25	11	2021

PROGRAM 1 : CPU SCHEDULING

AIM

To implement the FCFS,SJF,Priority and Round Robin CPU scheduling algorithm.

ALGORITHM

STEP 1 : Start.
STEP 2 : Read the number of processes.
STEP 3 : Read arrival time,burst time and priority of each processes.

(A) For FCFS

STEP A1 : Process with least arrival time is executed first and find its performance parameters.
STEP A2 : Mark the process as executed.
STEP A3 : Repeat steps 4 and 5 for all processes and calculate average of performance parameters.

(B) For SJF

STEP B1 : Find process with least burst time and whose arrival time <= system time.
STEP B2 : The process got in step 7 is executed first, and find its performance parameters.
STEP B3 : Mark the process as executed.
STEP B4 : Repeat steps 7,8,9 for all processes and calculate average of performance parameters.

(C) For Priority

STEP C1 : Find the process having highest priority and whose arrival time <= system time.
STEP C2 : Process got in step 11 is executed first and find its performance

parameters.

STEP C3 : Mark the processes as executed.

STEP C4 : Repeat steps 11,12,13 for all processes and calculate average of performance parameters.

(D) For Round Robin

STEP D1 : Set the time slice to one second.

STEP D2 : Each process is assigned with a variable field to store the number of times the process is executed.

STEP D3 : Select the process with the least value in step 16 and whose arrival time <= system time.

STEP D4 : Process selected in step 17 is executed for one second.

STEP D5 : Increment its number of times of execution and decrement its remaining burst time.

STEP D6 : Update its performance parameters.

STEP D7 : Continue steps 17,18,19,20 upto when the remaining burst time of all processes become zero.

STEP 4 : Print the average of performance parameters of FCFS,SJF,Priority and Round Robin CPU scheduling algorithms.

STEP 5 : Stop.

PROGRAM CODE (CPU SCHEDULING)

```
#include<stdio.h>
struct performance_parameters
{
   int rt;
   int ct;
   int tt;
   int wt;
};
//function to read inputs
void readInputs(int n,int *bsum,int arr[][5])
{
   for(int i=0;i<n;i++)
   {
      printf("enter details of process %d\n",(i+1));
      printf("enter burst time\n");
      scanf("%d",&arr[i][0]);
      printf("enter arrival time\n");
      scanf("%d",&arr[i][1]);
      arr[i][2]=1;
      arr[i][3]=0;
      printf("enter priority\n");
      scanf("%d",&arr[i][4]);
      *bsum=*bsum+arr[i][0];
   }
}
//function to calculate execution sequence and perfoemance parameteres of
fcfs
void fcfs(int n,int arr[][5],struct performance_parameters pp[])
{
   printf("FCFS\n----\n");
   int first,small,l=0,systime=0,k=0;
   first=0;//variable used to refer the process, which is to be executed first in
each iteration in the for loop
   for(int i=0;i<n;++i)
   {
      small=100;//variable used to compare the arrival time of processes
      for(int j=0;j<n;++j)
      {
```

```
            if ((arr[j][2] != 0)&&(arr[j][1]<=systime))
            {
               if(arr[j][1]<small)
               {
                  small=arr[j][1];
                  first=j;
               }
               l=1;
            }
         }
         if(l==1)
         {
            if(k==0)
            {
               k=1;
               systime=arr[first][1];
            }
            arr[first][2]=0;
            //printf("executing process %d\n",(first+1));
            pp[first].rt=systime-arr[first][1];
            pp[first].ct=systime+arr[first][0];
            pp[first].tt=pp[first].ct-arr[first][1];
            pp[first].wt=systime-arr[first][1];
            systime=systime+arr[first][0];
            l=0;
         }
         else
         {
            systime=systime+1;
            i--;
         }
      }
   }
   //function to calculate execution sequence and perfoemance parameteres of
   sjf
   void sjf(int n,int arr[][5],struct performance_parameters pp[])
   {
      printf("sjf\n---\n");
      int first,small,l=0,systime=0,k=0;
```

