**Batch: C3 Roll No.: 16010123217**

**Experiment / assignment / tutorial :- 7**

|  |
| --- |
| **TITLE :** Implementation ofFIFO 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: (Mention CO/CO’s attained here)**

C03: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:**

The 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.

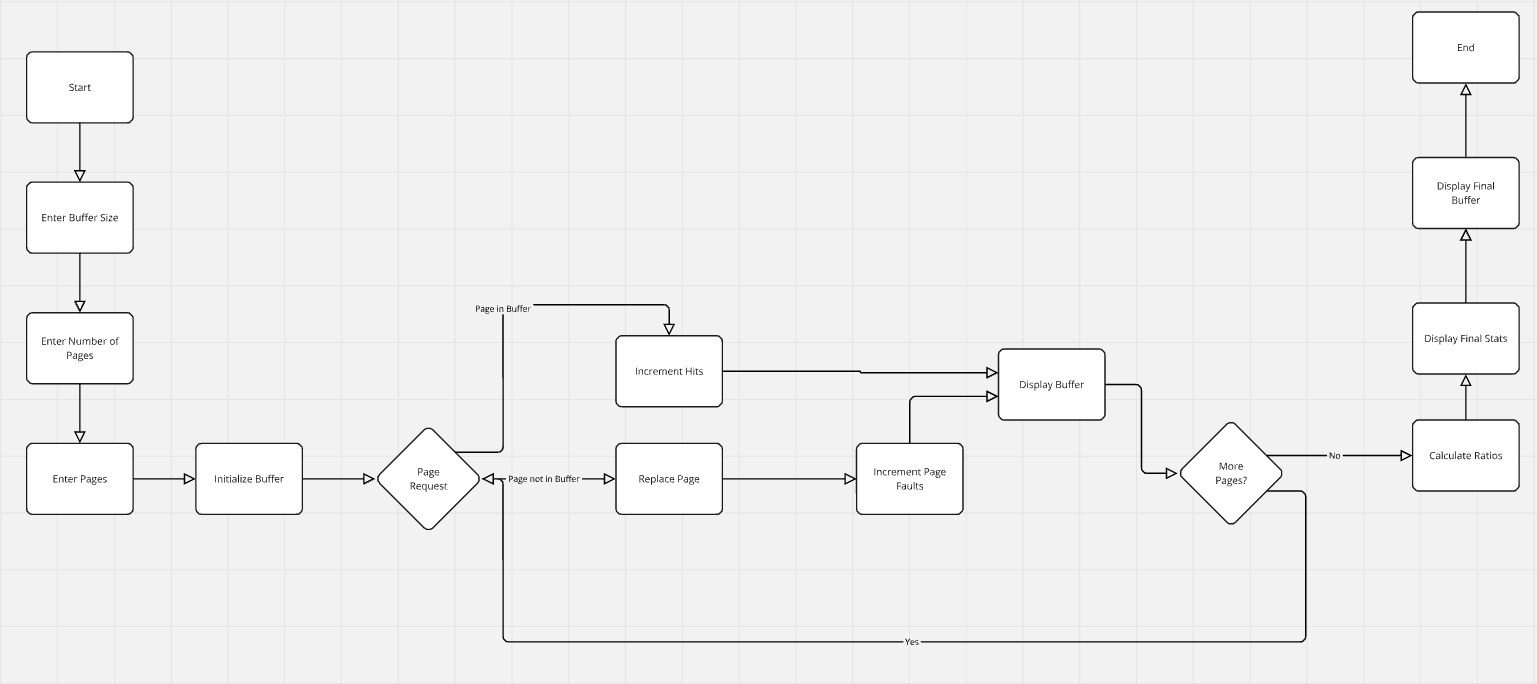
**Algorithm:**

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:**



 **Input Handling**:

* It asks the user for the buffer size (number of frames) and the number of pages.
* It collects the page numbers the user wants to access.

