NAME: Meith Navlakha
SAP: 60004190068
POA Practical

EXPERIMENT 4

AIM: Implement Sequential Memory Organization

THEORY:

Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations.

Thus, in order to access the data from the secondary memory, load it into the L1 and L2 cache and make it available for the processor, a linking or mapping technique is needed for accessing the data and passing it through the memory hierarchy. This is explained by **mapping functions**.

Mapping Functions:

The mapping functions are used to map a particular block of main memory to a particular block of memory of cache. This mapping is used to transfer the block from the main memory to cache memory. Three different mapping functions are available:

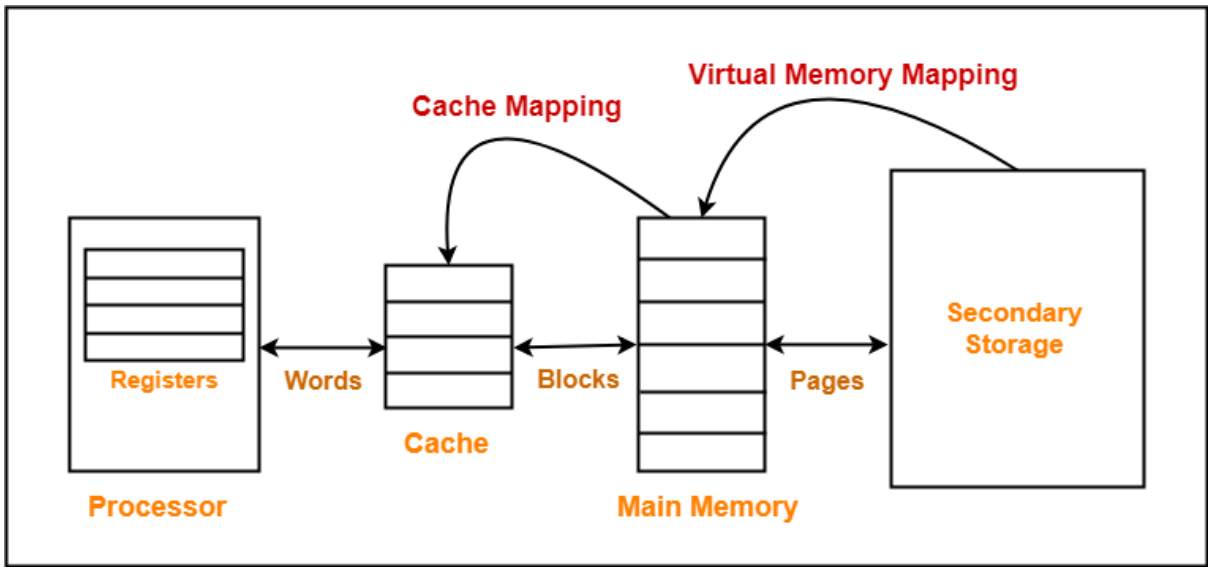
1**) Direct Mapping:** In this technique, block k of main memory maps into k modulo m of the cache, where m is the total number of blocks in cache. A particular block of memory can be brought to a particular block of cache memory. So, it is not flexible.

2) **Associative Mapping:** In this mapping function any block of main memory can potentially reside in any cache block position. In this case, the main memory is divided into two groups, low-order bits identifies the location within a block and high-order bits identifies the block. This is much more flexible mapping method.

3) **Block Set-Associative Mapping:** In this method, blocks of cache are grouped into sets, and the mapping allows a block of main memory to reside in any block of a specific set. From the flexibility point of view, it is in between the other two methods. This method reduces the searching overhead, because the search is restricted to number of sets, instead of blocks.

All the above 3 mapping techniques require **Page Replacement Policy** when the cache is full. Th property of locality of reference gives some clue to design a good replacement policy. Some types:

- LRU
- FIFO



CODE:

```
import java.util.*;
import java.util.Arrays;

public class SeqMem {
  public static int MAX_PAGES = 64;
  public static int MAX_BLOCKS = 64;
  public static int MAX_WORDS = 32;
  public static int[][][] mm = new int[MAX_PAGES][4][2];
  public static int[][] L2 = new int[MAX_BLOCKS][2];
  public static int[] L1 = new int[MAX_WORDS];
  public static int processor = -1;
  public static int TL1 = 20;
  public static int TL2 = 60;
  public static int TMM = 120;
  public static int in_index_L1 = -1;
  public static int in_index_L2 = -1;
```

```
public static int[] searchMM(int elem) {

  int p = (elem / (mm[0].length * mm[0][0].length) % MAX_PAGES);
  int b = (elem / 2) % 4;
  int w = elem % 2;

  // System.out.print(" "+Arrays.toString((mm[p][b]))+" ") ;

  System.out.print(" W" + elem + " Found in Main Memory --> PAGE " + p);
  System.out.print("\n TOTAL TIME TAKEN : " + (TL1 + TL2 + TMM) + " ns\n");

  return mm[p][b];

}

public static int searchL2(int elem) {

  if (in_index_L2 == -1) {
    // Empty
    int[] b = searchMM(elem);
    L2[++in_index_L2] = b;

    for (int i = 0; i < b.length; i++) {
      if (b[i] == elem) {
        return b[i];
      }
    }

  } else {

    for (int i = 0; i <= in_index_L2; i++) {
      for (int k = 0; k < L2[0].length; k++) {
        if (L2[i][k] == elem) {
          System.out.print(" W" + elem + " Found in L2 --> BLOCK " + i);
          System.out.print("\n TOTAL TIME TAKEN : " + (TL1 + TL2) + " ns\n");
          return elem;
        }
      }
    }
```

```
    int[] b = searchMM(elem);
    in_index_L2++;
    L2[in_index_L2 % MAX_BLOCKS] = b;

    for (int i = 0; i < b.length; i++) {
      if (b[i] == elem) {
        return b[i];
      }
    }

  }

  return 0;
}

public static int searchL1(int elem) {

  if (in_index_L1 == -1) {
    // Empty
    int w = searchL2(elem);
    in_index_L1++;
    L1[in_index_L1] = w;
    return w;

  } else {

    for (int i = 0; i <= in_index_L1; i++) {

      if (L1[i] == elem) {
        System.out.print(" W" + elem + " Found in L1 WORD --> " + i);
        System.out.print("\n TOTAL TIME TAKEN : " + (TL1) + " ns\n");

        // break;
        return elem;
      }
    }

    int w = searchL2(elem);
    in_index_L1++;
```

```
    L1[in_index_L1 % MAX_WORDS] = w;
    return w;

  }

}

public static void SeqMemSearch(int elem) {

  if (processor != elem) {
    int w = searchL1(elem);
    processor = w;

  } else {
    System.out.print("W" + elem + " Found in Processor.");
    System.out.print("\n TIME TAKEN : 0ns\n");

  }
}

public static void main(String args[]) {

  Scanner sc = new Scanner(System.in);

  // Data in MM
  int data = 0;
  for (int i = 0; i < MAX_PAGES; i++) {
    for (int j = 0; j < mm[0].length; j++) {
      for (int k = 0; k < mm[0][0].length; k++) {

        mm[i][j][k] = data;
        data++;
      }
    }

  }

  int elem = 0;
  while (elem != -1) {
    System.out.print("\n Enter the word to Search : ");
```

```
      elem = sc.nextInt();
      if (elem == -1) {
        break;
      } else {
        SeqMemSearch(elem);
      }
    }
  }
}
```

OUTPUT:

```
C:\Users\meith\Desktop\SEM 5\POA>java SeqMem

 Enter the word to Search : 5
 W5 Found in Main Memory --> PAGE 0
 TOTAL TIME TAKEN : 200 ns

 Enter the word to Search : 9
 W9 Found in Main Memory --> PAGE 1
 TOTAL TIME TAKEN : 200 ns

 Enter the word to Search : 37
 W37 Found in Main Memory --> PAGE 4
 TOTAL TIME TAKEN : 200 ns

 Enter the word to Search : 98
 W98 Found in Main Memory --> PAGE 12
 TOTAL TIME TAKEN : 200 ns

 Enter the word to Search : 5
 W5 Found in L1 WORD --> 0
 TOTAL TIME TAKEN : 20 ns

 Enter the word to Search : 4
 W4 Found in L2 --> BLOCK 0
 TOTAL TIME TAKEN : 80 ns

 Enter the word to Search : 4
W4 Found in Processor.
 TIME TAKEN : 0ns

 Enter the word to Search : 18
 W18 Found in Main Memory --> PAGE 2
 TOTAL TIME TAKEN : 200 ns

 Enter the word to Search : 17
 W17 Found in Main Memory --> PAGE 2
 TOTAL TIME TAKEN : 200 ns

 Enter the word to Search : 16
 W16 Found in L2 --> BLOCK 5
 TOTAL TIME TAKEN : 80 ns
```

CONCLUSION: In this experiment, I implemented sequential memory organization having Main Memory of size 2048 bytes, L2 having capacity of 128 words, L1 having capacity of 32 words and processor which can store 1 word, where 1word is equal to 8 bytes. User requests for a word, the system will find it in the processor, if not there then finds in the L1 cache, if not there then finds in the L2 cache and if not there finally into main memory. While traversing back the corresponding block and the word will be stored in L2 and L1 respectively (all the higher levels of memory) so that it is present in the cache incase of future request for the same word. FIFO is used as a page replacement policy. The final output also calculates the total time taken to fetch the word from the memory storage hierarchy.