first=0;//variable used to refer the process, which is to be executed first in each iteration in thefor loop

```
for(int i=0;i<n;++i)
{
   small=100;//variable used to compare the BURST time of processes
   for(int j=0;j<n;++j)
   {
      if ((arr[j][2] != 0)&&(arr[j][1]<=systime))
      {
         if(arr[j][0]<small)
         {
            small=arr[j][0];
            first=j;
         }
         l=1;
      }
   }
   if(l==1)
   {
      if(k==0)
      {
         k=1;
         systime=arr[first][1];
      }
      arr[first][2]=0;
      //printf("executing process %d\n",(first+1));
      pp[first].rt=systime-arr[first][1];
      pp[first].ct=systime+arr[first][0];
      pp[first].tt=pp[first].ct-arr[first][1];
      pp[first].wt=systime-arr[first][1];
      systime=systime+arr[first][0];
      l=0;
   }
   else
   {
      systime=systime+1;
      i--;
   }
}
```

```
}
//function to calculate execution sequence and perfoemance parameteres of
priority
void priority(int n,int arr[][5],struct performance_parameters pp[])
{
   printf("PRIORITY\n--------\n");
   int first,small,l=0,systime=0,k=0;
   first=0;//variable used to refer the process, which is to be executed first in
each iteration in thefor loop
   for(int i=0;i<n;++i)
   {
      small=100;//variable used to compare the priority of processes
      for(int j=0;j<n;++j)
      {
         if ((arr[j][2] != 0)&&(arr[j][1]<=systime))
         {
            if(arr[j][4]<small)
            {
               small=arr[j][4];
               first=j;
            }
            l=1;
         }
      }
      if(l==1)
      {
         if(k==0)
         {
            k=1;
            systime=arr[first][1];
         }
         arr[first][2]=0;
         //printf("executing process %d\n",(first+1));
         pp[first].rt=systime-arr[first][1];
         pp[first].ct=systime+arr[first][0];
         pp[first].tt=pp[first].ct-arr[first][1];
         pp[first].wt=systime-arr[first][1];
         systime=systime+arr[first][0];
         l=0;
      }
```

```
        else
        {
            systime=systime+1;
            i--;
        }
    }
}
//function to calculate execution sequence and perfoemance parameteresof
round robin
void rr(int n,int bsum,int arr[][5],struct performance_parameters pp[])
{
    printf("ROUND ROBIN\n----------\n");
    int first,small,l=0,systime=0,k=0;
    first=0;
    for(int i=0;i<bsum;++i)
    {
        small=100;
        for(int j=0;j<n;++j)
        {
            if ((arr[j][2] != 0)&&(arr[j][1]<=systime))
            {
                if(arr[j][3]<small)
                {
                    small=arr[j][3];
                    first=j;
                }
                l=1;
            }
        }
        if(l==1)
        {
            if(k==0)
            {
                k=1;
                systime=arr[first][1];
            }
            //printf("executing process %d\n",(first+1));
            arr[first][3]=arr[first][3]+1;
            arr[first][0]=arr[first][0]-1;
            if(arr[first][0]<=0)
```

```
        {
            arr[first][2]=0;
            if(arr[first][0]<0)
            {
                pp[first].ct=systime;
            }
            else
            {
                pp[first].ct=systime+1;
            }
            pp[first].tt=pp[first].ct-arr[first][1];
            pp[first].wt=pp[first].tt-arr[first][3];
        }
        if(arr[first][3]==1)
        {
            pp[first].rt=systime-arr[first][1];
        }
        systime=systime+1;
        l=0;
      }
      else
      {
        systime=systime+1;
        i--;
      }
   }
}
//function to print the result
void printResult(int n,struct performance_parameters pp[])
{
   printf("\n");
        float rtsum=0,ctsum=0,ttsum=0,wtsum=0;
        printf("PID \t RESPONCE TIME \t COMPLETION TIME \t TURNAROUND
TIME \t WAITING TIME\n");
   printf("_ \t _____ \t _____ \t _____ \t ____\n");
   for(int i=0;i<n;++i)
   {
      printf("P%d   \t %d      \t  %d           \t  %d      \t
%d",i+1,pp[i].rt,pp[i].ct,pp[i].tt,pp[i].wt);
      printf("\n");
```

```
        rtsum=rtsum+pp[i].rt;
        ctsum=ctsum+pp[i].ct;
        ttsum=ttsum+pp[i].tt;
        wtsum=wtsum+pp[i].wt;
    }
    printf("\nAVERAGE RESPONCE TIME     : %f\n",(rtsum/n));
    printf("AVERAGE COMPLETION TIME   : %f\n",(ctsum/n));
    printf("AVERAGE TURNAROUND TIME   : %f\n",(ttsum/n));
    printf("AVERAGE WAITING TIME      : %f\n",(wtsum/n));
    printf("\n");
}
void main()
{
    int n;
    int bsum=0;//sum of burst times of all processes
    printf("enter the number of processes\n");
    scanf("%d",&n);
    int arr[n][5];
    //arr[x][0] used to store burst time
    //arr[x][1] used to store arrival time
    //arr[x][2] initially set to 1 for all processes and set to 0 if the process is
choosen to execute
    //arr[x][3] initially set to 0, used to count the number of times CPU allocated
for the process
    //arr[x][4] used to store priority
    readInputs(n,&bsum,arr);
    struct performance_parameters ppfcfs[n];
    fcfs(n,arr,ppfcfs);
    printResult(n,ppfcfs);
    for(int a=0;a<n;a++)
        arr[a][2]=1;
    struct performance_parameters ppsjf[n];
    sjf(n,arr,ppsjf);
    printResult(n,ppsjf);
    for(int a=0;a<n;a++)
        arr[a][2]=1;
    struct performance_parameters pppriority[n];
    priority(n,arr,pppriority);
    printResult(n,pppriority);
    for(int a=0;a<n;a++)
```

```
    arr[a][2]=1;
  struct performance_parameters pprr[n];
  rr(n,bsum,arr,pprr);
  printResult(n,pprr);
}
```

SAMPLE INPUT AND OUTPUT

INPUT :

5 4 0 1 3 0 2 7 6 1 4 11 3 2 12 2

OUTPUT :

FCFS

PID	RT	CT	TT	WT
P1	0	4	4	0
P2	4	7	7	4
P3	1	14	8	1
P4	3	18	7	3
P5	6	20	8	6

AVERAGE RESPONCE TIME : 2.800000
AVERAGE COMPLETION TIME : 12.600000
AVERAGE TURNAROUND TIME : 6.800000
AVERAGE WAITING TIME : 2.800000

SJF

PID	RT	CT	TT	WT
P1	3	7	7	3
P2	0	3	3	0
P3	1	14	8	1
P4	5	20	9	5
P5	2	16	4	2

AVERAGE RESPONCE TIME : 2.200000
AVERAGE COMPLETION TIME : 12.000000

AVERAGE TURNAROUND TIME : 6.200000
AVERAGE WAITING TIME : 2.200000

PRIORITY

PID	RT	CT	TT	WT
P1	0	4	4	0
P2	4	7	7	4
P3	1	14	8	1
P4	5	20	9	5
P5	2	16	4	2

AVERAGE RESPONCE TIME : 2.400000
AVERAGE COMPLETION TIME : 12.200000
AVERAGE TURNAROUND TIME : 6.400000
AVERAGE WAITING TIME : 2.400000

ROUND ROBIN

PID	RT	CT	TT	WT
P1	0	10	10	6
P2	1	6	6	3
P3	0	20	14	7
P4	0	17	6	2
P5	0	15	3	1

AVERAGE RESPONCE TIME : 0.200000
AVERAGE COMPLETION TIME : 13.600000
AVERAGE TURNAROUND TIME : 7.800000
AVERAGE WAITING TIME : 3.800000

RESULT

The program for CPU scheduling algorithm executed successfully and the output is verified.

25	11	2021

PROGRAM 2 : BANKER'S ALGORITHM

AIM

To implement the banker's algorithm for deadlock avoidance.

ALGORITHM

STEP 1 : Start.
STEP 2 : Read the number of processes.
STEP 3 : Read the number of resources.
STEP 4 : Set the available instances of the resources in the system using an array available[m].
STEP 5 : Read allocation and maximum need of each resources of each processes and store it into the arrays allocation[n][m] and max[n][m].
STEP 6 : Now, available = available – allocation &
need = max – allocation.
STEP 7 : Use data structure work[m] = available and Boolean finish[n] = false initially.
STEP 8 : Choose the process whose finish is false and whose need <= work.If these conditions satisfied, add the process to the safe sequence.
STEP 9 : Do, work = work + allocation of that process, set its finish = true.
STEP 10 : Repeat steps 8,9 for nxm times.
STEP 11 : If finish of all the processes is true, print 'system is in safe state' and print the safe sequence.
STEP 12 : Else, print 'system is not in safe state'.
STEP 13 : Stop.

