CONTENTS

OS EXPERIMENTS

EXPERIMENT NO : 01 DATE : 02/12/2021

CPU SCHEDULING

AIM

To simulate following CPU scheduling algorithms to find turnaround time and waiting time:

a) FCFS
b) SJF
c) Round Robin
d) Priority

ALGORITHM

OVERALL STRUCTURE

```
start while(1)
print ("------MENU------:1.FCFS 2.SJF 3.Priority 4.Round Robin 5.Exit") print ("Enter
the option: ") read (option)  while(option < 1 || option > 5)          print ("Invalid option")
              break if(option
== 1) //FCFS
   print ("Enter the number of processes: ")
   read(n)              FCFSinput(p, n)              FCFS(p,
n)            displaytable1(p, n)           printaverage(p, n)
else if(option == 2) //SJF
   print ("Enter the number of processes: ")
   read(n)              SJFinput(p, n)               SJF(p, n)
   displaytable1(p, n)           printaverage(p, n) else
if(option == 3) // Priority Scheduling              print
("Enter the number of processes: ")                read (n)
              Priorityinput(p, n)
               PrioritySchedule(p, n)
              displaytable2(p, n)
   printaverage(p, n)
else if(option == 4) // Round Robin Scheduling
            print ("Enter the number of processes: ")
            read (n)
            print ("Enter the Time Quantum: ")
            read (tq)
            FCFSinput(p, n)
            RRS(p, n, tq)
      else printaverage(p, n)

            break
end
```

FCFSinput(process p[], int n)

Accept process array p, sorted by arrival time

SJFinput(process p[], int n)

Accept process array p, sorted by burst time and internally sorted by arrival time

Priorityinput(process p[], int n)

Accept process array p, sorted by priority and internally sorted by arrival time

displaytable1(process p[], int n)

Display the processes details in tabular form

displaytable2(process p[], int n)

Display the processes details in tabular form with priority

printaverage(process p[], int n)

```
avgwt = 0  avgtat = 0
for(i = 0; i < n; i++)
   avgwt += p[i].wt
   avgtat += p[i].tat avgwt /=
n avgtat /= n print (avgwt,
avgtat)
end
```

void FCFS(process p[], int n)

```
clock = p[0].at for(i =
0; i < n; i++)
   p[i].wt = clock -
p[i].at       p[i].tat =
p[i].wt + p[i].bt
                  if(i < n - 1 && clock + p[i].bt < p[i + 1].at)
                           clock = p[i + 1].at
   else                    clock += p[i].bt
end
```

void SJF(process p[], int n)

```
  process temp   clock = 0
for(i = 0; i < n; i++)
   selected_process = -1      for(j = i; j <
n; j++)
                           if(p[j].at <= clock)
   selected_process = j                              break
                  if(selected_process == -1)
                           selected_process = i                    for(j = i+1;
j < n; j++)                        if(p[j].at < p[selected_process].at)
                           selected_process = j
                                    clock = p[selected_process].at
                     p[selected_process].wt = clock - p[selected_process].at
```

```
            p[selected_process].tat = p[selected_process].wt + p[selected_process].bt
  clock += p[selected_process].bt      temp = p[selected_process]          for(j =
selected_process - 1; j >= i; j--)       p[j + 1] = p[j]      p[i] = temp
end
```

PrioritySchedule(process p[], int n)

SJF(p, n)

RRS(process p[], int n, int tq)

```
for(i = 0; i < n; i++)
            bt[i] = p[i].bt
clock = p[0].at PC = n queue = 1 while(PC)     if(p[0].bt <=
tq)                    timeTaken = p[0].bt         p[0].tat = clock
+ timeTaken - p[0].at          p[0].wt = p[0].tat - bt[p[0].pid -
1]            else
                        timeTaken = tq
                    while(queue < PC && clock + timeTaken >= p[queue].at)
                        queue++
                if(timeTaken - p[0].bt == 0)
                        PC--
                          queue--
  p[0].bt = 0;                    temp = p[0]
            for(i = 1; i <= PC; i++)
            p[i - 1] = p[i]
                                  p[PC] = temp
            else                  p[0].bt -= tq
  temp = p[0]                     for(i = 1; i <
queue; i++)                               p[i - 1] =
p[i]
                                  p[queue - 1] = temp
            if(queue)
  clock += timeTaken          else
  clock = p[queue].at
end
```

