Batch: C-2 Roll No.: 16010122267

Experiment / assignment / tutorial No.8

TITLE : Implementation of Cache Mapping Techniques.

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:

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.

2. 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.

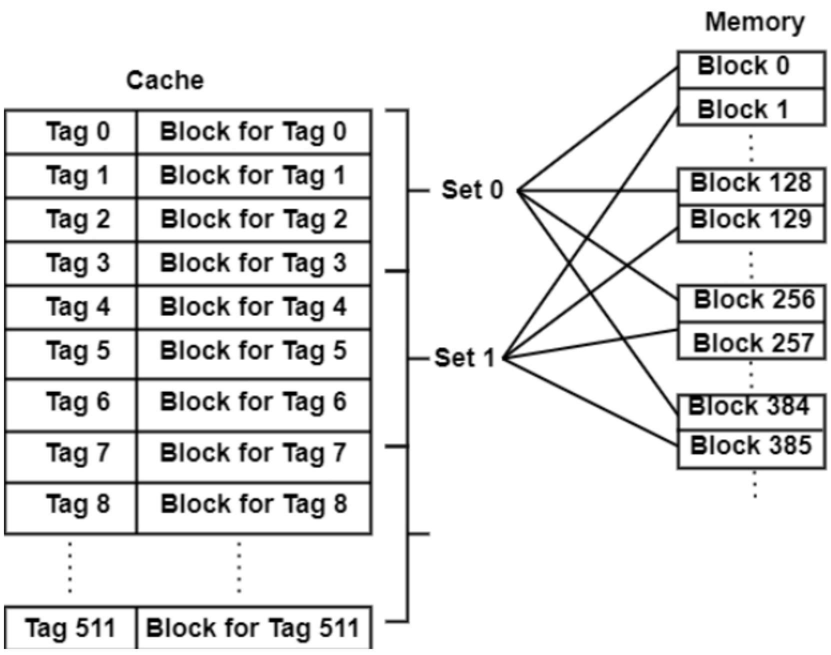
Direct Mapped Cache: The direct mapped cache is the simplest form of cache and the easiest to check for a hit. Since there is only one possible place that any memory location can be cached, there is nothing to search; the line either contains the memory information we are looking for, or it doesn't. 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 "least recently used" 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)

Set associative mapping combines direct mapping with fully associative mapping by arrangement lines of a cache into sets. The sets are persistent using a direct mapping scheme. However, the lines within each set are treated as a small fully associative cache where any block that can save in the set can be stored to any line inside the set.

Set Associative Mapping of Main Memory to Cache



A set-associative cache that includes k lines per set is known as a k way set-associative cache. Because the mapping approach uses the memory address only like direct mapping does, the number of lines included in a set should be similar to an integer power of two, for example, two, four, eight, sixteen, etc.

Direct Mapping Implementation:

The mapping is expressed as

i=j modulo m

i=cache line number

j= main memory block number

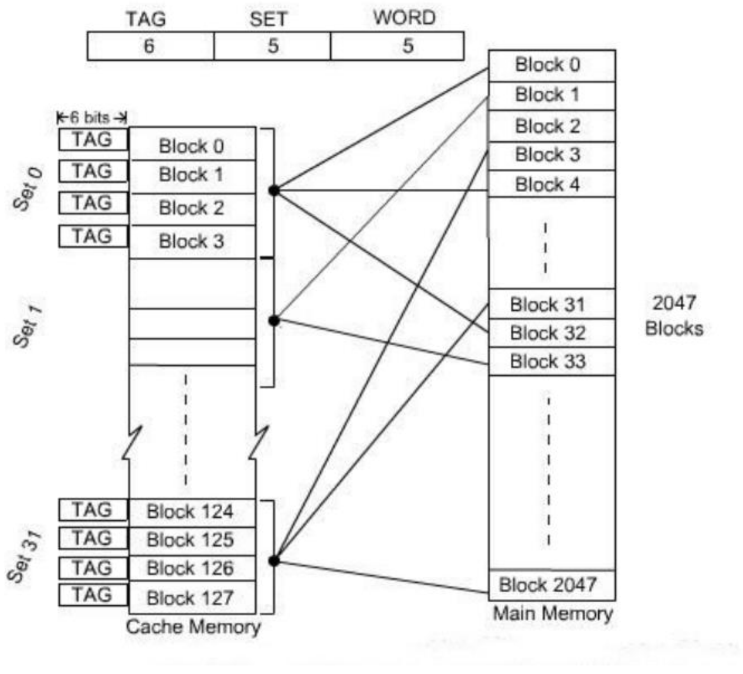
m= number of lines in the cache

- 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)**

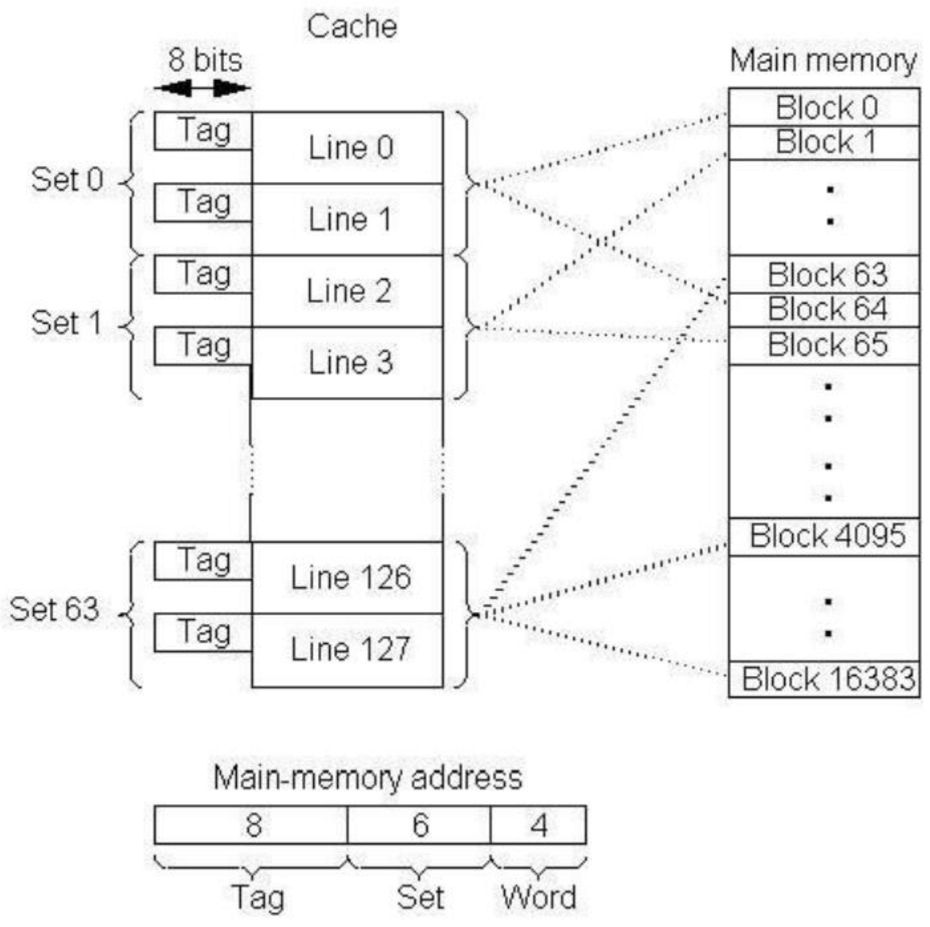
We have a main memory consisting of 16 kilobytes, with each block comprising 4 bytes, and a fully associative cache with a capacity of 256 bytes and a block size of 4 bytes. To uniquely represent a memory address in this 16kB main memory, we require a minimum of 14 bits.

Since each cache block has a size of 4 bytes, the cache is divided into a total of 64 sets or cache lines.



Set Associative Mapping Implementation:

- Cache lines are organized into sets, each consisting of k lines.
- However, a specific main memory block is constrained to map to only one set within the cache.
- Within that particular set, the memory block can be placed in any available cache line.
- The formula to determine which cache set a particular main memory block maps to is given by: Cache set number = (Main Memory Block Address) % (Number of sets in Cache).

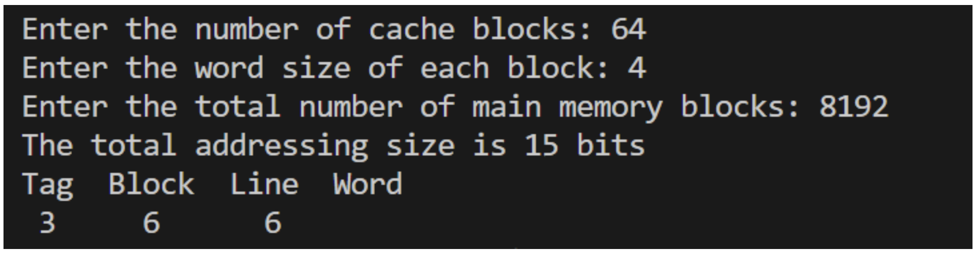


Code:

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    int m;

    do
    {
        cout << "Enter the number of your choice:" << endl;
        cout << "1: Direct Mapping" << endl;
        cout << "2: Two way Set Associative Mapping" << endl;
        cout << "3: Fully Associative Mapping" << endl;
        cout << "4: Exit" << endl;
        cin >> m;

        if (m == 1)
        {
            int cacheBlocks, wordSize, totalMainMemBlocks, d, e = 0,
totalAddSize = 0, g = 0, h = 0, i = 0, j = 0, k;
            cout << "Enter the number of cache blocks: ";
            cin >> cacheBlocks;
            cout << "Enter the word size of each block: ";
            cin >> wordSize;
            cout << "Enter the total number of main memory blocks: ";
            cin >> totalMainMemBlocks;
            d = totalMainMemBlocks * wordSize;

            while (e != d)
            {
                totalAddSize = totalAddSize + 1;
                e = pow(2, totalAddSize);
            }
            cout << "The total addressing size is " << totalAddSize << "
bits" << endl;

            while (g != cacheBlocks)
            {
                h = h + 1;
                g = pow(2, h);
            }
            while (i != cacheBlocks)
            {
                j = j + 1;
                i = pow(2, j);
            }
            k = totalAddSize - h - j;
            cout << "Tag  Block  Line  Word" << endl;
            cout << " " << k << "     " << h << "       " << j << endl;
        }
        else if (m == 2)
        {
            int cacheBlocks, wordSize, totalMainMemBlocks, d, e = 0,
totalAddSize = 0, g = 0, i = 0, j = 0, k;
            cout << "Enter the number of cache blocks: ";
            cin >> cacheBlocks;
            cout << "Enter the word size of each block: ";
            cin >> wordSize;
            cout << "Enter the total number of main memory blocks: ";
            cin >> totalMainMemBlocks;
            d = totalMainMemBlocks * wordSize;

            while (e != d)
            {
                totalAddSize = totalAddSize + 1;
                e = pow(2, totalAddSize);
            }
            cout << "The total addressing size is " << totalAddSize << "
bits" << endl;

            while (i != wordSize)
            {
                j = j + 1;
                i = pow(2, j);
            }
            k = totalAddSize - j;
            cout << "Tag  Word" << endl;
            cout << " " << k << "     " << j << endl;
        }
        else if (m == 3)
        {
            int cacheBlocks, wordSize, totalMainMemBlocks, d, e = 0,
totalAddSize = 0, g = 0, h = 0, i = 0, j = 0, k, l, n;
            cout << "Enter the number of cache blocks: ";
            cin >> cacheBlocks;
            cout << "Enter the word size of each block: ";
            cin >> wordSize;
            cout << "Enter the total number of main memory blocks: ";
            cin >> totalMainMemBlocks;
            cout << "Enter the set number: ";
            cin >> l;
            d = totalMainMemBlocks * wordSize;

            while (e != d)
            {
                totalAddSize = totalAddSize + 1;
                e = pow(2, totalAddSize);
            }
            cout << "The total addressing size is " << totalAddSize << "
bits" << endl;
            n = cacheBlocks / l;

            while (g != n)
            {
                h = h + 1;
                g = pow(2, h);
            }

            while (i != wordSize)
            {
                j = j + 1;
                i = pow(2, j);
            }
            k = totalAddSize - h - j;
            cout << "Tag  Set  Word" << endl;
            cout << " " << k << "     " << h << "    " << j << endl;
        }
        else
        {
            cout << "Invalid input" << endl;
        }
    } while (m != 4);
    return 0;
}
```

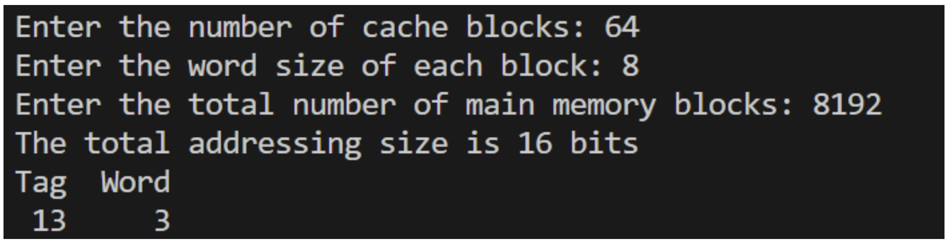
Output:

Direct Mapping:



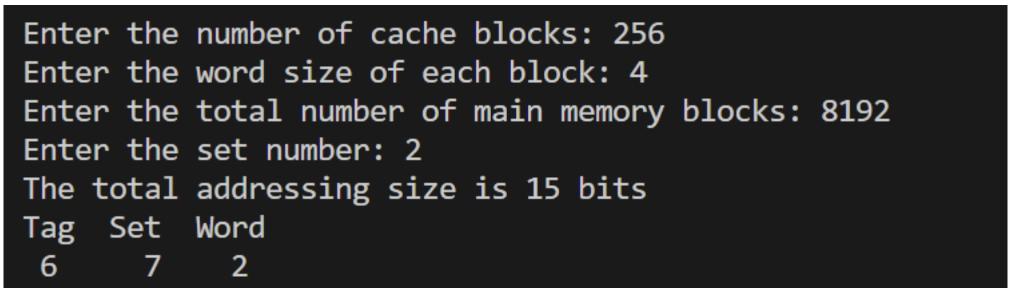
```
Enter the number of cache blocks: 64
Enter the word size of each block: 4
Enter the total number of main memory blocks: 8192
The total addressing size is 15 bits
Tag  Block  Line  Word
 3     6      6
```

Two Way Set Associative Mapping:



```
Enter the number of cache blocks: 64
Enter the word size of each block: 8
Enter the total number of main memory blocks: 8192
The total addressing size is 16 bits
Tag  Word
 13     3
```

Fully Associative Mapping:



```
Enter the number of cache blocks: 256
Enter the word size of each block: 4
Enter the total number of main memory blocks: 8192
Enter the set number: 2
The total addressing size is 15 bits
Tag  Set  Word
 6     7    2
```

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.

Ans:

1. **Block Field (Block Offset):**

- Definition: This field determines the size of each cache block and is used to identify the specific word or byte within a cache block.
- Purpose: It helps to select the desired word or byte within a cache block and allows for efficient data retrieval within a block. The size of this field is determined by the block size.

2. **Index Field (Cache Index):**

- Definition: The index field is used to identify the cache line (or slot) to which a specific main memory block can be mapped. It is typically created by taking a subset of the memory address bits.

- Purpose: It helps determine the cache line where a specific main memory block should be stored, facilitating quick lookup and retrieval. The size of this field is determined by the number of cache lines in the cache.

3. **Tag Field:**

- Definition: The tag field contains the remaining bits of the memory address after the block offset and index bits have been extracted. It uniquely identifies the main memory block stored in a particular cache line.
- Purpose: It allows the cache to distinguish between different main memory blocks that could potentially be mapped to the same cache line. This field is crucial for ensuring data integrity and preventing data conflicts.

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

Ans:

1. **Access Time (Latency):** Access time represents the time it takes to fetch or store data in a memory system. Faster access times mean quicker data retrieval, which is vital for responsive and high-performance computing. High-speed memory technologies like SRAM offer blazing-fast access times but are typically more expensive and provide lower storage capacity.
2. **Memory Cost:** Memory cost refers to the monetary expense associated with acquiring and maintaining memory. Memory technologies vary in cost, with high-performance options like DRAM and solid-state drives (SSDs) typically being more expensive per unit of storage compared to traditional hard disk drives (HDDs) and other slower alternatives.
3. **Capacity:** Capacity denotes the total amount of data a memory system can store. It is measured in bytes, and higher capacity allows for the storage of more data, applications, and files. However, as you increase capacity, memory costs typically rise.

Here's a more detailed perspective:

- **Access Time vs. Capacity:** There's often an inverse relationship. Faster memory (e.g., SRAM, DRAM) offers quick access times but lower capacity. Slower memory (e.g., HDDs) provides greater capacity but longer access times. Faster access usually means less storage space, and vice versa.
- **Access Time vs. Memory Cost:** Faster memory tech is generally pricier. High-speed memory components come with higher price tags due to their performance advantages. Low access times are associated with higher memory costs.
- **Capacity vs. Memory Cost:** Generally, the more storage capacity you need, the more it will cost. Expanding storage capacity, such as choosing a larger SSD or HDD, typically means paying a higher price.

Finding the right balance among these factors is essential. It depends on the specific needs of your computer system. High-performance systems prioritize low access times, even if it means higher costs. In contrast, budget-conscious applications may opt for larger storage options, accepting longer access times in exchange for affordability. Memory hierarchies, like combining high-speed cache with larger but slower main memory and storage, aim to strike a balance among these trade-offs to provide efficient computing solutions.

Conclusion

Through this experiment, we learnt various types of mapping techniques and implemented them through a program.

Date: _____________