PROGRAM CODE (BANKER'S ALGORITHM)

```
#include <stdio.h>
#include<stdbool.h>

int main()
{
	int n,m,i,j,k,count=0;
	bool flag=false,flag1=true;
	printf("enter number of resources\n");
	scanf("%d",&m);
	int available[m];
	for(i=0;i<m;i++)
	{
		printf("Enter number of instances of resource %d\n",i+1);
		scanf("%d",&available[i]);
	}
	printf("enter number of processes\n");
	scanf("%d",&n);
	int allocation[n][m];
	int max[n][m];
	int need[n][m];
	int safeSequence[n];
	for(i=0;i<n;i++)
	{
		for(j=0;j<m;j++)
		{
			printf("Enter Allocation of resource %d of process
				%d\n",(j+1),(i+1));
			scanf("%d",&allocation[i][j]);
			printf("Enter maximum need of resource %d of process
				%d\n",(j+1),(i+1));
			scanf("%d",&max[i][j]);
		}
	}
	for(i=0;i<m;i++)
	{
		for(j=0;j<n;j++)
		{
```

```
                available[i]=available[i]-allocation[j][i];
        }
}
for(i=0;i<n;i++)
{
        for(j=0;j<m;j++)
        {
                need[i][j]=max[i][j]-allocation[i][j];
        }
}

//safety algorithm

int work[m];
bool finish[n];
for(i=0;i<m;i++)
{
        work[i] = available[i];
}
for(i=0;i<n;i++)
{
        finish[i]=false;
}
for(i=0;i<n;i++)
{
        for(j=0;j<n;j++)
        {
                if(finish[j]==false)
                {
                        for(k=0;k<m;k++)
                        {
                                if(need[j][k]>work[k])
                                {
                                        flag1=false;
                                        break;
                                }
                                else
                                        flag1=true;
                        }
                        if(flag1==true)
```

```
                        {
                                for(k=0;k<m;k++)
                                {
                                        work[k]=work[k]+allocation[j][k];
                                }
                                finish[j]=true;
                                safeSequence[count]=j;
                                count++;
                        }
                    }
                }
            }
            for(i=0;i<n;i++)
    {
       if(finish[i]==true)
          flag=true;
       else
       {
          flag=false;
          break;
       }
    }
    if(flag==true)
          {
       printf("System is in safe state\n");
            printf("Safe Sequence is \n");
                for(i=0;i<n;i++)
                        printf("P%d -> ",safeSequence[i]+1);
                printf("\n");
          }
          else
                printf("System is not in safe state\n");
}
```

SAMPLE INPUT AND OUTPUT

INPUT :

Enter number of resources : 3
Enter number of instances of resource 1 : 10
Enter number of instances of resource 2 : 5
Enter number of instances of resource 3 : 7
enter number of processes : 5

Enter Allocation of resource 1 of process 1 : 0
Enter maximum need of resource 1 of process 1 : 7
Enter Allocation of resource 2 of process 1 : 1
Enter maximum need of resource 2 of process 1 : 5
Enter Allocation of resource 3 of process 1 : 0
Enter maximum need of resource 3 of process 1 : 3
Enter Allocation of resource 1 of process 2 : 2
Enter maximum need of resource 1 of process 2 : 3
Enter Allocation of resource 2 of process 2 : 0
Enter maximum need of resource 2 of process 2 : 2
Enter Allocation of resource 3 of process 2 : 0
Enter maximum need of resource 3 of process 2 : 2
Enter Allocation of resource 1 of process 3 : 3
Enter maximum need of resource 1 of process 3 : 9
Enter Allocation of resource 2 of process 3 : 0
Enter maximum need of resource 2 of process 3 : 0
Enter Allocation of resource 3 of process 3 : 2
Enter maximum need of resource 3 of process 3 : 2
Enter Allocation of resource 1 of process 4 : 2
Enter maximum need of resource 1 of process 4 : 2
Enter Allocation of resource 2 of process 4 : 1
Enter maximum need of resource 2 of process 4 : 2
Enter Allocation of resource 3 of process 4 : 1
Enter maximum need of resource 3 of process 4 : 2
Enter Allocation of resource 1 of process 5 : 0
Enter maximum need of resource 1 of process 5 : 4
Enter Allocation of resource 2 of process 5 : 0
Enter maximum need of resource 2 of process 5 : 3
Enter Allocation of resource 3 of process 5 : 2
Enter maximum need of resource 3 of process 5 : 3

OUTPUT :

System is in safe state
Safe Sequence is
P2 -> P4 -> P5 -> P1 -> P3

RESULT

The program for Banker's algorithm executed successfully and the output is verified.

04	12	2021

PROGRAM 3 : DISK SCHEDULING ALGORITHMS

AIM

To implement the FCFS,SCAN and CSCAN disk scheduling algorithms.

ALGORITHM

STEP 1 : Start.
STEP 2 : Read the number of requests,n.
STEP 3 : Read location of each of the requests.
STEP 4 : Read current head position.
STEP 5 : Read limit of the disk.

(A) For FCFS

STEP A1 : Execute requests in the order of their arrival.
STEP A2 : Move head from current position to next request position.
STEP A3 : seek time = seek time + positive of the difference between the old and new head position.
STEP A4 : Repeat steps 7,8 n times.
STEP A5 : Print the final seek time.

(B) For SCAN

STEP B1 : Search in the request list, if there is any requests for the current head position.
STEP B2 : If found, execute the request.
STEP B3 : Change head position to the next position if current head position < limit.
Repeat steps 11,12,13 and increment seek time by 1.
STEP B4 : Else, change head position to next position in the backward direction.
Repeat steps 11,12,14 and increment seek time by 1.
STEP B5 : Repeat above steps until n requests are processed.
STEP B6 : Print the final seek time.

(C) For CSCAN

STEP C1 : Search in the request list if there is any requests for the current head position.
STEP C2 : If found, process request.
STEP C3 : Chenge head position to the next position if current head position < limit.
Repeat steps 17,18,19.
STEP C4 : Else, change head position to 0 (Starting of the disk).
seek time=seek time + limit.
Repeat steps 17,18,19.
STEP C5 : Repeat above steps until n requests are processed.
STEP C6 : Print the final seek time.

STEP 6 : Stop.