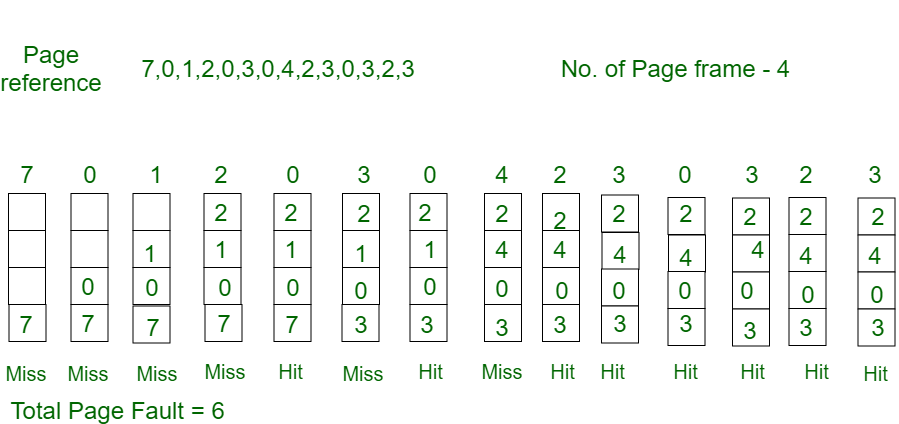
 **Page Replacement Logic**:

* Initializes a buffer (array) to store pages currently in memory, initially filled with -1 to denote empty frames.
* For each page access:
  + Checks if the page is already in the buffer (hit).
  + If not, it adds the new page into the buffer (page fault).
  + Uses a circular approach to replace the oldest page when the buffer is full.

 **Output**:

* Displays the state of the memory buffer after each page access.
* Finally prints total page faults, hits, hit ratio, and fault ratio.

**Example:**



**CODE:-**

#include <stdio.h>

int main() {

    int i, j, pos = 0, pageFaults = 0, hits = 0,arrsize, pcount, flag;

    printf("Enter Buffer size: ");

    scanf("%d", &arrsize);

    printf("Enter number of pages: ");

    scanf("%d", &pcount);

    int arr[arrsize], pages[pcount];

    printf("Enter the pages: ");

    for (i = 0; i < pcount; i++) {

        scanf("%d", &pages[i]);

    }

    for (i = 0; i < arrsize; i++) {

        arr[i] = -1;

    }

    for (i = 0; i < pcount; i++) {

        flag =0;

        for (j = 0; j < arrsize; j++) {

            if (arr[j] == pages[i]) {

                hits++;

                flag = 1;

                break;

            }

        }

        if (flag == 0) {

            arr[pos] = pages[i];

            pageFaults++;

        }

        printf("\nMemory buffer after accessing page %d: ", pages[i]);

        for (j = 0; j < arrsize; j++) {

            if (arr[j] != -1) {

                printf("%d ", arr[j]);

            } else {

                printf("- ");

            }

        }

        pos = (pos + 1) % arrsize;

    }

    float hitRatio = (float)hits / pcount;

    float faultRatio = (float)pageFaults / pcount;

    printf("\n\nTotal Page Faults: %d", pageFaults);

    printf("\nTotal Hits: %d", hits);

    printf("\nHit Ratio: %.2f", hitRatio);

    printf("\nFault Ratio: %.2f", faultRatio);

    printf("\n\nFinal memory buffer: ");

    for (i = 0; i < arrsize; i++) {

        if (arr[i] != -1) {

            printf("%d ", arr[i]);

        } else {

            printf("- ");