PROGRAM CODE

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

typedef struct
{   int at;   int bt;
int wt, tat;   int pid,
priority;
}process;

 //Accept sorted by arrival Time

void FCFSinput(process p[], int n)
{
  int i, j;
  process temp;
  printf("\nEnter Process Arrival Time\n");   for(i=0;i<n;i++)
  {
      printf("P[%d]:",i+1);
scanf("%d",&p[i].at);       p[i].pid=i+1;
  }
  printf("\nEnter Process Burst Time\n");   for(i=0;i<n;i++)
  {
    printf("P[%d]:",i+1);
    scanf("%d",&p[i].bt);
  }
  for(i=1; i<n; i++)
  {       temp = p[i];
      j= i-1;
      while(j>=0 && temp.at < p[j].at)
      {
         p[j+1] = p[j];
j = j-1;
      }
      p[j+1] = temp;
  }
}

//Accept sorted by burst Time and internally by arrival Time (non-preemptive SJF)

void SJFinput(process p[], int n)
{
  int i, j;
  process temp;
  printf("\nEnter Process details :-");   for(i = 0; i
< n; i++)
  {
    temp.pid = i+1;     printf("\nProcess
[%d]", i+1);     printf("\nArrival Time: ");
scanf("%d", &temp.at);     printf("Burst
Time: ");     scanf("%d", &temp.bt);
    for(j = i-1; j >= 0 && (temp.bt < p[j].bt || (temp.bt == p[j].bt && temp.at < p[j].at)); j--)     p[j+1] = p[j];
    p[j+1] = temp;
  }
}
```

```
//Accept sorted by Priority and internally by arrival Time (non-preemptive Priority)

void Priorityinput(process p[], int n)
{
  int i, j;
  process temp;
  printf("\nEnter Process details:\n(Low Priority value implies Higher priority)");   for(i = 0; i < n; i++)
  {
    temp.pid = i+1;     printf("\nProcess
[%d]", i+1);     printf("\nArrival Time:
");     scanf("%d", &temp.at);
printf("Burst Time: ");     scanf("%d",
&temp.bt);     printf("Priority: ");
scanf("%d", &temp.priority);
    for(j = i-1; j >= 0 && (temp.priority < p[j].priority || (temp.priority == p[j].priority
&&temp.at < p[j].at)); j--)
    p[j+1] = p[j];
    p[j+1] = temp;
  }
}

void displaytable1(process p[], int n)
{
  int i;
  printf("\nProcess\t\tArrival Time\tBurst Time \tWaiting Time\tTurnaround Time");   for (int i = 0 ; i < n
; i++)
  {
     printf("\nP[%d]\t\t%d\t\t%d\t\t%d\t\t%d",p[i].pid,p[i].at,p[i].bt,p[i].wt,p[i].tat);   }
}

void displaytable2(process p[], int n)
{
  int i;
  printf("\nProcess\t\tArrival Time\tBurst Time\tPriority \tWaiting Time\tTurnaround Time");   for (int i = 0 ; i <
n ; i++)
  {
printf("\nP[%d]\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d",p[i].pid,p[i].at,p[i].bt,p[i].priority,p[i].wt,p[i]. tat);   }
}

//printing average waiting and turnaround time

void printaverage(process p[], int n)
{
  int i;
  float avgwt = 0, avgtat = 0;   for(i = 0;
i < n; i++)
  {
    avgwt += p[i].wt;     avgtat
+= p[i].tat;
  }   avgwt /=
n;   avgtat /= n;
  printf("\n\nAverage Waiting Time: %f\nAverage Turnaround Time: %f\n", avgwt, avgtat); }

//clock represents current time of the system

void FCFS(process p[], int n)
```

```
{   int clock, i;   clock =
p[0].at;   for(i = 0; i < n;
i++)
  {
    p[i].wt = clock - p[i].at;     p[i].tat =
p[i].wt + p[i].bt;
    if(i < n - 1 && clock + p[i].bt < p[i + 1].at)
      clock = p[i + 1].at;  //taking idle time into consideration     else
      clock += p[i].bt;
  }
}

