Batch: B2	**Roll No.:16010122151**
Experiment / assignment / tutorial No. 6	

TITLE: Implementation of LRU Page Replacement Algorithm.

AIM: The LRU algorithm replaces the least recently used that is the last accessed memory block from user.

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.

Pre Lab/ Prior Concepts:

It follows a simple logic, while replacing a page/block it will replace that page/block which is least recently used out of all pages/blocks present in cache.

a) A hit is said to be occurred when a memory location requested is already there in the cache.
b) When cache is not full, the newly demanded block is added to cache if not present already
c) When cache is full, and if demanded block is not present in cache, then this demanded incoming block will replace the least recently used block.

Algorithm:

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 LRU policy. (Least Recently Used page)
9. Remove least recently used page and load demanded page to cache at the position of 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 (As shown in example below)
- 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	2*	5	1	
6	2	2	5	1*	Hit
7	4	2	5*	4	
8	5	2*	5	4	Hit
9	3	3	5	4*	
10	2	3	5*	2	
11	5	3*	5	2	Hit
12	2	3*	5	2	Hit

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

Hit = 5/ (5+7) = 5/12 = 0.4166

Code:

```
#include <iostream>

#include <vector>

#include  <list>

#include <unordered_map>



using namespace std;

void printCacheStatus(const list<int>& cache)

{

    for (int page : cache)

    {

        cout << page << " ";

    }

    cout << endl;

}

int main()

{

    int cacheSize, numPages;

    cout << "Enter the maximum number of memory blocks/pages the cache can accommodate: ";

    cin >> cacheSize;

    cout << "Enter the number of pages in the sequence (10 to 12 pages): ";

    cin >> numPages;

    vector<int> pageSequence(numPages);

    cout << "Enter the page sequence: ";

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

    {

        cin >> pageSequence[i];

    }

    unordered_map<int, list<int>::iterator> pageMap;

    list<int> cache;

    int hitCount = 0;
```

```
    for (int i = 0; i < numPages; ++i)
    {
        int page = pageSequence[i];
        if (pageMap.find(page) != pageMap.end())
        {
            cout << "Hit: ";
            hitCount++;
            cache.erase(pageMap[page]);
        }
        else
        {

            cout << "Miss: ";
            if (cache.size() >= cacheSize)
            {

                int lruPage = cache.back();
                cache.pop_back();
                pageMap.erase(lruPage);
                cout << "Removed " << lruPage << "* ";
            }
        }

        cache.push_front(page);
        pageMap[page] = cache.begin();
        printCacheStatus(cache);
    }
    double hitRatio = static_cast<double>(hitCount) / numPages;
    cout << "Hit Ratio: " << hitRatio << endl;
}
```

Output:

```
/tmp/fUp56nTvLG.o
Enter the maximum number of memory blocks the cache can accommodate in it: 3
Enter the number of pages in the sequence: 12
Enter the page sequence: 2 3 2 1 5 2 4 5 3 2 5 2
Miss: ->2
Miss: ->3 2
Hit: ->2 3
Miss: ->1 2 3
Miss: ->Removed 3* 5 1 2
Hit: ->2 5 1
Miss: ->Removed 1* 4 2 5
Hit: ->5 4 2
Miss: ->Removed 2* 3 5 4
Miss: ->Removed 4* 2 3 5
Hit: ->5 2 3
Hit: ->2 5 3
Hit Ratio: 0.416667
```

Post Lab Descriptive Questions

1. Define hit rate and miss ratio?

Hit rate and miss ratio are metrics used to evaluate the performance of a cache memory system.
A hit ratio is a calculation of cache hits, and comparing them with how many total content requests were received.

$$\frac{\text{Number of Cache Hits}}{\text{Total Number of Memory Accesses}}$$

A higher hit rate indicates that a larger portion of memory accesses are satisfied by the cache, which is desirable because it reduces the need to access slower main memory.
Miss Ratio: A miss ratio is the flip side of this where the cache misses are calculated and compared with the total number of content requests that were received.
Miss ratio=1-Hit ratio

2. What is the need for virtual memory?

Virtual memory is crucial because it allows computers to:
1. **Run Large Programs:** It enables the execution of programs larger than available physical RAM.
2. **Enable Multitasking:** Multiple programs can run simultaneously despite limited RAM.
3. **Simplify Programming:** Developers can assume a larger memory space, simplifying coding.
4. **Provide Security:** Ensures memory isolation between processes, enhancing system security.
5. **Facilitate Memory Management:** Allows the OS to manage physical memory efficiently.
6. **Enhance System Reliability:** Prevents crashes by using disk space as an extension of RAM.
7. **Support Large Data:** Essential for applications dealing with massive datasets.
8. **Ensure Portability:** Abstracts hardware differences, making software portable across systems.

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

(a) 3 Pages

Time	Page References in Sequence	Cache Page Status			
1	1	1			Miss
2	6	1	6		Miss
3	4	1*	6	4	Miss
4	5	6*	4	5	Miss
5	1	4	5	1	Miss
6	4	5*	1	4	Hit
7	3	1*	4	3	Miss
8	2	4	3	2	Miss
9	1	3	2	1	Miss
10	2	3	1	2	Hit
11	1	3	2	1	Hit
12	4	2	1	4	Miss
13	6	1	4	6	Miss
14	7	4	6	7	Miss
15	4	6	7	4	Hit
16	1	7	4	1	Miss
17	3	4	1	3	Miss
18	1	4	3	1	Hit
19	7	3	1	7	Miss

Hit ratio: 0.263158

(b) 4 Pages

Time	Page References in Sequence	Cache Page Status				
1	1	1				Miss
2	6	1	6			Miss
3	4	1	6	4		Miss
4	5	1	6	4	5	Miss
5	1	6	4	5	1	Hit
6	4	6*	5	1	4	Hit
7	3	5*	1	4	3	Miss
8	2	1	4	3	2	Miss
9	1	4	3	2	1	Hit
10	2	4	3	1	2	Hit
11	1	4	3	2	1	Hit
12	4	3*	2	1	4	Miss
13	6	2*	1	4	6	Miss
14	7	1	4	6	7	Hit
15	4	1	6	7	4	Hit
16	1	6*	7	4	1	Miss
17	3	7	4	1	3	Hit
18	1	7	4	3	1	Hit
19	7	4	3	1	7	Hit

Hit Ratio= 0.416667

Date: 12/10/2023