Batch: E2	**Roll No.:** 16010123325
Experiment / assignment / tutorial No._____	

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)

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)

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, m;
    cout << "Number of pages in memory: ";
    cin >> n;
    cout << "Number of lines in cache: ";
    cin >> m;

    vector<int> cac(m, -1);

    int t = 1;
    while (t == 1) {
        cout << "1. The Memory line: \n";
        cout << "2. Exit \n";
        cout << "Enter your choice: ";
        int ch;
        cin >> ch;

        switch(ch) {
            case 1:
                cout << "Enter line number: " << '\n';
                int x;
                cin >> x;
                if (x > n || x < 0) {
                    cout << "Wrong Input! \n";
                }
                else {
                    int ver = x % m;
                    if (cac[ver] == -1) {
                        cout << "Adding to Cache Line: " << ver << '\n';
                        cac[ver] = x;
                    }
                    else {
                        cout << "Cache Line: " << ver << " can't be added.
\n";
                        cout << "Cache Line already occupied by " << cac[ver]
<< '\n';
                    }
                }
                break;
```

```
            case 2:
                t = 0;
                break;

            default:
                cout << "Invalid Choice! \n";
                break;
        }
    }
    return 0;
}
```

```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\COA> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\COA\Programs\" ; if ($?) { g
++ Cache-Mapping.cpp -o Cache-Mapping } ; if ($?) { .\Cache-Mapping }
Number of pages in memory: 16
Number of lines in cache: 4
1. The Memory line:
2. Exit
Enter your choice: 1
Enter line number:
2
Adding to Cache Line: 2
1. The Memory line:
2. Exit
Enter your choice: 1
Enter line number:
2
Cache Line: 2 can't be added.
Cache Line already occupied by 2
1. The Memory line:
2. Exit
Enter your choice: 1
Enter line number:
3
Adding to Cache Line: 3
1. The Memory line:
2. Exit
Enter your choice: 1
Enter line number:
4
Adding to Cache Line: 0
1. The Memory line:
2. Exit
Enter your choice: 1
Enter line number:
2
Cache Line: 2 can't be added.
Cache Line already occupied by 2
1. The Memory line:
2. Exit
Enter your choice: 2
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\COA\Programs>
```

Set Associative Mapping Implementation:

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, m;
    cout << "Enter number of pages: ";
    cin >> n;
    cout << "Enter number of cache lines: ";
    cin >> m;

    vector<int> cac1(m, -1);
    vector<int> cac2(m, -1);
    vector<int> cac3(m, -1);

    int t = 1;
    while (t == 1) {
        cout << "1. Memory Line: \n";
        cout << "2. Exit \n";
        cout << "Enter your choice: ";
        int ch;
        cin >> ch;

        switch(ch) {
            case 1: {
                cout << "Enter line number: ";
                int x;
                cin >> x;
                if (x >= n || x < 0) cout << "Wrong Input! \n";
                else {
                    int ver = x % m;
                    if (cac1[ver] == -1 || cac2[ver] == -1 || cac3[ver] == -1)
{
                        if (cac1[ver] == -1) {
                            cout << "Adding to 1st Cache line: " << ver <<
'\n';
                            cac1[ver] = x;
                        }
                        else if (cac2[ver] == -1) {
                            cout << "Adding to 2nd Cache line: " << ver <<
'\n';
                            cac2[ver] = x;
                        }
                        else {
                            cout << "Adding to 3rd Cache line: " << ver <<
'\n';
                            cac3[ver] = x;
```

```
                        }
                    }
                    else {
                        cout << "Cache Line: " << ver << " can't be added.
\n";
                        cout << "1st Cache line occupied by: " << cac1[ver] <<
'\n';
                        cout << "2nd Cache line occupied by: " << cac2[ver] <<
'\n';
                        cout << "3rd Cache line occupied by: " << cac3[ver] <<
'\n';
                    }
                }
                break;
            }
            case 2:
                t = 0;
                break;

            default:
                cout << "Invalid Choice! \n";
                break;
        }
    }
    return 0;
}
```

```
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\COA> cd "c:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\COA\Programs\" ; if ($?) { g
++ Cache-Mapping.cpp -o Cache-Mapping } ; if ($?) { .\Cache-Mapping }
Enter number of pages: 32
Enter number of cache lines: 4
1. Memory Line:
2. Exit
Enter your choice: 1
Enter line number: 0
Adding to 1st Cache line: 0
1. Memory Line:
2. Exit
Enter your choice: 1
Enter line number: 4
Adding to 2nd Cache line: 0
1. Memory Line:
2. Exit
Enter your choice: 0
Invalid Choice!
1. Memory Line:
2. Exit
Enter your choice: 1
Enter line number: 8
Adding to 3rd Cache line: 0
1. Memory Line:
2. Exit
Enter your choice: 1
Enter line number: 4
Cache Line: 0 can't be added.
1st Cache line occupied by: 0
2nd Cache line occupied by: 4
3rd Cache line occupied by: 8
1. Memory Line:
2. Exit
Enter your choice: 2
PS C:\Users\Shrey\OneDrive\Desktop\KJSCE\SEM-3\COA\Programs>
```

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.

In a direct-mapped cache, the main memory address is divided into three fields:

1. **Tag**: Identifies which block is in the cache.
2. **Index**: Selects the specific cache line.
3. **Block Offset**: Locates the specific byte within the block.

These fields help efficiently map memory blocks to cache lines.

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

The general relationship is: **Faster access time** results in **higher memory cost** and typically **smaller capacity** (e.g., cache memory). Conversely, **slower access time** comes with **lower cost** and **larger capacity** (e.g., main memory or disk storage). This trade-off balances speed, cost, and storage size in memory hierarchies.