EXPERIMENT NO:5(B)

AIM:

To execute the LRU algorithm.

THEORY:

A good approximation to the optimal algorithm is based on the observation that pages that have been heavily used in the last few instructions will probably be heavily used again in the next few. Conversely, pages that have not been used for ages will probably remain unused for a long time. This idea suggests a realizable algorithm: when a page fault occurs, throw out the page that has been unused for the longest time. This strategy is called LRU (Least Recently Used) paging.

Although LRU is theoretically realizable, it is not cheap. To fully implement LRU, it is necessary to maintain a linked list of all pages in memory, with the most recently used page at the front and the least recently used page at the rear. The difficulty is that the list must be updated on every memory reference. Finding a page in the list, deleting it, and then moving it to the front is a very time consuming operation, even in hardware (assuming that such hardware could be built).

However, there are other ways to implement LRU with special hardware. Let us consider the simplest way first. This method requires equipping the hardware with a 64-bit counter, *C*, that is automatically incremented after each instruction. Furthermore, each page table entry must also have a field large enough to contain the counter. After each memory reference, the current value of *C* is stored in the page table entry for the page just referenced. When a page fault occurs, the operating system examines all the counters in the page table to find the lowest one. That page is the least recently used.

Now let us look at a second hardware LRU algorithm. For a machine with *n* page frames, the LRU hardware can maintain a matrix of *n* &acute; *n* bits, initially all zero. Whenever page frame *k* is referenced, the hardware first sets all the bits of row *k* to 1, then sets all the bits of column *k* to 0. At any instant, the row whose binary value is lowest is the least recently used, the row whose value is next lowest is next least recently used, and so forth. The workings of this algorithm are given in Fig. 4-3 for four page frames and page references in the order

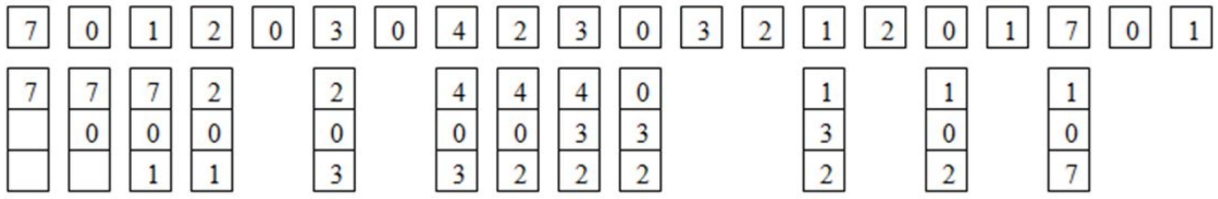
ALGORITHM:

Explanation –

Consider the same page reference string and find out total number of page faults using least recently used algorithm. Assume total number of free frames are 3.

Page Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

- Initially all three frames are free so the first three page references for pages 7, 0 and 1 will result in page fault so first, second and third free frames will be allocated to page 7, 0, and page 1 respectively. At this point all the three frames are occupied and we have no more free frames available.



- Reference for page 2 will generate page fault as page 2 is not available in the memory and since we don't have any free frame available, swapping is required according to least recently used algorithm by finding the page that has used least in the recent past.

- To look back in the past we have to check in the backward direction of in the page reference string starting from the current page reference i.e. page 2. We can figure out that page 1 and page 0 is recently used so page 7 will be selected as it is the least recently used page. Page 7 will be replaced by page 2.
- Reference for the page 0 will generate no page fault as it is already available in the memory. Then after reference for page 3 will result in the page fault and by looking in the backward direction, we found the page as least recently used and replaced with page 3.
- Similarly all the reference will be checked against the available pages in the memory and in the page reference string to find out least recently used page. You can see complete calculation in the above figure.
- The LRU is quite good compare to FIFO but the major problem is how to implement LRU algorithm. The problem is to determine an order for the frames defined by the time of last use. It can be implemented by two ways.

CODE:

```
#include<stdio.h>


 findLRU(int time[], int n){

  int i, minimum = time[0], pos = 0;


  for(i = 1; i < n; ++i){

            if(time[i] < minimum){

                      minimum = time[i];

                      pos = i;

            }

  }


  return pos;




 main()



nt no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1,
g2, i, j, pos, faults = 0;
```

```
  printf("Enter number of frames: ");

  scanf("%d", &no_of_frames);


  printf("Enter number of pages: ");

  scanf("%d", &no_of_pages);


  printf("Enter reference string: ");


for(i = 0; i < no_of_pages; ++i){

  scanf("%d", &pages[i]);

}


  for(i = 0; i < no_of_frames; ++i){

  frames[i] = -1;

}


for(i = 0; i < no_of_pages; ++i){

  flag1 = flag2 = 0;


  for(j = 0; j < no_of_frames; ++j){
        if(frames[j] == pages[i]){

            counter++;

            time[j] = counter;

                flag1 = flag2 = 1;

                break;

            }

}


if(flag1 == 0){

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

            if(frames[j] == -1){

                counter++;

                faults++;

                frames[j] = pages[i];

                time[j] = counter;

                flag2 = 1;

                break;

            }

        }

}
```

```
 if(flag2 == 0){
        pos = findLRU(time, no_of_frames);
        counter++;
        faults++;
        frames[pos] = pages[i];
        time[pos] = counter;
 }

 printf("\n");

 for(j = 0; j < no_of_frames; ++j){
        printf("%d\t", frames[j]);
 }
 }

 printf("\n\nTotal Page Faults = %d", faults);

return 0;
```

OUTPUT:

```
Enter number of frames: 3

Enter number of pages: 6

Enter reference string: 5 7 5 6 7 3

5 -1 -1

5 7 -1

5 7 -1

5 7 6

5 7 6

3 7 6

Total Page Faults = 4
```

CONCLUSION:

Hence the LRU algorithm has been studied thoroughly and executed.