PROGRAM CODE (DISK SCHEDULING)

```
#include<stdio.h>
void fcfs(int n,int data[][2],int cp)
{
   printf("FCFS\n");
   printf("\n");
   int i,j,seek_time=0;
        for(i=0;i<n;++i)
        {
                printf("executing request at position %d\n",data[i][0]);
                j=cp-data[i][0];
                if(j<0)
                {
                        j=j*-1;
                }
                cp=data[i][0];
                seek_time=seek_time+j;
        }
        printf("total seek time  is : %d\n",seek_time);
}
void scan(int n,int data[][2],int cp,int limit)
{
   printf("SCAN\n");
   printf("\n");
   int i,k=0,seek_time=0;
        while(k<n && cp<=limit)
        {
                for(i=0;i<n;++i)
                {
                        if(cp==data[i][0]&&data[i][1]==0)
                        {
                                printf("executing request at position
%d\n",data[i][0]);
                                k++;
                                data[i][1]=1;
                        }
                }
                cp++;
```

```
                seek_time++;
        }
        if(cp>limit && k<n)
        {
                seek_time--;
                cp--;
                while(k<n)
                {
                        for(i=0;i<n;++i)
                        {
                                if(cp==data[i][0]&&data[i][1]==0)
                                {
                                        printf("executing request at position
%d\n",data[i][0]);
                                        k++;
                                        data[i][1]=1;
                                }
                        }
                        cp--;
                        seek_time++;
                }
        }
        printf("total seek time  is : %d\n",(seek_time-1));
}
void cscan(int n,int data[][2],int cp,int limit)
{
   printf("CSCAN\n");
   printf("\n");
   int i,k=0,seek_time=0;
        while(k<n && cp<=limit)
        {
                for(i=0;i<n;++i)
                {
                        if(cp==data[i][0]&&data[i][1]==0)
                        {
                                printf("executing request at position
%d\n",data[i][0]);
                                k++;
                                data[i][1]=1;
                        }
```

```
                }
                cp++;
                seek_time++;
        }
        if(cp>limit && k<n)
        {
                seek_time--;
                seek_time+=limit;
                cp=0;
                while(k<n)
                {
                        for(i=0;i<n;++i)
                        {
                                if(cp==data[i][0]&&data[i][1]==0)
                                {
                                        printf("executing request at position
%d\n",data[i][0]);
                                        k++;
                                        data[i][1]=1;
                                }
                        }
                        cp++;
                        seek_time++;
                }
        }
        printf("total seek time  is : %d\n",(seek_time-1));
}

void main()
{
   int n,i,cp,limit;
        printf("Enter the number of requests\n");
        scanf("%d",&n);
        int data[n][2],data1[n][2];
        for(i=0;i<n;++i)
        {
                printf("Enter the location of data %d\n",i);
                scanf("%d",&data[i][0]);
      data1[i][0]=data[i][0];
                data[i][1]=0;
```

```
      data1[i][1]=0;
        }
        printf("enter the curent head position\n");
        scanf("%d",&cp);
        printf("enter the limit of disk \n");
        scanf("%d",&limit);
   printf("\n");
   fcfs(n,data,cp);
   printf("\n");
   scan(n,data,cp,limit);
   printf("\n");
   cscan(n,data1,cp,limit);
}
```

SAMPLE INPUT AND OUTPUT

INPUT

4 50 30 70 100 40 100

OUTPUT

FCFS

```
executing request at position 50
executing request at position 30
executing request at position 70
executing request at position 100
total seek time is : 100
```

SCAN

```
executing request at position 50
executing request at position 70
executing request at position 100
executing request at position 30
total seek time is : 130
```

CSCAN

executing request at position 50
executing request at position 70
executing request at position 100
executing request at position 30
total seek time is : 90

RESULT

The program for disk scheduling algorithm executed successfully and the output is verified.

04	12	2021

PROGRAM 4 : PAGE REPLACEMENT ALGORITHM

AIM

To implement FIFO,LRU and LFU page replacement algorithms.

ALGORITHM

STEP 1 : Start.
STEP 2 : Number of frames set to 3.
STEP 3 : Read number of page requests,n.
STEP 4 : Read the pages.

(A) For FIFO

STEP A1 : Choose the first page request, check if the page is already in the frame.
STEP A2 : If yes, print already in frame.
STEP A3 : Else, if there is a vacant frame, add the new page to the vacant frame.
STEP A4 : Else, print page fault and replace the first added page with the new page.
STEP A5 : Choose the next page request and repeat steps 6 – 9 until all the requests are processed.
STEP A6 : Print number of page faults.

(B) For LRU

STEP B1 : Choose first request, check whether the frame is in the frame.
STEP B2 : If yes, print already present.
STEP B3 : Else, if vacant frame is available, add page to that frame.
STEP B4 : Else, print page fault, Increment page fault count, replace least

recently used page with new page.

STEP B5 : Choose next page and repeat steps 12 - 15

(C) For LFU

STEP C1 : Add a field to store frequency of usage to each frame.
STEP C2 : Choose a request, check whether the page is present in the frame.
STEP C3 : If yes, print already present.
STEP C4 : Else, if a vacant frame is available, add the page to that frame.
STEP C5 : Increment frequency of usage of the current frame.
STEP C6 : Else, print page fault, Increment page fault count, Replace page with least usage count by the new page.
STEP C7 : Choose next page and repeat steps 17 – 22.
STEP C8 : Print total page faults.

STEP 5 : Stop.

PROGRAM CODE (PAGE REPLACEMENT)

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

void fifo(int n,int req[])
{
   printf("FIFO");
   printf("\n");
        int frame[3],fn=0,k=0,i,j,flag=0,pageFaultCount=0;
   for(i=0;i<n;++i)
   {
      if(fn<3)
      {
         for(j=0;j<fn;++j)
         {
            if(frame[j]==req[i])
               flag++;
         }
         if(flag==1)
         {
            printf("page %d is in frame\n",req[i]);
            flag--;
         }
         else
         {
            printf("page %d is allocated  on frame %d\n",req[i],fn);
            frame[fn]=req[i];
            fn++;
         }
      }
      else
      {
         for(j=0;j<3;++j)
         {
            if(frame[j]==req[i])
               flag++;
```

```
        }
        if(flag==1)
        {
           printf("page %d is in frame\n",req[i]);
           flag--;
        }
        else
        {
           printf("page fault occured\n");
           pageFaultCount++;
           printf("page %d replaced with page %d\n",frame[k],req[i]);
           frame[k]=req[i];
           k=(k+1)%3;
        }
     }
  }
  printf("Number of page faults is : %d\n",pageFaultCount);
}

void lru(int n,int req[])
{
  struct lru
  {
     int data;
     struct lru* next;
  };
  struct lru* ptr;
  struct lru* previous;
  struct lru* header=(struct lru*)malloc(sizeof(struct lru));
  header->data=-1;
  header->next=NULL;
  printf("LRU");
  printf("\n");
  int frame[3],fn=0,i,j,flag=0,pageFaultCount=0;
  for(i=0;i<n;++i)
  {
     if(fn<3)
     {
        for(j=0;j<fn;++j)
        {
```

```
        if(frame[j]==req[i])
            flag++;
    }
    if(flag==1)
    {
        printf("page %d is in frame\n",req[i]);
        flag--;
        ptr=header;
        previous=header;
        while(ptr->data != req[i])
        {
            previous=ptr;
            ptr=ptr->next;
        }
        previous->next=ptr->next;
        previous=ptr;
        ptr=header;
        while(ptr->next != NULL)
        {
            ptr=ptr->next;
        }
        ptr->next=previous;
        previous->next=NULL;
    }
    else
    {
        printf("page %d is allocated  on frame %d\n",req[i],fn);
        frame[fn]=req[i];
        fn++;
        ptr=header;
        while(ptr->next != NULL)
        {
            ptr=ptr->next;
        }
        struct lru* newlru=(struct lru*)malloc(sizeof(struct lru));
        ptr->next=newlru;
        newlru->data=req[i];
        newlru->next=NULL;
    }
}
```