        }

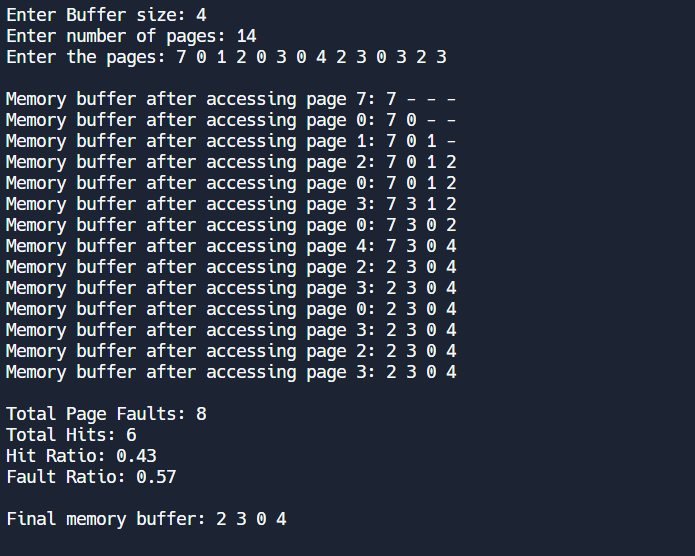
    }

    printf("\n");

    return 0;

}

**SCREENSHOT:-**

****

**Post Lab Descriptive Questions**

**1. What is meant by memory interleaving?**

**Memory Interleaving** is a technique used to enhance the performance of memory access in computer systems. It involves dividing memory into multiple banks, allowing simultaneous access to these banks. This parallelism helps reduce wait times and increase data throughput.

### Key Aspects:

1. **Parallel Access**: Multiple memory banks can be accessed at once, improving speed.
2. **Access Patterns**: Data is arranged so that sequential addresses are distributed across different banks.
3. **Types**: Includes block interleaving and round-robin interleaving.
4. **Benefits**: Increases effective memory bandwidth and reduces latency, leading to better overall system performance.

**2. Explain Paging Concept?**

**Paging** is a memory management scheme that eliminates the need for contiguous allocation of physical memory and thus avoids fragmentation. It allows the operating system to retrieve data from secondary storage in fixed-size blocks called **pages**.

### Key Concepts:

1. **Pages and Frames**:

* **Pages**: The logical memory is divided into fixed-size blocks called pages (typically ranging from 4 KB to 64 KB).
* **Frames**: The physical memory (RAM) is divided into blocks of the same size called frames. Each page maps to a frame in physical memory.

1. **Page Table**:

* Each process has a page table that maintains the mapping between the pages in virtual memory and the frames in physical memory. This table keeps track of where each page is stored.

1. **Logical vs. Physical Address Space**:

* The logical address space is what a process uses (composed of pages), while the physical address space is where data is actually stored in memory (composed of frames).

1. **Address Translation**:

* When a process accesses a memory location, the logical address is translated into a physical address using the page table. The logical address consists of a page number and an offset within that page.

1. **No External Fragmentation**:

* Because pages are of a fixed size, paging eliminates external fragmentation. However, it can lead to internal fragmentation if the last page of a process is not fully utilized.

1. **Demand Paging**:

* This allows pages to be loaded into memory only when they are needed, which conserves memory and speeds up the initial loading time of a process.

1. **Swapping**:

* When physical memory is full, pages can be swapped in and out between RAM and disk storage (often called a page file or swap space) as needed.

### Benefits of Paging:

* **Efficient Memory Utilization**: Memory can be allocated more flexibly without needing contiguous blocks.
* **Simplified Memory Management**: The operating system can handle memory more easily by using fixed-size blocks.
* **Isolation**: Each process operates in its own logical address space, enhancing security and stability.

### Drawbacks of Paging:

* **Overhead**: The page table requires additional memory and management overhead.
* **Internal Fragmentation**: Unused space within a page can lead to wasted memory.

**Conclusion:-**

The implementation of the FIFO page replacement algorithm demonstrated effective memory management by efficiently replacing the oldest pages when cache is full. This experiment highlighted the importance of hit and fault ratios in evaluating system performance. Additionally, it reinforced key concepts like memory interleaving and paging, essential for optimizing memory access in computer systems. Overall, it enhanced our understanding of memory organization and cache structures.

**Date: 11-10-2024**