void SJF(process p[], int n)
{
  int selected_process, clock, i, j;
  process temp;   clock =
0;   for(i = 0; i < n; i++)
  {
    selected_process = -1;
    for(j = i; j < n; j++) //Searching for a Ready process of smallest burst time     if(p[j].at <= clock)
      {
        selected_process = j;
        break;
      }
    if(selected_process == -1) //Search failed
      {
        selected_process = i;
        for(j = i+1; j < n; j++) //Searching for the earliest Available process         if(p[j].at
< p[selected_process].at)          selected_process = j;
        clock = p[selected_process].at;
      }
    p[selected_process].wt = clock - p[selected_process].at;
    p[selected_process].tat = p[selected_process].wt + p[selected_process].bt;     clock +=
p[selected_process].bt;     temp = p[selected_process];
    for(j = selected_process - 1; j >= i; j--) //moving the completed process       p[j + 1] =
p[j];     p[i] = temp;
  }
}

void PrioritySchedule(process p[], int n)
{
  SJF(p, n); //only change is in condition of ordering the set during the input }

void RRS(process p[], int n, int tq)
{
  int i, clock, timeTaken;   int
PC, queue, bt[20];   process
temp;   for(i = 0; i < n; i++)
bt[i] = p[i].bt;   clock =
p[0].at;   PC = n;   queue = 1;
  while(PC)
  {
    if(p[0].bt <= tq)
      {
        timeTaken = p[0].bt;
        p[0].tat = clock + timeTaken - p[0].at;
        p[0].wt = p[0].tat - bt[p[0].pid - 1];       }
```

```
   else
     timeTaken = tq;     //Set no of
ready processes
   while(queue < PC && clock + timeTaken >= p[queue].at)       queue++;
   //Remove completed process
   if(timeTaken - p[0].bt == 0)
   {       PC--;
queue--;
p[0].bt = 0;
     //Insert p[0] at p[PC]
temp = p[0];       for(i = 1; i <=
PC; i++)         p[i - 1] = p[i];
     p[PC] = temp;
   }
   else //Move p[0] to end of Queue
   {
     p[0].bt -= tq;
     //Insert p[0] at p[queue - 1]
temp = p[0];       for(i = 1; i < queue;
i++)       p[i - 1] = p[i];
     p[queue - 1] = temp;
   }     if(queue)       clock +=
timeTaken;     else
     clock = p[queue].at;
  }
}

void main()
{
  int option, n, tq, i;   process
p[20];
  while(1)
  {
    printf("------MENU------:\n1.FCFS\n2.SJF\n3.Priority\n4.Round Robin\n5.Exit");
printf("\nEnter the option: ");     scanf("%d", &option);
   while(option < 1 || option > 5)
   {
     printf("Invalid option");
     break;
   }
   if(option == 1) //FCFS
   {
     printf("Enter the number of processes: ");
     scanf("%d", &n);
FCFSinput(p, n);       FCFS(p, n);
displaytable1(p, n);
     printaverage(p, n);
   }
   else if(option == 2) //SJF
   {
     printf("Enter the number of processes: ");
     scanf("%d", &n);
SJFinput(p, n);       SJF(p, n);
displaytable1(p, n);
     printaverage(p, n);
   }
   else if(option == 3) // Priority Scheduling
```

```
   {
     printf("Enter the number of processes: ");
scanf("%d", &n);       Priorityinput(p, n);
PrioritySchedule(p, n);       displaytable2(p, n);
     printaverage(p, n);
   }
   else if(option == 4) // Round Robin Scheduling
   {
     printf("Enter the number of processes: ");
scanf("%d", &n);       printf("Enter the Time
Quantum: ");
     scanf("%d", &tq);
FCFSinput(p, n);       RRS(p, n,
tq);
     printaverage(p, n);
   }     else
break;
printf("\n");
  }
}
```

OUTPUT

```
tom@TomThomas:~/Classwork/lab2$ gedit exp3.c
tom@TomThomas:~/Classwork/lab2$ gcc exp3.c
tom@TomThomas:~/Classwork/lab2$ gcc exp3.c
tom@TomThomas:~/Classwork/lab2$ ./a.out
------MENU------:
1.FCFS
2.SJF
3.Priority
4.Round Robin
5.Exit
Enter the option: 1
Enter the number of processes: 5

Enter Process Arrival Time
P[1]:0
P[2]:2
P[3]:2
P[4]:1
P[5]:3

Enter Process Burst Time
P[1]:8
P[2]:6
P[3]:1
P[4]:9
P[5]:3

Process         Arrival Time    Burst Time      Waiting Time    Turnaround Time
P[1]            0               8               0               8
P[4]            1               9               7               16
P[2]            2               6               15              21
P[3]            2               1               21              22
P[5]            3               3               21              24

