Batch: B1 **Roll No.: 1711072**

Experiment / assignment / tutorial No. 7
Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Implementation of FIFO Page Replacement Algorithm

AIM: The FIFO algorithm uses the principle that the block in the set which has been in for the longest time will be replaced

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. William Stallings, "Computer Organization and Architecture: Designing for Performance", Eighth Edition, Pearson.
3. Dr. M. Usha, T. S. Srikanth, "Computer System Architecture and Organization", First Edition, Wiley-India.

Pre Lab/ Prior Concepts:

T he FIFO algorithm uses the principle that the block in the set which has been in the block for the longest time is replaced. FIFO is easily implemented as a round robin or criteria buffer technique. The data structure used for implementation is a queue. Assume that the number of cache pages is three. Let the request to this cache is shown alongside.

Algorithm:

1. A hit is said to be occurred when a memory location requested is already in the cache.
2. When cache is not full, the number of blocks is added.
3. When cache is full, the block is replaced which was added first

Design Steps:

1. Start
2. Get input as memory block to be added to cache
3. Consider an element of the array
4. If cache is not full, add element to the cache array
5. If cache is full, check if element is already present
6. If it is hit is incremented
7. If not, element is added to cache removing first element (which is in first).
8. Repeat step 3 to 7 for remaining elements
9. Display the cache at very instance of step 8
10. Print hit ratio
11. End.

Example:

```
Enter stack size:  3
Enter number of elements you want to enter:  6

Enter number 1:  1

Current stack: 1 0 0
Enter number 2:  2

Current stack: 1 2 0
Enter number 3:  5

Current stack: 1 2 5
Enter number 4:  3

Current stack: 3 2 5
Enter number 5:  5

Stack unchanged: 3 2 5
Enter number 6:  2

Stack unchanged: 3 2 5
Hit: 2
```

Implementation Details (in Java):

```
import java.util.*;
class Main {
  public static void main(String[] args) {
    Scanner sc=new Scanner(System.in);
    System.out.print("Enter stack size: ");
    int len=sc.nextInt();
    int j=-1, elem, hit=0;
    int arr[]=new int[len];
    System.out.print("Enter number of elements you want to enter: ");
    int n=sc.nextInt();
    for(int i=0;i<n; i++){
      System.out.print("\nEnter number "+(i+1)+": ");
       elem=sc.nextInt();
      boolean result=search(elem, arr);
      if(result){
        System.out.print("\nStack unchanged: ");
        print(arr);
        hit++;
      }
      else{
        arr[(j+1)%len]=elem;
        System.out.print("\nCurrent stack: ");
        print(arr);
        j++;
      }
      if(i==n-1)
        System.out.println("\nHit: "+hit);
    }
  }
static boolean search(int elem, int[] arr){
    for(int i=0;i<arr.length;i++){
      if(arr[i]==elem){
        return true;
      }
    }
    return false;
  }
static void print(int[] arr){
  for(int i=0;i<arr.length;i++){
    System.out.print(arr[i]+" ");
    }
  }
}
```

For verification, my code can be found at:

https://repl.it/@ARGHYADEEPDAS/FIFOPageReplacementAlgorithm

Output Screen:

```
Enter stack size:  3
Enter number of elements you want to enter:  6

Enter number 1:  1

Current stack: 1 0 0
Enter number 2:  2

Current stack: 1 2 0
Enter number 3:  5

Current stack: 1 2 5
Enter number 4:  3

Current stack: 3 2 5
Enter number 5:  5

Stack unchanged: 3 2 5
Enter number 6:  2

Stack unchanged: 3 2 5
Hit: 2
```