```
        else
        {
           for(j=0;j<3;++j)
           {
              if(frame[j]==req[i])
                 flag++;
           }
           if(flag==1)
           {
              printf("page %d is in frame\n",req[i]);
              flag--;
              ptr=header;
              while(ptr->data != req[i])
              {
                 previous=ptr;
                 ptr=ptr->next;
              }
              previous->next=ptr->next;
              previous=ptr;
              ptr=header;
              while(ptr->next != NULL)
              {
                 ptr=ptr->next;
              }
              ptr->next=previous;
              previous->next=NULL;
           }
           else
           {
              printf("page fault occured\n");
              pageFaultCount++;
              printf("page %d replaced with page %d\n",(header->next)-
>data,req[i]);
              for(j=0;j<3;++j)
              {
                 if(frame[j]==(header->next)->data)
                    break;
              }
              frame[j]=req[i];
              ptr=header->next;
```

```
            header->next=ptr->next;
            free(ptr);
            ptr=header;
            while(ptr->next != NULL)
            {
               ptr=ptr->next;
            }
            struct lru* newlru=(struct lru*)malloc(sizeof(struct lru));
            ptr->next=newlru;
            newlru->data=req[i];
            newlru->next=NULL;
         }
      }
   }
   printf("Number of page faults is : %d\n",pageFaultCount);
}

void lfu(int n,int req[])
{
   struct lfu
   {
      int data;
      int count;
      struct lfu* next;
   };
   struct lfu* ptr;
   struct lfu* previous;
   struct lfu* header=(struct lfu*)malloc(sizeof(struct lfu));
   header->data=-1;
   header->next=NULL;
   printf("LFU");
   printf("\n");
   int frame[3],fn=0,k,l,i,j,flag=0,temp,pageFaultCount=0;
   for(i=0;i<n;++i)
   {
      if(fn<3)
      {
         for(j=0;j<fn;++j)
         {
            if(frame[j]==req[i])
```

```
            flag++;
        }
        if(flag==1)
        {
            printf("page %d is in frame\n",req[i]);
            flag--;
            ptr=header;
            while(ptr->data != req[i])
            {
                ptr=ptr->next;
            }
            ptr->count++;
        }
        else
        {
            printf("page %d is allocated  on frame %d\n",req[i],fn);
            frame[fn]=req[i];
            fn++;
            ptr=header;
            while(ptr->next != NULL)
            {
                ptr=ptr->next;
            }
            struct lfu* newlfu=(struct lfu*)malloc(sizeof(struct lfu));
            newlfu->data=req[i];
            newlfu->count++;
            ptr->next=newlfu;
        }
    }
    else
    {
        for(j=0;j<3;++j)
        {
            if(frame[j]==req[i])
                flag++;
        }
        if(flag==1)
        {
            printf("page %d is in frame\n",req[i]);
            flag--;
```

```
    ptr=header;
    while(ptr->data != req[i])
    {
       ptr=ptr->next;
    }
    ptr->count++;
 }
 else
 {
    printf("page fault occured\n");
    pageFaultCount++;
    l=100;
    ptr=header;
    while(ptr->next != NULL)
    {
       ptr=ptr->next;
       for(int t=0;t<3;++t)
       {
          if((ptr->count < l)&&(ptr->data==frame[t]))
          {
             previous=ptr;
             l=ptr->count;
          }
       }
    }
    printf("page %d replaced with page %d\n",previous->data,req[i]);
    for(j=0;j<3;++j)
    {
       if(frame[j]==previous->data)
          break;
    }
    frame[j]=req[i];
    ptr=header;
    while(ptr->data != req[i] && ptr->next != NULL)
    {
       ptr=ptr->next;
    }
    if(ptr->data == req[i])
       ptr->count++;
    else
```

```
            {
               ptr=header;
               while(ptr->next != NULL)
               {
                  ptr=ptr->next;
               }
               struct lfu* newlfu=(struct lfu*)malloc(sizeof(struct lfu));
               newlfu->data=req[i];
               newlfu->count++;
               ptr->next=newlfu;
            }
         }
      }
   }
   printf("Number of page faults is : %d\n",pageFaultCount);
}

void main()
{
   int i,n;
   printf("Number of frammes is : 3\n");
   printf("enter the number of page requests\n");
   scanf("%d",&n);
   int req[n];
   printf("enter the pages\n");
   for(i=0;i<n;++i)
      scanf("%d",&req[i]);
   printf("\n");
   fifo(n,req);
   printf("\n");
   lru(n,req);
   printf("\n");
   lfu(n,req);
}
```

SAMPLE INPUT AND OUTPUT

INPUT

14 7 0 1 2 0 3 0 4 2 3 0 3 2 3

OUTPUT

FIFO

page 7 is allocated on frame 0
page 1 is allocated on frame 1
page 2 is allocated on frame 2
page fault occured
page 7 replaced with page 3
page 1 is in frame
page fault occured
page 1 replaced with page 4
page fault occured
page 2 replaced with page 1
page fault occured
page 3 replaced with page 5
page fault occured
page 4 replaced with page 3
page fault occured
page 1 replaced with page 4
Number of page faults is : 6

LRU

page 7 is allocated on frame 0
page 1 is allocated on frame 1
page 2 is allocated on frame 2
page fault occured
page 7 replaced with page 3
page 1 is in frame
page fault occured
page 2 replaced with page 4
page 1 is in frame
page fault occured
page 3 replaced with page 5
page fault occured
page 4 replaced with page 3
page fault occured

page 1 replaced with page 4
Number of page faults is : 5
LFU

page 7 is allocated on frame 0
page 1 is allocated on frame 1
page 2 is allocated on frame 2
page fault occured
page 7 replaced with page 3
page 1 is in frame
page fault occured
page 2 replaced with page 4
page 1 is in frame
page fault occured
page 3 replaced with page 5
page fault occured
page 4 replaced with page 3
page fault occured
page 5 replaced with page 4
Number of page faults is : 5

RESULT

The program for page replacement algorithm executed successfully and the output is verified.

08	01	2022

PROGRAM 5 : POWER OF 2

AIM

To implement a 8086 trainer kit program that would find the n powers of 2.

ALGORITHM

STEP 1 : Start.
STEP 2 : Read count value, n.
STEP 3 : Store 2^0 = 1 directly to first output location 2000.Point to next location.
STEP 4 : Load 2 to accumulator.
STEP 5 : Store content of accumulator to output location point to next location.
STEP 6 : Left shift content of accumulator by 1 position.
STEP 7 : Repeat steps 3,4 for n times.
STEP 8 : Stop.

PROGRAM CODE