Average Waiting Time: 12.800000
Average Turnaround Time: 18.200001
```

```
------MENU------:
1.FCFS
2.SJF
3.Priority
4.Round Robin
5.Exit
Enter the option: 2
Enter the number of processes: 4

Enter Process details :-
Process [1]
Arrival Time: 0
Burst Time: 8

Process [2]
Arrival Time: 2
Burst Time: 4

Process [3]
Arrival Time: 4
Burst Time: 9

Process [4]
Arrival Time: 5
Burst Time: 5

Process         Arrival Time    Burst Time      Waiting Time    Turnaround Time
P[1]            0               8               0               8
P[2]            2               4               6               10
P[4]            5               5               7               12
P[3]            4               9               13              22

Average Waiting Time: 6.500000
Average Turnaround Time: 13.000000
```

```
------MENU------:
1.FCFS
2.SJF
3.Priority
4.Round Robin
5.Exit
Enter the option: 3
Enter the number of processes: 5

Enter Process details:
(Low Priority value implies Higher priority)
Process [1]
Arrival Time: 0
Burst Time: 8
Priority: 4

Process [2]
Arrival Time: 2
Burst Time: 6
Priority: 1

Process [3]
Arrival Time: 2
Burst Time: 1
Priority: 2

Process [4]
Arrival Time: 1
Burst Time: 9
Priority: 2

Process [5]
Arrival Time: 3
Burst Time: 3
Priority: 3

Process         Arrival Time    Burst Time      Priority        Waiting Time    Turnaround Time
P[1]            0               8               4               0               8
P[2]            2               6               1               6               12
P[4]            1               9               2               13              22
P[3]            2               1               2               21              22
P[5]            3               3               3               21              24

Average Waiting Time: 12.200000
Average Turnaround Time: 17.600000
```

```
------MENU------:
1.FCFS
2.SJF
3.Priority
4.Round Robin
5.Exit
Enter the option: 4
Enter the number of processes: 5
Enter the Time Quantum: 4

Enter Process Arrival Time
P[1]:0
P[2]:5
P[3]:9
P[4]:13
P[5]:17

Enter Process Burst Time
P[1]:11
P[2]:13
P[3]:6
P[4]:9
P[5]:12


Average Waiting Time: 19.400000
Average Turnaround Time: 29.600000

------MENU------:
1.FCFS
2.SJF
3.Priority
4.Round Robin
5.Exit
Enter the option: 5
tom@TomThomas:~/Classwork/lab2$
```

RESULT

The program was successfully executed and the desired output was obtained.

EXPERIMENT NO : 02 DATE : 09/12/2021

BANKER'S ALGORITHM

AIM

To write a C program to simulate the banker's algorithm for deadlock avoidance

ALGORITHM

OVERALL STRUCTURE

start
print ("********** Banker's Algorithm **********") input()
show(0) newrequest() show(1)
safestate()
end

calcneed()

calculates the need matrix (max-allocation)

input()

Accept the no of processes & resource, max & allocation matrix and available resources calcneed()

show(int x)

Displays the max, allocation, need matrix if(x==0)
display the available array else if(x==1)
display the work array
end **newrequest()**
print ("Enter process
no of the requesting
process :") read(id)
id=id-1
print ("Enter the request of Resources :") for(j=0;j<r;j++)
read(request[j])
for(j=0;j<r;j++)
alloc[id][j]=alloc[id][j]+request[j]
calcneed()
end

safestate() for(i=0;i<n;i++)
finish[i]=0
for(j=0;j<r;j++) work[j]=avail[j]-request[j]
print ("\Work") for(j=0;j<r;j++)
print (work[j]) print ("The Execution
order is:-") while(flag)

```
            flag=0
             for(i=0;i<n;i++)
                        int c=0
                         for(j=0;j<r;j++)
                                    if((finish[i]==0)&&(need[i][j]<=work[j]))
                                          c++
                         if(c==r)
   for(k=0;k<r;k++)
   work[k]+=alloc[i][j]
   finish[i]=1                                          flag=1
                         print ("P->",i+1)
   if(finish[i]==1)
   i=n          for(i=0;i<n;i++)
                         if(finish[i]==1)
                                    c1++
                         else
   print ("P->",i+1)
              if(c1==n)
                print (" The system is in safe state"              else
                print ("Process are in dead lock")
   print (" System is in unsafe state")
end
```

PROGRAM CODE