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

1. What is meant by memory interleaving?

2. Explain Paging Concept?

1. In computing, **interleaved memory** is a design made to compensate for the relatively slow speed of dynamic random-access memory (DRAM) or core memory, by spreading memory addresses evenly across memory banks. That way, contiguous memory reads and writes are using each memory bank in turn, resulting in higher memory throughputs due to reduced waiting for memory banks to become ready for desired operations. With interleaved memory, memory addresses are allocated to each memory bank in turn. For example, in an interleaved system with two memory banks (assuming word-addressable memory), if logical address 32 belongs to bank 0, then logical address 33 would belong to bank 1, logical address 34 would belong to bank 0, and so on. An interleaved memory is said to be *n-way interleaved* when there are n banks and memory location i resides in bank $i \bmod n$. Interleaved memory results in contiguous reads (which are common both in multimedia and execution of programs) and contiguous

writes (which are used frequently when filling storage or communication buffers) actually using each memory bank in turn, instead of using the same one repeatedly. This results in significantly higher memory throughput as each bank has a minimum waiting time between reads and writes.

2. In computer operating systems, **paging** is a memory management scheme by which a computer stores and retrieves data from secondary storage for use in main memory. In this scheme, the operating system retrieves data from secondary storage in same-size blocks called *pages*. Paging is an important part of virtual memory implementations in modern operating systems, using secondary storage to let programs exceed the size of available physical memory.

   For simplicity, main memory is called "RAM" (an acronym of "random-access memory") and secondary storage is called "disk" (a shorthand for "hard disk drive"), but the concepts do not depend on whether these terms apply literally to a specific computer system.

   Paging scheme permits the physical address space of a process to be non – contiguous.

- Logical Address or Virtual Address (represented in bits): An address generated by the CPU
- Logical Address Space or Virtual Address Space(represented in words or bytes): The set of all logical addresses generated by a program
- Physical Address (represented in bits): An address actually available on memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

Example:

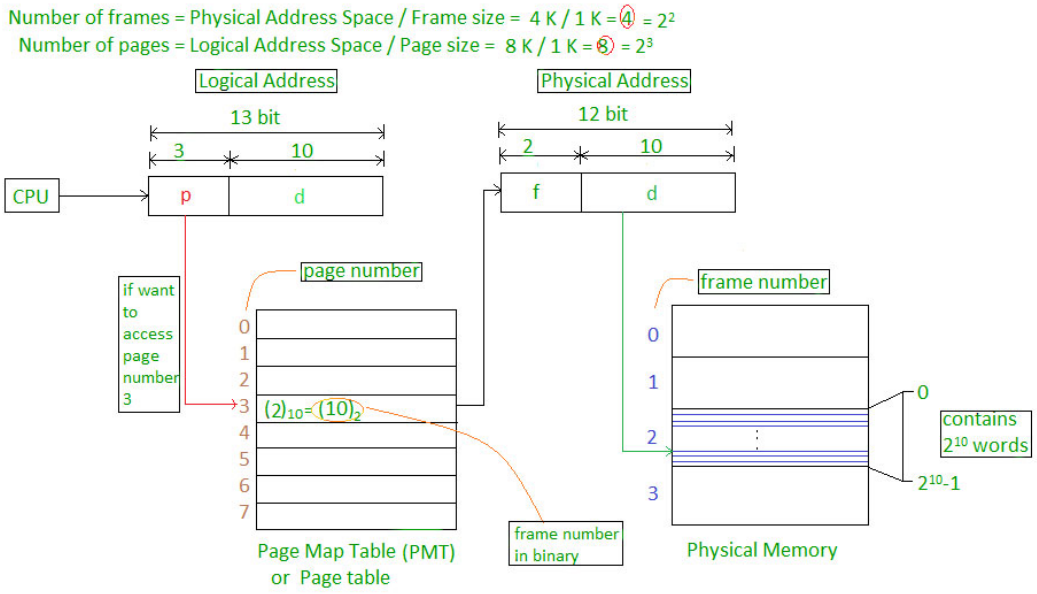
- If Logical Address = 31 bit, then Logical Address Space = 2^{31} words = 2 G words (1 G = 2^{30})
- If Logical Address Space = 128 M words = $2^7 * 2^{20}$ words, then Logical Address = $\log_2 2^{27}$ = 27 bits
- If Physical Address = 22 bit, then Physical Address Space = 2^{22} words = 4 M words (1 M = 2^{20})
- If Physical Address Space = 16 M words = $2^4 * 2^{20}$ words, then Physical Address = $\log_2 2^{24}$ = 24 bits

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as paging technique.

- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
- The Logical address Space is also splitted into fixed-size blocks, called **pages**.
- Page Size = Frame Size

Let us consider an example:

- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)



Address generated by CPU is divided into

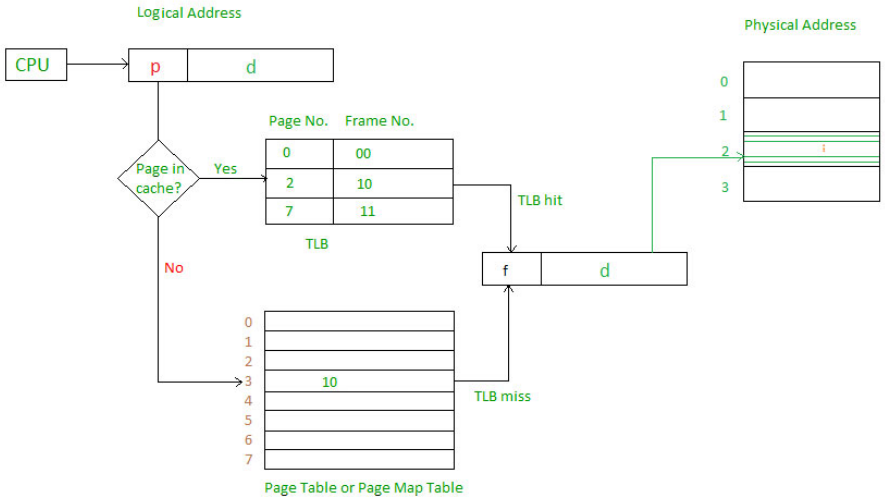
- **Page number (p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page offset (d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into

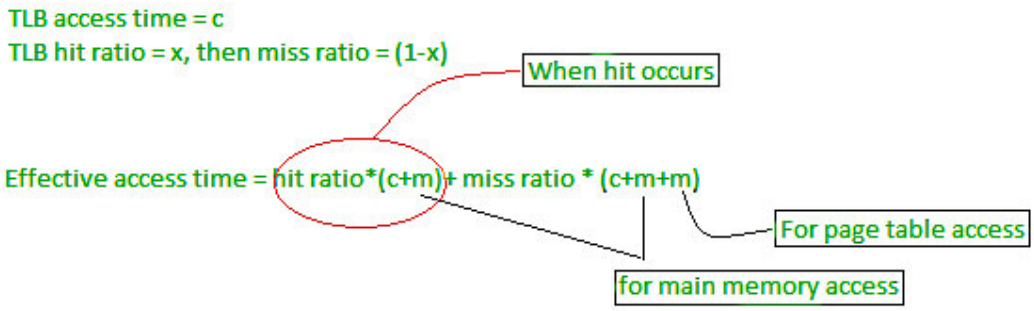
- **Frame number (f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
- **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

The hardware implementation of page table can be done by using dedicated registers. But the usage of register for the page table is satisfactory only if page table is small. If page table contain large number of entries then we can use TLB(translation Look-aside buffer), a special, small, fast look up hardware cache.

- The TLB is associative, high speed memory.
- Each entry in TLB consists of two parts: a tag and a value.
- When this memory is used, then an item is compared with all tags simultaneously.If the item is found, then corresponding value is returned.



Main memory access time = m
If page table are kept in main memory,
Effective access time = m(for page table) + m(for particular page in page table)



Conclusion: The program was successfully implemented in Java as we were able to simulate the FIFO Page Replacement Algorithm for various test cases.

Date: 19/09/2018 **Signature of faculty in-charge**