**Batch: A4 Roll No.: 16010122083**

**Experiment / assignment / tutorial No.\_\_\_\_\_\_\_**

|  |
| --- |
| **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: CO3**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Carl Hamacher, ZvonkoVranesic and SafwatZaky, “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 there in the cache.

2. When cache is not full, and requested page is not present in cache, then that requested page is copied from Main Memory to cache memory.

3. When cache is full, and requested page is not present in cache, then identify the page to be removed from cache memory. Incoming page will take it’s place. Page to be de-allocated from cache is identified by FIFO policy. FIFO is First In First Out.

**Design Steps:**

1. Start
2. Get input as maximum number of memory blocks/ pages which cache can accommodate.
3. Get input in terms of page sequence which processor will demand while executing a given program. (About 10 to 12 pages)
4. Consider next page in sequence (No more page in sequence, then it’s a end of program go to step 11)
5. If cache is not full, check if demanded page is already present in cache, if yes then it’s a hit. Increment hit counter. Go to step 4.
6. If cache is not full and demanded page is not present in cache (It’s a miss), add that page to the cache then go to step 4. Else go to step 7.
7. If cache is full, and demanded page is present in cache then it’s a hit. Increment hit counter. Go to step 4. Else go to step 8.
8. If cache is full, and demanded page is not present in cache, then it’s a situation of removing some page from cache and make space in cache for demanded page. Identify a page to be de-allocated from cache by FIFO policy. (First In First Out)
9. Remove oldest page and load demanded page to cache at the position of this outgoing page.
10. Repeat step 4 to 9 till you complete given page sequence
11. End. Print Hit Ratio.

* Display the cache status at every instance of step 9
* Mark hit by the word “Hit” below/ ahead of cache status wherever it occurs.

**Example:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Time** | **Page References in Sequence** | **Cache Page Status** | | |  |
| 1 | 2 | 2\* |  |  |  |
| 2 | 3 | 2\* | 3 |  |  |
| 3 | 2 | 2\* | 3 |  | Hit |
| 4 | 1 | 2\* | 3 | 1 |  |
| 5 | 5 | 5 | 3\* | 1 |  |
| 6 | 2 | 5 | 2 | 1\* |  |
| 7 | 4 | 5\* | 2 | 4 |  |
| 8 | 5 | 5\* | 2 | 4 | Hit |
| 9 | 3 | 3 | 2\* | 4 |  |
| 10 | 2 | 3 | 2\* | 4 | Hit |
| 11 | 5 | 3 | 5 | 4\* |  |
| 12 | 2 | 3\* | 5 | 2 |  |

\*Indicates page to be removed (If necessary) as per FIFO Policy

Hit = 3/ (3+9) = ¼ = 0.25

Code:

#include <bits/stdc++.h>

using namespace std;

int main() {

    cout << "Enter the size of the cache memory\n";

    int cap;

    cin >> cap;

    cout << "Enter the number of pages\n";

    int n;

    cin >> n;

    vector<int> v;

    int c = 0;

    int hit  = 0;

    cout << "Enter the page references\n";

    vector<int> ref(n);

    for(auto &i: ref)cin >> i;

    queue<int> q;

    for(int i = 0; i < n; i++)

    {

        int p;

        p = ref[i];

        if(c < cap)

        {

            int r = q.front();

            if(find(v.begin(), v.end(), p) != v.end())

            {

                for(int i = 0; i < c; i++)

                {

                   if(v[i] != r)cout << v[i] << " ";

                    else cout << v[i] << "\* ";

                }

                cout << "Hit\n";

                hit++;

                continue;

            }

            q.push(p);

            v.push\_back(p);

            c++;

            r  = q.front();

            for(int i = 0; i < c; i++)

            {

                if(v[i] != r)cout << v[i] << " ";

                else cout << v[i] << "\* ";

            }

        }

        else

        {

            int r = q.front();

            if(find(v.begin(), v.end(), p) != v.end())

            {

                for(int i = 0; i < c; i++)

                {

                    if(v[i] != r)cout << v[i] << " ";

                    else cout << v[i] << "\* ";

                }

                cout << "Hit\n";

                hit++;

                continue;

            }

            int x = q.front();

            q.pop();

            replace(v.begin(), v.end(), x, p);

            q.push(p);

            r = q.front();

            for(int i = 0; i < c; i++)

            {

                if(v[i] != r)cout << v[i] << " ";

                else cout << v[i] << "\* ";

            }

        }

        cout << "\n";