```
#include<stdio.h>
int max[20][20]; int
alloc[20][20]; int
need[20][20]; int
avail[20]; int
request[20];
int n,r;

       //to calculate the need matrix
void calcneed() {
  int i,j;
  for(i=0;i<n;i++)
  {     for(j=0;j<r;j++)
    {
      need[i][j]=max[i][j]-alloc[i][j];
    } } }
//to input the details
void input() {
  int i,j;
  printf("\nEnter the no of Processes :");   scanf("%d",&n);
  printf("\nEnter the no of resources instances :");
  scanf("%d",&r);   printf("\nEnter the Max
Matrix\n");   for(i=0;i<n;i++)
  {     printf("P[%d] :",i+1);
    for(j=0;j<r;j++)
    {
      scanf("%d",&max[i][j]);
    }
  }
```

```
  printf("\nEnter the Allocation Matrix\n");   for(i=0;i<n;i++)
  {     printf("P[%d] :",i+1);
    for(j=0;j<r;j++)
    {
      scanf("%d",&alloc[i][j]);
    }
  }
  printf("\nEnter the Available no of Resources :");   for(j=0;j<r;j++)
  {
    scanf("%d",&avail[j]);
  }   calcneed();
}

//to display the details
void show(int x)
{
  int i,j;
  printf("\nProcess\t Allocation\t Max\t\t Need\t");   for(i=0;i<n;i++)
  {
    printf("\nP[%d]\t ",i+1);
    for(j=0;j<r;j++)
    {
      printf("%d ",alloc[i][j]);
    }     printf("\t\t");
    for(j=0;j<r;j++)
    {
      printf("%d ",max[i][j]);
    }     printf("\t\t");
    for(j=0;j<r;j++)
    {
      printf("%d ",need[i][j]);
    }
  }   if(x==0)
{
    printf("\n\nAvailable\n");
for(j=0;j<r;j++)       printf("%d
",avail[j]);
  }
}
void newrequest()
{
  int id,j;
  printf("\nEnter process no of the requesting process :");
  scanf("%d",&id);   id=id-1;
  printf("\nEnter the request of Resources :");   for(j=0;j<r;j++)
  {
    scanf("%d",&request[j]);
  }   for(j=0;j<r;j++)
{
    alloc[id][j]=alloc[id][j]+request[j];
  }   calcneed();

}

void safestate() {
  int finish[20],temp,flag=1,k,c1=0;
  int i,j;
```

```
 int work[20];
for(i=0;i<n;i++)  {
finish[i]=0;  }
for(j=0;j<r;j++)  {
   work[j]=avail[j]-request[j];
 }
 printf("\n\nWork\n");     for(j=0;j<r;j++)
printf("%d ",work[j]);  printf("\n\nThe
Execution order is:-\n");
 printf("\n");
while(flag)  {
flag=0;
for(i=0;i<n;i++)  {
int c=0;
   for(j=0;j<r;j++)
   {
    if((finish[i]==0)&&(need[i][j]<=work[j]))
    {        c++;
if(c==r)
      {
        for(k=0;k<r;k++)
        {
          work[k]+=alloc[i][j];
finish[i]=1;            flag=1;
        }
        printf("P%d->",i+1);
        if(finish[i]==1)
        {
i=n;
        } } } } } }
 for(i=0;i<n;i++)
 {
   if(finish[i]==1)
   {
c1++;    }
else
   {
    printf("P%d->",i+1);
   }
 }   if(c1==n)
   printf("\n The system is in safe state\n");
   else
{
   printf("\n Process are in dead lock");     printf("\n
System is in unsafe state\n");
 }
}

int main() {
   printf("\n********** Banker's Algorithm ************\n");
   input();
   show(0);
   newrequest();
   show(1);
   safestate();
   return 0;
}
```

OUTPUT

```
tom@TomThomas:~/Classwork/lab2$ gedit exp4.c
tom@TomThomas:~/Classwork/lab2$ gcc exp4.c
tom@TomThomas:~/Classwork/lab2$ ./a.out

********** Banker's Algorithm ************

Enter the no of Processes :5

Enter the no of resources instances :3

Enter the Max Matrix
P[1] :7 5 3
P[2] :3 2 2
P[3] :9 0 2
P[4] :2 2 2
P[5] :4 3 3

Enter the Allocation Matrix
P[1] :0 1 0
P[2] :2 0 0
P[3] :3 0 2
P[4] :2 1 1
P[5] :0 0 2
```

