Batch: B1 Roll No.: 1711072

Experiment / assignment / tutorial No. 8

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

Signature of the Staff In-charge with date

TITLE: Implementation of Cache Mapping Techniques.

AIM: To study and implement concept of various mapping techniques designed for cache memory.

Expected OUTCOME of Experiment:
CO 4-Learn and evaluate memory organization and cache structure

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.

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

<u>Implementation Details (in Java):</u>

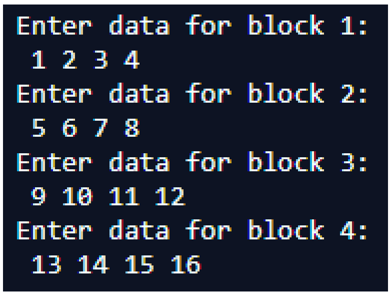
```
import java.util.*;
class Main {
  public static void main(String[] args) {
    int choice;
    int[] page1=new int[4],page2=new int[4],page3=new
int[4],page4=new int[4];
    int mat[][]=new int[4][4];
    Scanner sc=new Scanner(System.in);
    Random rand=new Random();
    System.out.println("Enter data for block 1: ");
    for(int i=0;i<4;i++)
      page1[i]=mat[0][i]=sc.nextInt();
    System.out.println("Enter data for block 2: ");
    for(int i=0;i<4;i++)
      page2[i]=mat[1][i]=sc.nextInt();
    System.out.println("Enter data for block 3: ");
    for(int i=0;i<4;i++)
      page3[i]=mat[2][i]=sc.nextInt();
    System.out.println("Enter data for block 4: ");
    for(int i=0;i<4;i++)
      page4[i]=mat[3][i]=sc.nextInt();
    System.out.printf("%10s %10s %10s %10s", "Block 1","Block 2", "Block
3","Block 4");

    System.out.println();
    for(int i=0;i<4;i++){
      System.out.format("%10s %10s %10s %10s", mat[i][0], mat[i][1],
mat[i][2], mat[i][3]);
      System.out.println();
    }
    do{
        System.out.print("\n1. Direct Mapping\n2. Two way set
associative mapping\n3. Exit\nEnter your choice: ");
        choice=sc.nextInt();
        switch(choice){
          case 1:
          System.out.println("Cache memory: ");
          System.out.printf("%d %d %d %d",
mat[rand.nextInt(4)][0], mat[rand.nextInt(4)][1],
mat[rand.nextInt(4)][2], mat[rand.nextInt(4)][3]);
          break;
          case 2:
          System.out.println("Cache memory 1: ");
          System.out.printf("%d %d %d %d",
mat[rand.nextInt(4)][0], mat[rand.nextInt(4)][1],
mat[rand.nextInt(4)][2], mat[rand.nextInt(4)][3]);
          System.out.println("\nCache memory 2: ");
          System.out.printf("%d %d %d %d",
mat[rand.nextInt(10)%4][0], mat[rand.nextInt(10)%4][1],
mat[rand.nextInt(10)%4][2], mat[rand.nextInt(10)%4][3]);
          break;
          case 3: System.exit(1);
          default:
          System.out.println("Enter valid choice.");
          }
        }while(choice!=5);
    }
```

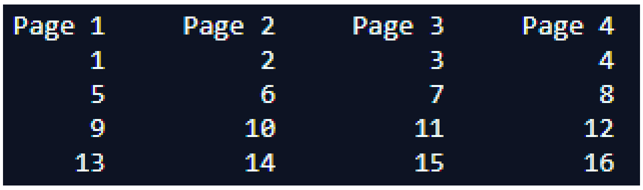
```
  }
```

For verification, my code is available on: https://repl.it/@ARGHYADEEPDAS/CacheMapping

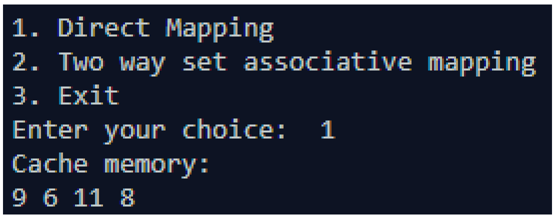
Output Screen:



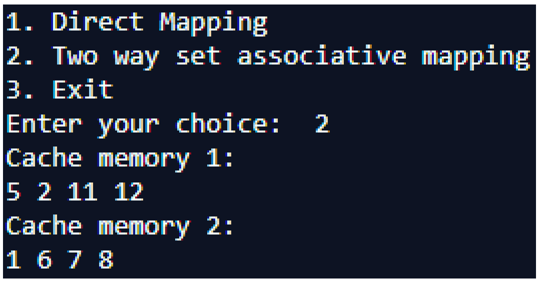
```
Enter data for block 1:
 1 2 3 4
Enter data for block 2:
 5 6 7 8
Enter data for block 3:
 9 10 11 12
Enter data for block 4:
 13 14 15 16
```



```
Page 1      Page 2      Page 3      Page 4
     1           2           3           4
     5           6           7           8
     9          10          11          12
    13          14          15          16
```



```
1. Direct Mapping
2. Two way set associative mapping
3. Exit
Enter your choice:  1
Cache memory:
9 6 11 8
```



```
1. Direct Mapping
2. Two way set associative mapping
3. Exit
Enter your choice:  2
Cache memory 1:
5 2 11 12
Cache memory 2:
1 6 7 8
```

Post Lab Descriptive Questions (Add questions from examination point view)

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

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

Ans. One field 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.

Ans. This is the relationship among Access time, Memory cost per bit and the storage capacity:

Faster access tune implies greater cost per bit and greater storage capacity.

Conversely, slower access time implies smaller cost per bit and greater capacity.

This means:

1. If computer can access something in so fast then its cost per bit will be also greater and capacity of storage also be greater.
2. If computer can/need access something is slow, then its cost per bit will be smaller and the capacity remain constant and will be greater.

Conclusion: The program ran successfully as we were able to simulate both direct cache mapping technique and 2-way associative cache mapping technique.

Date: 10/10/2018 **Signature of faculty in-charge**