```
        MOV AX,0002
        MOV BX, [5000]
        MOV SI,2000
        MOV DX,0001
        MOV [SI], DX
        INC SI
        INC SI
        MOV [SI], AX
XX :    INC SI
        INC SI
        SHL AX,0001
        MOV [SI], AX
        DEC BX
        JNZ XX
        HLT
```

SAMPLE INPUT AND OUTPUT

INPUT :

Input 4 in memory 5000

OUTPUT :

2000:01
2001:00
2002:02
2003:00
2004:04
2005:00
2006:08
2007:00

RESULT

The program executed successfully and the output is verified.

12	02	2022

PROGRAM 6 : BUBBLE SORT

AIM

To implement a 8086 trainer kit program for sorting a given list of numbers in descending order using bubble sort algorithm.

ALGORITHM

STEP 1 : Start.
STEP 2 : Initialize CL to number of inputs.
STEP 3 : Point SI to first element of the input array .
STEP 4 : Move content of SI to AX .
STEP 5 : Increment SI.
STEP 6 : Compare AX and value of SI.
STEP 7 : Exchange is value of AX is greater.
STEP 8 : Increment SI and repeat steps 4,5,6,7,8 upto end of inputs.
STEP 9 : Decrement CL and repeart steps 3 to 9 until CL becomes zero.
STEP 10 : Stop.

PROGRAM CODE

```
        MOV SI,0500
        MOV CL, [SI]
        DEC CL
XX :    MOV SI,0500
        MOV CH, [SI]
        DEC CH
        INC SI
YY :    MOV AL, [SI]
        INC SI
        CMP AL, [SI]
        JNC ZZ
        XCHG AL, [SI]
        MOV BX, SI
        DEC BX
```

```
        XCHG AL, [BX]
ZZ :    DEC CH
        JNZ YY
        DEC CL
        JNZ XX
        HLT
```

SAMPLE INPUT AND OUTPUT

INPUT

500:05
501:06
502:04
503:02
504:03
505:05

OUTPUT

500:05
501:06
502:05
503:04
504:03
505:02

RESULT

The program executed successfully and the output is verified.

PROGRAM 7 : BINARY SEARCH

AIM

To implement a 8086 trainer kit program that would check whether a number is present in a given list of numbers using Binary Search.

ALGORITHM

STEP 1 : Start.
STEP 2 : Begin with an interval covering the whole array.
STEP 3 : If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
STEP 4 : Otherwise, narrow it to the upper half.
STEP 5 : Repeatedly check until the value is found or the interval is empty.
STEP 6 : Stop.

PROGRAM CODE

```
        ARR DW 1234H,1342H,2345H,3456H,4675H
            LEN DW ($-ARR)/2
            KEY DW 3456H
            RESULT RESW 1
            MOV BX,01H
            MOV DX,LEN
            MOV CX,KEY
AGAIN:      CMP BX,DX
            JA FAIL
            MOV AX,BX
            ADD AX,DX
            SHR AX,01
```

```
            MOV SI,AX
            DEC SI
            ADD SI,SI
            CMP CX,ARR[SI]
            JAE BIG
            DEC AX
            MOV DX,AX
            JMP AGAIN
BIG:        JE SUCCESS
            INC AX
            MOV BX,AX
            JMP AGAIN
SUCCESS:    ADD AL,'O'
            MOV RESULT,AL
            JMP DISP
FAIL:       MOV RESULT,O
DISP:       END
```

SAMPLE INPUT AND OUTPUT

INPUT

array in memory 1000:1234H,1342H,2345H,3456H,4675H
key :3457H

OUTPUT

key not found therefore memory location with label RESULT contains "0".

RESULT

The program executed successfully and the output is verified.

PROGRAM 8 : PASS ONE OF TWO PASS ASSEMBLER

AIM

To implement the pass one algorithm of the two pass assembler.

ALGORITHM

STEP 1 : Start.
STEP 2 : Read the input line.
STEP 3 : Check to see if the opcode field in the input line is “START”.
- (i) Find if there is any operand field after START; initialize the LOCCTR to the operand value.
- (ii) Other wise if there is no value in the operand field the LOCCTR is set to zero.

STEP 4 : Write the line to the intermediate file.
STEP 5 : Repeat the following for the other lines in the program until the opcode field contains END directive.
- (A) If there is a symbol in the label field.
  - (a) Check the symbol table to see if has already been stored over There.If so then it is a duplicate symbol,the error messages should be displayed.
  - (b) Other wise the symbol is entered into the SYMTAB,along with the memory address in which it is stored.
- (B) If there is an opcode in the opcode field.
  - (a) Search the OPTAB to see if the opcode is present,if so increment the location counter (LOCCTR) by three.
  - (b)
    - (i) If the opcode is WORD, increment the LOCCTR by three.
    - (ii) If the opcode is BYTE, increment the LOCCTR by one.
    - (iii) If the opcode is RESW, increment the LOCCTR by integer equivalent of the operand value *3.
    - (iv) If the opcode is RESB, increment the LOCCTR by integer equivalent of the operand value.
- (C) Write each and every line processed to the intermediate file along

with their location counters.

STEP 6 : Calculate the length of the program from the final value of the LOCCTR.

STEP 7 : Close all the opened files and exit.

STEP 8 : Stop.

PROGRAM CODE

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
        char opcode[10],mnemonic[10],operand[10],label[10],code[10];
        int l,locctr,start,length;
        FILE *fp1,*fp2,*fp3,*fp4;

        clrscr();
        /* creating input file */
        fp1=fopen("c:\\turboc3\\bin\\input.dat","w");
        fputs("COPY START 2000\n** LDA FIVE\n** STA ALPHA\n** LDCH
CHARZ\n** STCH C1\n ALPHA RESW 1\n FIVE WORD 5\n CHARZ BYTE C'EOF'\n
C1 RESB 1\n** END **",fp1);
        fclose(fp1);
        fp2=fopen("c:\\turboc3\\bin\\symtab.dat","w");
        fp3=fopen("c:\\turboc3\\bin\\inter.dat","w");
        /* creating OPTAB */
        fp4=fopen("c:\\turboc3\\bin\\optab.dat","w");
        fputs("ADD\t18\nSUB\t1C\nMUL\t20\nDIV\t24\nLDA\t18\nSUB\t1C\nM
UL\t20\nDIV\t24\nLDA\t00\nLDB\t68\nLDX\t04\nLDCH\t50\nSTA\t0C\nSTB\t7
8\nSTX\t10\nSTCH\t54\nSTART\t*\nEND\t*\n",fp4);
        fclose(fp4);
        /* opening input file for reading */
        fp1=fopen("c:\\turboc3\\bin\\input.dat","r");
        fscanf(fp1,"%s%s%s",label,opcode,operand);

        // if a starting address is specified with STRAT then locctr = address
specified with START
        // else
```

```
// locctr = 0

if(strcmp(opcode,"START")==0)
{
	start=atoi(operand);
	locctr=start;
	fprintf(fp3,"%s\t%s\t%s\n",label,opcode,operand);
	fscanf(fp1,"%s%s%s",label,opcode,operand);
}
else
	locctr=0;

/* while opcode != 'END' do the following */

fp4=fopen("c:\\turboc3\\bin\\optab.dat","r");
while(strcmp(opcode,"END")!=0)
{
	fprintf(fp3,"%d\t",locctr);
	if(strcmp(label,"**")!=0)
	fprintf(fp2,"%s\t%d\n",label,locctr);
	rewind(fp4);
	fscanf(fp4,"%s",mnemonic);
	while(strcmp(mnemonic,"END")!=0)
	{
		if(strcmp(opcode,mnemonic)==0)
		{
			locctr+=3;
			break;
		}
		fscanf(fp4,"%s",mnemonic);
	}
	if(strcmp(opcode,"WORD")==0)
		locctr+=3;
	else if(strcmp(opcode,"RESW")==0)
		locctr+=(3*(atoi(operand)));
	else if(strcmp(opcode,"RESB")==0)
		locctr+=(atoi(operand));
	else if(strcmp(opcode,"BYTE")==0)
	{
		l=strlen(operand)-3;
```

```
                locctr+=1;
            }
            fprintf(fp3,"%s\t%s\t%s\n",label,opcode,operand);
            fscanf(fp1,"%s%s%s",label,opcode,operand);
        }
        fprintf(fp3,"%d\t%s\t%s\t%s\n",locctr,label,opcode,operand);
        length=locctr-start;
        printf("\nThe length of the program is %d",length);
        fclose(fp1);
        fclose(fp2);
        fclose(fp3);
        fclose(fp4);
        getch();
}
```

