Batch: B-2 Roll No.: 16010122151

**Experiment / assignment / tutorial No._____**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

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

Set-associative cache is a specific type of cache memory that occurs in RAM and processors. It divides the cache into between two to eight different sets or areas. Data is stored in them all, but the cache distributes it to each set-in sequence, rather than randomly. In a set associative cache, there are a fixed number of locations (called a set) that a given address may be stored in. The number of locations in each set is the associative of the cache. Since a set associative cache has more sets and areas mean more performance, but also higher cost when it comes to manufacturing and implementing that memory.

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

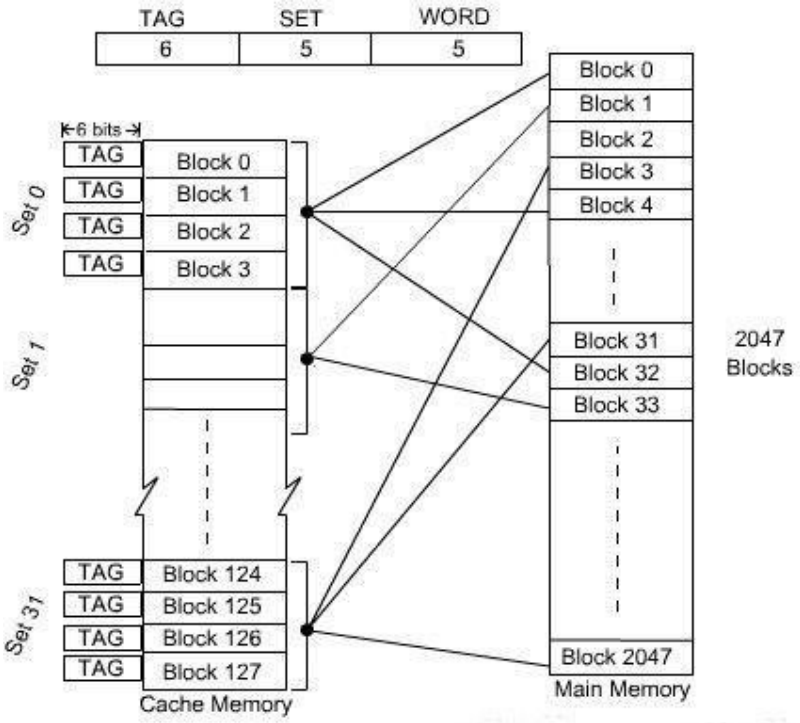
- Address length = (s+w) bits
- Number of addressable units = $2^{s+w}$ words or bytes ● Block size = line size = $2^{w}$ words or bytes
- Number of blocks in main memory = $2^{s+w} / 2^{w} = 2^{s}$
- Number of lines in cache = m = $2^{r}$
- Size of tag = (s-r) tags

Output

Associative Mapping Implementation:

Sample implementation

- Consider a main memory of 16 kilobytes, which is organized as 4-byte blocks, and a fully associative cache of 256 bytes and a block size of 4 bytes.
- Since, the main memory is 16kB, we need a minimum of 14 bits to uniquely represent a memory address.
- Since each cache block is of size 4 bytes, the total number of sets in the cache is 256/4, which equals 64 sets or cache lines.

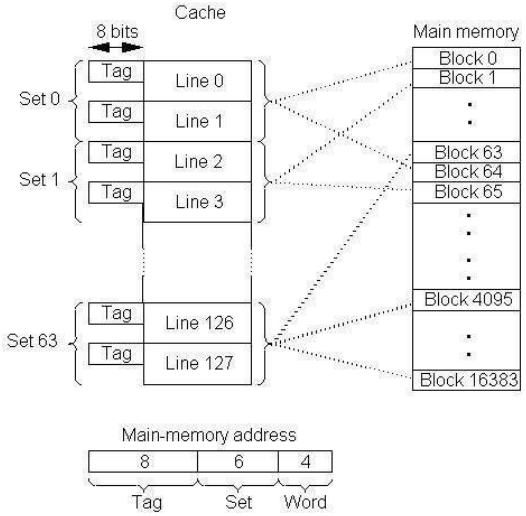


Set Associative Mapping Implementation:

In Set associative mapping,

- Cache lines are grouped into sets where each set contains k number of lines
- However, a particular block of main memory can map to only one set of the cache
- But, within that set, the memory block can map to any freely available cache line
- The formula for the set of the cache to which a particular block of the main memory can map is given by-

Cache set number = (Main Memory Block Address) % (Number of sets in Cache)