```
Enter the Available no of Resources :3 3 2

Process  Allocation      Max             Need
P[1]     0 1 0         7 5 3          7 4 3
P[2]     2 0 0         3 2 2          1 2 2
P[3]     3 0 2         9 0 2          6 0 0
P[4]     2 1 1         2 2 2          0 1 1
P[5]     0 0 2         4 3 3          4 3 1

Available
3 3 2
Enter process no of the requesting process :2

Enter the request of Resources :1 0 2
```

```
Process  Allocation      Max             Need
P[1]     0 1 0         7 5 3           7 4 3
P[2]     3 0 2         3 2 2           0 2 0
P[3]     3 0 2         9 0 2           6 0 0
P[4]     2 1 1         2 2 2           0 1 1
P[5]     0 0 2         4 3 3           4 3 1

Work
2 3 0

The Execution order is:-

P2->P4->P5->P1->P3->
 The system is in safe state
tom@TomThomas:~/Classwork/lab2$
```

RESULT

The program was successfully executed and the desired output was obtained.

EXPERIMENT NO : 03 DATE : 16/12/2021

DISK SCHEDULING ALGORITHMS

AIM

To Write a C program to simulate the following disk scheduling algorithms a) FCFS
b) SCAN
c) C-SCAN

ALGORITHM

OVERALL STRUCTURE

```
start
print ("Enter the maximum number of cylinders : ") read (n)
print ("Enter the number of disk queue elements :") read (qsize)
print ("Enter the disk queue elements : ") for(i=0;i<qsize;i++)
                read (diskqueue[i])
               visit[i] = 0
print ("Enter the disk start starting posision : ") read (start)
while(1)
   print ("-----MENU------ ")              print
("1. FCFS ")          print ("2. SCAN ")
   print ("3. C-SCAN ")        print ("4.
EXIT ")    print ("Enter your choice: ")
   read (ch)             switch(ch)
                           case 1: print ("FCFS disk scheduling ")
                          fcfs(qsize,diskqueue,start)
break                     case 2: print ("SCAN disk scheduling ")
                           scan(qsize,diskqueue,start,n)
    break
                          case 3: print ("C-SCAN disk scheduling ")
CSCAN(qsize,diskqueue,start,n)
                           break
   case 4: exit(0) stop
```

FCFS

```
void fcfs(int qsize, int diskqueue[25], int st)
s=0 for(i=0;i<qsize;i++)
   s=s+abs(st-diskqueue[i])
   st=diskqueue[i]
print ("Disk traversal order")
for(i=0;i<qsize;i++)             print
(diskqueue[i]) print ("Total seek
time ",s) stop
```

SCAN

```
void scan(int qsize, int diskqueue[25], int st, int size) s = 0
sort(qsize, diskqueue) dir=direction() print ("Disk traversal
order ") for (i = 0; i < qsize; i++)
  if (st < diskqueue[i])                    index = i
  break if(dir == 1)            for (i = index; i < qsize;
i++)                    s = s + abs(diskqueue[i] - st)
  st = diskqueue[i]             print (diskqueue[i])
  s = s + abs(size - diskqueue[i - 1] - 1)
      st = size - 1             for (i = index - 1; i >=
0; i--)                 s = s + abs(diskqueue[i] - st)
  st = diskqueue[i]             print (diskqueue[i])
else if(dir == 2)     for (i = index -1 ; i >= 0; i--)
            s = s + abs(diskqueue[i] - st)
  st = diskqueue[i]             print (diskqueue[i])
s = s + abs(0 - diskqueue[i + 1])       st = 0
      for (i = index ; i < qsize; i++)
            s = s + abs(diskqueue[i] - st)
  st = diskqueue[i]             print (diskqueue[i])
print ("Total seek time : , s) stop
```

C-SCAN