SAMPLE INPUT AND OUTPUT

INPUT

```
COPY    START   2000
**      LDA     FIVE
**      STA     ALPHA
**      LDCH    CHARZ
**      STCH    C1
ALPHA   RESW    1
FIVE    WORD    5
CHARZ   BYTE    C'EOF'
C1      RESB    1
**      END     **
```

OUTPUT

INTERMEDIATE FILE :

```
        COPY    START   2000
2000    **      LDA     FIVE    33200F
2003    **      STA     ALPHA   44200C
2006    **      LDCH    CHARZ   532012
2009    **      STCH    C1      572015
200C    ALPHA   RESW    1
```

```
200F    FIVE      WORD    5          000005
2012    CHARZ     BYTE    C'EOF'     454F46
2015    C1        RESB    1
2018    **        END     **
```

SYMTAB :

```
ALPHA     200C
FIVE      200F
CHARZ     2012
C1        2015
```

RESULT

The program executed successfully and the output is verified.

PROGRAM 9 : PASS TWO OF TWO PASS ASSEMBLER

AIM

To implement the pass two algorithm of the two pass assembler.

ALGORITHM

STEP 1 : Start.
STEP 2 : Read the first line from the intermediate file.
STEP 3 : Check to see if the opcode field in the input line is “START”,if so then write the line onto the final output line.
STEP 4 : Repeat the following for the other lines in the intermediate file until the opcode fileld contains END directive.
- (i) If there is a symbol in the operand field, then the object code is assembled by combining the machine code equivalent of the instruction with the symbol address.
- (ii) If there is no symbol in the operand field, then the operand address is assigned as zero and it is assembled with the machine code equivalent of the instruction.
- (iii) If the opcode field is BYTE or WORD or RESB, then convert the constants in the operand field to the object code.
- (iv) Write the input line along with the object code onto the final output file.

STEP 5 : Close all the operand files and exit.
STEP 6 : Stop.

PROGRAM CODE

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
        char a[10],label[10],opcode[10],operand[10],symbol[10],ch;
```

```
int c=0;
int sa,st,diff,diff1=0,i,address,add,len,actual_len,finaddr,prevaddr,j=0;
char mnemonic[15][15]={"LDA","STA","LDCH","STCH"};
char code[15][15]={"33","44","53","57"};
FILE *fp1,*fp2,*fp3,*fp4;
clrscr();
fp1=fopen("c:\\turboc3\\bin\\project\\ASSMLIST.DAT","w");
fp2=fopen("c:\\turboc3\\bin\\SYMTAB.DAT","r");
fp3=fopen("c:\\turboc3\\bin\\INTER.DAT","r");
fp4=fopen("c:\\turboc3\\bin\\project\\OBJECT.DAT","w");
fscanf(fp3,"%s%s%s",label,opcode,operand);
while(strcmp(opcode,"END")!=0)
{
        if(strcmp(opcode,"START")==0)
                sa=atoi(operand);
        if(strcmp(opcode,"RESW")==0)
        {
                c=3*atoi(operand);
                diff1+=c;
        }
        if(strcmp(opcode,"RESB")==0)
        {
                c=atoi(operand);
                diff1+=c;
        }
        prevaddr=address;
        fscanf(fp3,"%d%s%s%s%s",&address,label,opcode,operand);
}
finaddr=address-sa;
diff=finaddr-diff1;
fclose(fp3);
fp3=fopen("c:\\turboc3\\bin\\project\\INTERMED.DAT","r");
fscanf(fp3,"%s%s%s",label,opcode,operand);
if(strcmp(opcode,"START")==0)
{
        fprintf(fp1,"\t%s\t%s\t%s\n",label,opcode,operand);
        fprintf(fp4,"H^%s^%s^0000%d\n",label,operand,finaddr);
        fscanf(fp3,%d%s%s%s",&address,label,opcode,operand);
        st=address;
        fprintf(fp4,"T^00%d^%d",address,diff);
```

```
	}
	while(strcmp(opcode,"END")!=0)
	{
		if(strcmp(opcode,"BYTE")==0)
		{
		fprintf(fp1,"%d\t%s\t%s\t%s\t",address,label,opcode,operand);
			len=strlen(operand);
			actual_len=len-3;
			fprintf(fp4,"^");
			for(i=2;i<(actual_len+2);i++)
			{
				fprintf(fp1,"%x",operand[i]);
				fprintf(fp4,"%x",operand[i]);
			}
			fprintf(fp1,"\n");
		}
		else if(strcmp(opcode,'WORD")==0)
		{
			len=strlen(operand);

	fprintf(fp1,"%d\t%s\t%s\t%s\t00000%s\n",address,label,opcode,operan
d,operand);
			fpintf(fp4,"^00000%s:,operand);
		}
		else if((strcmp(opcode,"RESB")==0) ||
(strcmp(opcode,"RESW")==0))

	fprintf(fp1,"%d\t%s\t%s\t%s\n",address,label,opcode.operand);
		else
		{
			while(strcmp(opcode,mnemonic[j]!=0)
			{
				j++;
				getch();
			}
			if(strcmp(opcode,"COPY")==0)

	fprintf(fp1,"%d\t%s\t%s\t%s\t%s0000\n",address,label,opcode,operand,
code[j]);
			else
```

