Batch: C3 **Roll No.: 16010121226**

Experiment / assignment / tutorial No.____8___

Grade: AA / AB / BB / BC / CC / CD /DD

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: (Mention CO/CO's attained here)
CO3- 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.

Pre Lab/ Prior Concepts:

It follows a simple logic, while replacing it will replace that page which has least recently used out of all.

a) A hit is said to be occurred when a memory location requested is already in the cache.

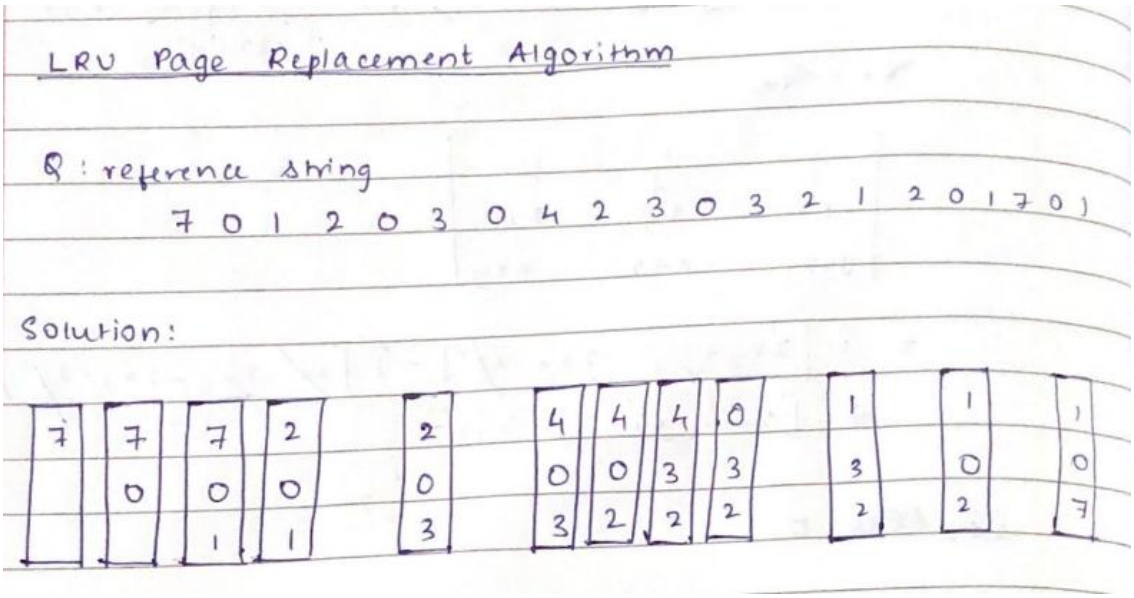
b) When cache is not full, the number of blocks is added.

c) When cache is full, the block is replaced which is recently used

Algorithm:

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 least recently used element
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:



Code:

```c
#include <stdio.h>
#include <stdlib.h>

int a[1][3];
int pr[20];
int num[3]={-1}; int count[3]={0};

int main()
{
    int m,hit=0,f=0,j,i,k,l;
    printf("Length of reference string: ");
    scanf("%d",&m);

    for(j=0;j<3;j++)
    {
        a[0][j]=-1;
    }

    printf("\nReference string:\n");

    for(i=0;i<m;i++)
    {
        scanf("%d",&pr[i]);
    }

    printf("\n");
    printf("LRU output is:\n");
    int c=0;

    while(c<m)
    {
        int p = pr[c];
        if(a[0][0]!=-1 && a[0][1]!=-1 && a[0][2]!=-1)
        {
            if(p!=num[0] && (p!=num[1] && p!=num[2]))
            {
```

```
                if((a[0][1]==pr[c-1]&&a[0][2]==pr[c-
2])||(a[0][1]==pr[c-2]&&a[0][2]==pr[c-1]))
                {
                    a[0][0]=-1;
                    count[0]=0;
                }

                else if((a[0][0]==pr[c-1]&&a[0][2]==pr[c-
2])||(a[0][0]==pr[c-2]&&a[0][2]==pr[c-1]))
                {
                    a[0][1]=-1;
                    count[1]=0;
                }

                else
                {
                    a[0][2]=-1; count[2]=0;
                }
            }

        }

        for(j=0;j<3;j++)
        {
            if(a[0][j]== -1)
            {
                a[0][j] = p;
                f++;
                num[j]=p;

                for(k=0;k<3;k++)
                {
                    if(a[0][k]>=0)
                    {
                        count[k]++;
                    }
                }
```

```
                break;
            }

            else if(a[0][j]== p)
            {
                hit++;
                for(k=0;k<3;k++)
                {
                    if(a[0][k]>=0)
                    {
                        count[k]++;
                    }
                }
                break;
            }
        }

        printf("%d - ",p);
        for(l=0 ;l<3;l++)
        {
            if(a[0][l]==-1)
            {
                printf("- ");
            }
            else
            {
                printf("%d ",a[0][l]);
            }
        }
        printf("\n");
        c++;
    }

    // printing hits, misses and hit ratio
    printf("\n");
    printf("\nHits : %d ",hit);
    printf("\nMiss : %d ",f);
    double ratio = (double)hit/m;
```

```
    printf("\nHIT Ratio : %.2lf",ratio);
    return 0;
}
```

Output:

```
Length of reference string: 20

Reference string:
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

LRU output is:
7 - 7 - -
0 - 7 0 -
1 - 7 0 1
2 - 2 0 1
0 - 2 0 1
3 - 2 0 3
0 - 2 0 3
4 - 4 0 3
2 - 4 0 2
3 - 4 3 2
0 - 0 3 2
3 - 0 3 2
2 - 0 3 2
1 - 1 3 2
2 - 1 3 2
0 - 1 0 2
1 - 1 0 2
7 - 1 0 7
0 - 1 0 7
1 - 1 0 7


Hits : 8
Miss : 12
HIT Ratio : 0.40
```

Post Lab Descriptive Questions

1. Define hit rate and miss ratio?

The Cache Hit Ratio is the ratio of the number of cache hits to the number of lookups, usually expressed as a percentage. Depending on the nature of the cache, expected hit ratios can vary from 60% to greater than 99%. A cache hit ratio of 90% and higher means that most of the requests are satisfied by the cache. A value below 80% on static files indicates inefficient caching due to poor configuration. Cache hit ratio = [Cache Hits / (Cache Hits + Cache Misses)] x 100 % Cache miss ratio is a state where the data requested for processing by a component or application is not found in the cache memory. It causes execution delays by requiring the program or application to fetch the data from other cache levels or the main memory.

2. What is the need for virtual memory?

System may be running on low physical memory that is not allowing the loader to load all the pages of executable into memory. Even if the processes currently sitting in memory are swapped out, physical memory requirements of the new process are not met. This is where virtual (aka unreal) memory comes into place. Demand Paging is a technique to implement virtual memory. Demand paging allows us to run programs in memory even if there is insufficient memory to begin with. This is the principle behind virtual memory. It is unreal memory. Each process has its own contiguous virtual address space comprising of code, data, bss, heap, and stack regions. Virtual address space is contiguous range of logical addresses (the one generated by CPU). The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Conclusion

Hence, LRU page replacement algorithm was successfully studied and implemented.

Date: **Signature of faculty in-charge**