repeats over and over. The net result is that the hit ratio here is

0%. This is a worst case scenario, but in general the performance is worst for this type

of mapping.

**Fully Associative Cache:** The fully associative cache has the best hit ratio because any

line in the cache can hold any address that needs to be cached. This means the problem

seen in the direct mapped cache disappears, because there is no dedicated single line

that an address must use.However (you knew it was coming), this cache suffers from problems involving searching the cache. If a given address can be stored in any of

16,384 lines, how do you know where it is? Even with specialized hardware to do the

searching, a performance penalty is incurred. And this penalty occurs for all accesses to

memory, whether a cache hit occurs or not, because it is part of searching the cache to

determine a hit. In addition, more logic must be added to determine which of the

various lines to use when a new entry must be added (usually some form of a &quot;least

recently used&quot; algorithm is employed to decide which cache line to use next). All this

overhead adds cost, complexity and execution time.

**Set Associative Cache (To be filled in by students)**

A set-associative cache is a compromise between a direct-mapped cache and a fully associative cache. It provides some degree of associativity while still maintaining the structure and simplicity of a direct-mapped cache. In a set-associative cache, the cache is divided into several sets, and each set contains multiple cache lines (also known as slots). Each set functions as a small direct-mapped cache. The primary purpose of set-associative caches is to reduce cache conflicts compared to a purely direct-mapped cache while maintaining a balance between hardware complexity and cache performance.

**Direct Mapping Implementation:**

The mapping is expressed as

i=j modulo m

i=cache line number

j= main memory block number

m= number of lines in the cache

● Address length = (s+w) bits

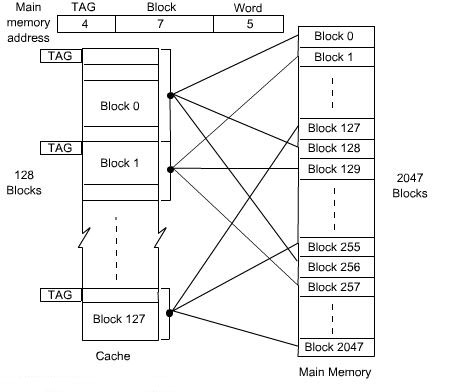
● Number of addressable units = 2 s+w words or bytes

● Block size = line size = 2 w words or bytes

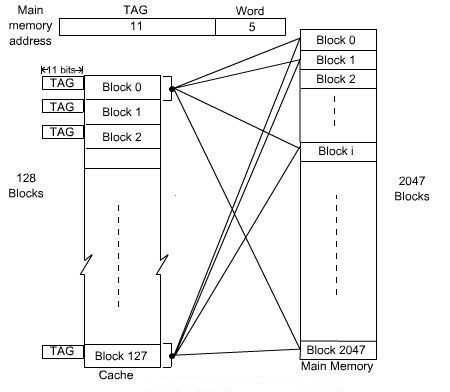
● Number of blocks in main memory = 2 s+w / 2 w = 2 s

● Number of lines in cache = m = 2 r

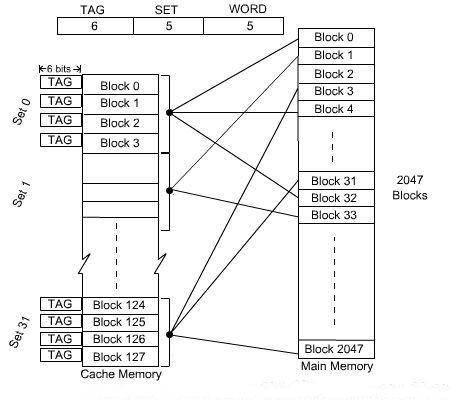
● Size of tag = (s-r) tags



**Associative Mapping Implementation: (To be filled in by students)**



**Set Associative Mapping Implementation:**



**Post Lab Descriptive Questions:-**

1. **For a direct mapped cache, a main memory is viewed as consisting of 3 fields.**

**List and define 3 fields.**

**Answer:**

For a direct-mapped cache, the main memory is viewed as consisting of three fields:

* **Block Offset**: This field identifies the specific data within a cache block (also known as a cache line). The size of this field is determined by the block size or cache line size. It specifies the number of bytes within a block and is used to select a specific byte or word from the block. In binary terms, the block offset is the least significant bits of the memory address.
* **Index**: The index field is used to determine which cache slot (or line) in the cache should store the data from a specific block in main memory. It is calculated based on the number of cache lines (slots) in the cache. The index specifies which slot is used to store data associated with a particular main memory block.
* **Tag**: The tag field contains the remaining bits of the memory address and is used to identify which block in main memory a particular cache slot (line) is storing. It is essentially a "tag" that helps match the data in the cache to the correct location in main memory. When a memory access is made, the tag in the cache is compared to the tag generated from the memory address. If they match, it indicates a cache hit; if they don't match, it's a cache miss, and the data must be retrieved from main memory.

These fields work together to determine where a particular piece of data should be stored in the cache and how it should be retrieved.

1. **What is the general relationship among access time, memory cost, and capacity?**

**Answer:**

1. **Access Time: Access time refers to the time it takes to retrieve or store data from or into a memory system. In general, as we move from the lower levels of the memory hierarchy (e.g., registers and caches) to higher levels (e.g., main memory and secondary storage), the access time tends to increase. Faster memory technologies, such as registers and cache, provide shorter access times compared to slower technologies, like main memory or disk storage.**
2. **Memory Cost: Memory cost refers to the cost associated with a specific level of memory in the hierarchy. Typically, faster memory technologies, like registers and cache, are more expensive per unit of storage compared to slower technologies. This means that faster memory is more costly, and the cost decreases as you move to slower, larger memory types. The cost of memory is an important factor in determining the overall cost of a computing system.**
3. **Capacity: Capacity refers to the amount of data that a specific memory level can store. In general, as we move from the lower levels of the memory hierarchy to higher levels, the capacity tends to increase. Registers and cache have limited capacities but provide very fast access times. Main memory has a larger capacity but slightly slower access times. Secondary storage devices, like hard drives, have even greater capacity but longer access times.**

* Faster memory technologies with shorter access times tend to be more expensive and have limited capacity.
* Slower memory technologies with longer access times tend to be less expensive and have larger capacity.

**Conclusion: We** implemented concept of various mapping techniques designed for

cache memory.

**Date: 11/10/2023**