```
                {
                        rewind(fp2);
                        fscanf(fp2,"%s%d",symbol,&add);
                        while(strcmp(operand,symbol)!=0)
                        fscanf(fp2,"%s%d",symbol,&add);
                        printf("\n j=%d",j);
                        getch();

        fprintf(fp1,"%d\t%s\t%s\t%s\t%s%d\n",address,label,opcode,operand,c
ode[j],add);
                        fprintf(fp4,"^%s%d",code[j],add);
                }
        }
        fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);
    }
    fprintf(fp1,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);
    fprintf(fp4,"\nE^00%d",st);
    printf("\nIntermediate file is converted into object code:);
    fcloseall();

    printf("\n\nThe contents of the Intermediate file :\n\n\t");
    fp3=fopen("c:\\turboc3\\bin\\project\\INTER.DAT","r");
    ch=fgetc(fp3);
    while(ch!=EOF)
    {
        printf("%c",ch);
        ch=fgetc(fp3);
    }
    printf("\n\nThe contents of the Symbol Table :\n\n");
    fp2=fopen("c:\\turboc3\\bin\\SYMTAB.DAT","r");
    ch=fgetc(fp2);
    while(ch!=EOF)
    {
        printf("%c",ch);
        ch=fgetc(fp2);
    }
    printf("\n\nThe contents of the Output file :\n\n");
    fp1=fopen("c:\\turboc3\\bin\\project\\ASSMLIST.DAT","r");
    ch=fgetc(fp1);
    while(ch!=EOF)
```

```
        {
                printf("%c",ch);
                ch=fgetc(fp1);
        }
        printf("\n\nThe contents of the Object code file :\n\n");
        fp4=fopen("c:\\turboc3\\bin\\project\\OBJCODE.DAT","r");
        ch=fgetc(fp4);
        while(ch!=EOF)
        {
                printf("%c",ch);
                ch=fgetc(fp4);
        }
        fcloseall();
}
```

SAMPLE INPUT AND OUTPUT

```
ALPHA       200C
FIVE        200F
CHARZ       2012
C1          2015
```

The contents of the Output file :

```
          COPY      START     2000
2000      **        LDA       FIVE      33200F
2003      **        STA       ALPHA     44200C
2006      **        LDCH      CHARZ     532012
2009      **        STCH      C1        572015
200C      ALPHA     RESW      1
200F      FIVE      WORD      5         000005
2012      CHARZ     BYTE      C'EOF'    454F46
2015      C1        RESB      1
2018      **        END       **
```

The contents of the Object code file :

```
H^COPY^002000^000022
T^002000^18^332015^442012^532018^572021^000005^454F46
E^002000
```

RESULT

The program executed successfully and the output is verified.

PROGRAM 10 : AVERAGE OF A SET OF NUMBERS(NASM)

AIM

To implement a NASM program to find the average of a set of numbers.

ALGORITHM

STEP 1 : Start.
STEP 2 : Read the count.
STEP 3 : Read the numbers.
STEP 4 : Find the sum of the numbers.
STEP 5 : Divide sum with the count.
STEP 6 : Print the result.
STEP 7 : Stop.

PROGRAM CODE

```
global   main
        extern   atoi
        extern   printf
        default  rel

        section  .text
main:
        dec      rdi
        jz       nothingToAverage
        mov      [count], rdi
accumulate:
        push     rdi
        push     rsi
        mov      rdi, [rsi+rdi*8]
        call     atoi
        pop      rsi
        pop      rdi
        add      [sum], rax
```

```
        dec     rdi
        jnz     accumulate
average:
        cvtsi2sd xmm0, [sum]
        cvtsi2sd xmm1, [count]
        divsd   xmm0, xmm1
        mov     rdi, format
        mov     rax, 1

        sub     rsp, 8
        call    printf
        add     rsp, 8
        ret

nothingToAverage:
        mov     rdi, error
        xor     rax, rax
        call    printf
        ret

        section  .data
count:  dq      0
sum:    dq      0
format: db      "%g", 10, 0
error:  db      "There are no command line arguments to average", 10, 0
```

SAMPLE INPUT AND OUTPUT

INPUT

```
5
1  2  3  4  5
```

OUTPUT

```
3
```

RESULT

The program executed successfully and the output is verified.

PROGRAM 11 : BINARY TO BCD(NASM)

AIM

To write a NASM program to implement binary to BCD conversion.

ALGORITHM

STEP 1 : Start.
STEP 2 : Shift the binary number left one bit.
STEP 3 : If 8 shifts have taken place, the BCD number is in the Hundreds, Tens, and Units column.
STEP 4 : If the binary value in any of the BCD columns is 5 or greater, add 3 to that value in that BCD column.
Go to 1.
STEP 5 : Stop.

PROGRAM CODE

```
bits 64
        push    rdx                     ;save uesed registers
        push    rcx
        mov     rax,rdi                  ;value in rax
        and     rax,0xFFFF                  ;only least significant word counts
        mov     cl,13                    ;bits to shift in = 16 – 3
        ror     rax,cl                  ;put 3 most significant bits in al
.repeat:
   push    rcx                      ;save bit counter
   ;branch free conversion of word to bcd
   mov     rdx,rax                     ;value in rdx
   mov     rcx,0x0000000003333333          ;add 3 to each digit
   add     rdx,rcx
   mov     rcx,0x0000000008888888          ;keep fourth bit of result
   and     rdx,rcx
```

```
   shr    rdx,3                  ;put in it bit position 0
   add    rax,rdx                  ;add 1 to rax if rdx = 1
   shl    rdx,1
   add    rax,rdx                  ;add 2 to rax if rdx = 1
   rol    rax,1                  ;shift in next bit left of rax
   pop    rcx                    ;restore loop counter
   loop   .repeat                   ;decrement bit counter and repeat if still not
zero
   pop    rcx                    ;restore used registers
   pop    rdx
   ret
```

SAMPLE INPUT AND OUTPUT

input : 1101
output : 00010011

RESULT

The program executed successfully and the output is verified.

PROGRAM 12 : BCD TO BINARY(NASM)

AIM

To write a NASM program to implement BCD to binary conversion.

ALGORITHM

STEP 1 : Start.
STEP 2 : Add 3 to each nibble.
STEP 3 : Keep bit 3 of each nibble.
STEP 4 : Divide result by 4.
STEP 5 : Subtract either 2 or 0 from each nibble in ax.
STEP 6 : Divide result by 2.
STEP 7 : Subtract either 1 or 0 from each nibble in ax .
STEP 8 : Save lowest nibble.
STEP 9 : Repeat until number is converted.
STEP 10 : Result in 5 lowest nibbles.
STEP 11 : Stop.

PROGRAM CODE

```
bits 64

global wordbcd2bin

section .text

wordbcd2bin:
   ;convert packed bcd in AX to binary in EAX.
   push    rcx
   push    rdx
   mov     rax,rdi
   and     ax,0xFFFF
   ror     eax,1
```

```
   mov     rcx,10
.repeat:
   mov     dx,ax
   add     dx,0x3333
   and     dx,0x8888
   shr     dx,2
   sub     ax,dx
   shr     dx,1
   sub     ax,dx
   ror     eax,1
   loop    .repeat
   rol     eax,11
   pop     rdx
   pop     rcx
   ret
```

SAMPLE INPUT AND OUTPUT

input : 00111001
output : 100111

RESULT

The program executed successfully and the output is verified.