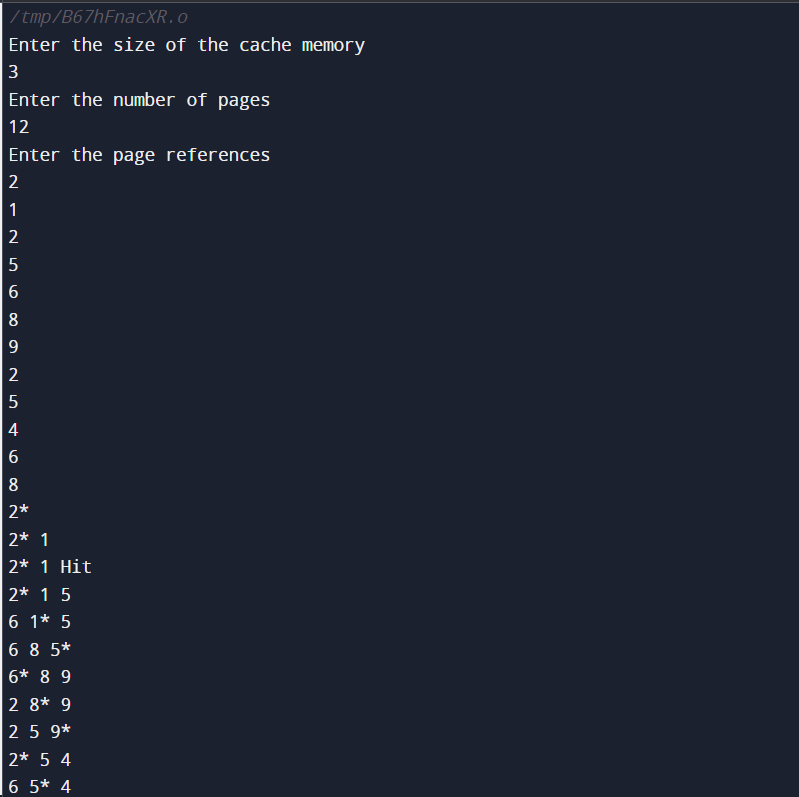
    }

    cout << "Hit Percentage: " << (float)hit/n << "\n";

    return 0;

}

Output:



**Post Lab Descriptive Questions**

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

Memory Interleaving: Memory interleaving is a technique used in computer architecture to enhance memory access performance by dividing the memory system into multiple banks or modules that can be accessed concurrently. This technique is particularly relevant in systems with a wide memory bus or multiple memory channels. The main idea behind memory interleaving is to distribute memory accesses evenly across the memory modules, which can reduce memory contention and improve memory bandwidth.

1. **Explain Paging Concept?**

Paging is a memory management technique used in computer operating systems to manage and organize the physical and virtual memory. It involves breaking up the physical memory and virtual memory into fixed-sized blocks called "pages." Similarly, the logical address space of a process is divided into fixed-sized blocks called "frames." The size of a page is typically the same as the size of a frame.

**Here's how the paging concept works:**

Virtual memory: The logical address space of a process is divided into pages, and each page is assigned a unique page number.

Physical memory: The physical memory is divided into fixed-sized frames, and each frame is also assigned a unique frame number.

Page table: To keep track of the mapping between virtual pages and physical frames, an operating system maintains a data structure called a "page table." The page table contains entries that map each virtual page number to the corresponding physical frame number.

Page table entries: Each entry in the page table typically includes the frame number and some control bits (e.g., valid/invalid bit, protection bits) to manage memory access permissions.

3**.** Consider following page address trace generated by a two level cache-main memory scheme that uses demand paging. Find the hit ratio and show the status of cache at every step ( as shown in above example) when cache has capacity of (a) 3 pages (b) 4 pages

1 6 4 5 1 4 3 2 1 2 1 4 6 7 4 1 3 1 7

**3 Pages:**

| TIME | PAGE REFERENCES IN SEQUENCES | CACHE PAGE STATUS | | |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1\* |  |  |  |
| 2 | 6 | 1\* | 6 |  |  |
| 3 | 4 | 1\* | 6 | 4 |  |
| 4 | 5 | 5 | 6\* | 4 |  |
| 5 | 1 | 5 | 1 | 4\* |  |
| 6 | 4 | 5 | 1 | 4\* | HIT |
| 7 | 3 | 5\* | 1 | 3 |  |
| 8 | 2 | 2 | 1\* | 3 |  |
| 9 | 1 | 2 | 1\* | 3 | HIT |
| 10 | 2 | 2 | 1\* | 3 | HIT |
| 11 | 1 | 2 | 1\* | 3 | HIT |
| 12 | 4 | 2 | 4 | 3\* |  |
| 13 | 6 | 2\* | 4 | 6 |  |
| 14 | 7 | 7 | 4\* | 6 |  |
| 15 | 4 | 7 | 4\* | 6 | HIT |
| 16 | 1 | 7 | 1 | 6\* |  |
| 17 | 3 | 7\* | 1 | 3 |  |
| 18 | 1 | 7\* | 1 | 3 | HIT |
| 19 | 7 | 7\* | 1 | 3 | HIT |

Hit Ratio= No of Hits/ (No of Hits+No of Miss) = 7/19 = 0.39

**4 Pages:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| TIME | PAGE REFERENCES IN SEQUENCES | CACHE PAGE STATUS | | | |  |
| 1 | 1 | 1\* |  |  |  |  |
| 2 | 6 | 1\* | 6 |  |  |  |
| 3 | 4 | 1\* | 6 | 4 |  |  |
| 4 | 5 | 1\* | 6 | 4 | 5 |  |
| 5 | 1 | 1\* | 6 | 4 | 5 | HIT |
| 6 | 4 | 1\* | 6 | 4 | 5 | HIT |
| 7 | 3 | 3 | 6\* | 4 | 5 |  |
| 8 | 2 | 3 | 2 | 4\* | 5 |  |
| 9 | 1 | 3 | 2 | 1 | 5\* |  |
| 10 | 2 | 3 | 2 | 1 | 5\* | HIT |
| 11 | 1 | 3 | 2 | 1 | 5\* | HIT |
| 12 | 4 | 3\* | 2 | 1 | 4 |  |
| 13 | 6 | 6 | 2\* | 1 | 4 |  |
| 14 | 7 | 6 | 7 | 1\* | 4 |  |
| 15 | 4 | 6 | 7 | 1\* | 4 | HIT |
| 16 | 1 | 6 | 7 | 1\* | 4 | HIT |
| 17 | 3 | 6 | 7 | 3 | 4\* |  |
| 18 | 1 | 6\* | 7 | 3 | 1 |  |
| 19 | 7 | 6\* | 7 | 3 | 1 | HIT |

Hit Ratio= No of Hits/ (No of Hits+No of Miss) = 7/19 = 0.37

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_**