Source Code

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

int main()
{
   int memory_lines, blocks;
   cout<<"Enter number of main memory lines:";
   cin>>memory_lines;
```

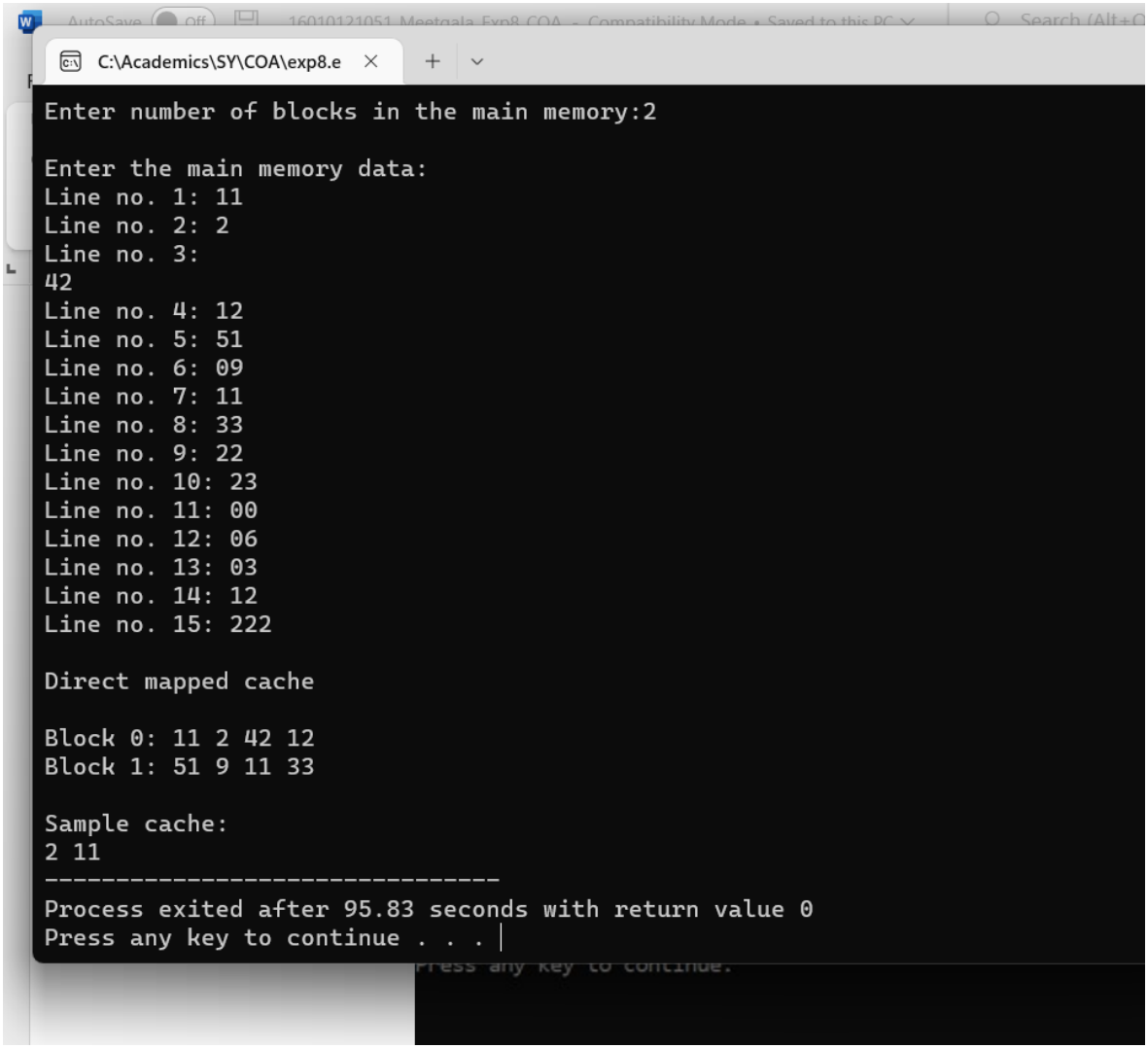
```
    cout<<"Enter number of blocks in the main memory:";
    cin>>blocks;
    int bmemory[blocks][4];
    int mmemory[memory_lines];

   cout<<"\nEnter the main memory data:"<<endl;

   for(int i = 0;i<memory_lines;i++)
   {
      cout<<"Line no. "<<i+1<<": ";
      cin>>mmemory[i];
   }
   int k = 0;

   for(int i = 0;i<blocks;i++)
   for(int j = 0;j<4;j++)
   bmemory[i][j] = mmemory[k++];
            cout<<"\nDirect mapped cache\n";
        for(int i = 0;i<blocks;i++)
   {
               cout<<endl<<"Block "<<i<<": ";
   for(int j = 0;j<4;j++)
      cout<<bmemory[i][j]<<" ";
   }
   cout<<"\n\nSample cache:\n";
   for(int i = 0;i<blocks;i++)
   {
      int random = rand()%5;
      cout<<bmemory[i][random]<<" ";
   }
}
```

Output




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.

- One field on the direct-mapped cache memory identifies a unique word or byte within a block of main memory
- The remaining two fields specify one of the blocks of main memory
- These two fields are a line field, which identifies one of the lines of the cache, and a tag field, which identifies one of the blocks that can fit into that line

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

- Faster access time is directly proportional to cost per bit, it means that as the access time speed increases the cost per bit also increases
- Memory capacity is inversely proportional to cost per bit, it means that as memory capacity increases, the cost per bit decreases
- Memory capacity is inversely proportional to access time, it means that as memory capacity increases the access time speed decreases

Conclusion: Therefore, with the help of the experiment the various mapping techniques were understood. Set associative and direct mapping techniques were implemented by writing programs to demonstrate them.

Date: 27/11/22 Signature of faculty in-charge