```
void CSCAN(int qsize, int diskqueue[25], int st, int size) s = 0
sort(qsize, diskqueue) dir = direction() print ("Disk traversal order
") for (i = 0; i < qsize; i++)
  if (st < diskqueue[i])
  index = i           break
if(dir==1)
  for (i = index; i < qsize; i++)                   s = s +
abs(diskqueue[i] - st)                  st =
diskqueue[i]                  print (diskqueue[i])
s = s + abs(size - diskqueue[i - 1] - 1)       s = s +
abs(size - 1 )       st = 0       for (i = 0; i < index;
i++)                  s = s + abs(diskqueue[i] - st)
  st = diskqueue[i]             print (diskqueue[i]) else
if(dir==2)            for (i = index - 1; i >= 0; i--)
  s = s + abs(diskqueue[i] - st)                    st =
diskqueue[i]                  print (diskqueue[i])
s = s + abs(0 - diskqueue[i + 1])       s = s + abs(size
- 1 )       st = size-1       for (i=qsize-1;i>=index;i--
)                       s = s + abs(diskqueue[i] - st)
  st = diskqueue[i]             print (diskqueue[i])
printf("Total seek time :", s)
Stop
```

PROGRAM

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

int direction() {
int opt;
   printf("\n****Direction of traversal****");
printf("\n1.Increasing oreder");
printf("\n2.Decreasing order");
printf("\nEnter the direction of traversal : ");
   scanf("%d",&opt);
return opt;
}

void sort(int qsize, int diskqueue[25])
{
   int t, i, j;
for (i = 0; i < qsize; i++)
for (j = i + 1; j < qsize; j++)
          if (diskqueue[i] > diskqueue[j])
          {
t = diskqueue[i];
diskqueue[i] = diskqueue[j];
diskqueue[j] = t;
          }
}

void fcfs(int qsize, int diskqueue[25], int st)
{     int i, s = 0;
for (i = 0; i < qsize; i++)
   {
      s = s + abs(st - diskqueue[i]); // taking absolute value
st = diskqueue[i];
   }
   printf("\nDisk traversal order\n");
for (i = 0; i < qsize; i++)
printf(" %d ", diskqueue[i]);
printf("\n Total seek time :%d", s);
}

void scan(int qsize, int diskqueue[25], int st, int size)
{
   int i,dir, j, s = 0, index;
sort(qsize, diskqueue);
dir=direction();
printf("\nDisk traversal order \n");
for (i = 0; i < qsize; i++)
   {
      if (st < diskqueue[i])
      {
index = i;
break;
      }     }
if(dir == 1)
```

```
    {
        // movement is towards high value
        for (i = index; i < qsize; i++)
        {
            s = s + abs(diskqueue[i] - st);
 st = diskqueue[i];
            printf("%d ", diskqueue[i]);
        }
        // last movement for max size
s = s + abs(size - diskqueue[i - 1] - 1);
        st = size - 1;
        for (i = index - 1; i >= 0; i--)
        {
            s = s + abs(diskqueue[i] - st);
st = diskqueue[i];
            printf("%d ", diskqueue[i]);
        }
}
    else if(dir == 2)
    {
        for (i = index -1 ; i >= 0; i--)
        {
            s = s + abs(diskqueue[i] - st);
st = diskqueue[i];
            printf("%d ", diskqueue[i]);
        }
        // last movement for smallest size
        s = s + abs(0 - diskqueue[i + 1]);
st = 0;
        for (i = index ; i < qsize; i++)
        {
            s = s + abs(diskqueue[i] - st);
st = diskqueue[i];
            printf("%d ", diskqueue[i]);
        }
    }
    printf("\n Total seek time : %d", s);
}

void CSCAN(int qsize, int diskqueue[25], int st, int size) {
    int i, j, s = 0, index,dir;     sort(qsize,
diskqueue);     dir = direction();
printf("\nDisk traversal order \n");
for (i = 0; i < qsize; i++)
    {
        if (st < diskqueue[i])
        {
index= i;
break;
        }     }
if(dir==1)
    {
        // movement is towards high value
        for (i = index; i < qsize; i++)
        {
            s = s + abs(diskqueue[i] - st);
                st = diskqueue[i];
```

```
            printf("%d ", diskqueue[i]);
        }
        // last movement for max size
 s = s + abs(size - diskqueue[i - 1] - 1);
/*movement max to min disk */
s = s + abs(size - 1 );
st = 0;
        for (i = 0; i < index; i++)
        {
            s = s + abs(diskqueue[i] - st);
st = diskqueue[i];
            printf("%d ", diskqueue[i]);
        }
    }     else if(dir==2)     {
for (i = index - 1; i >= 0; i--)
        {
            s = s + abs(diskqueue[i] - st);
st = diskqueue[i];
            printf("%d ", diskqueue[i]);
        }
        // last movement for smallest size
s = s + abs(0 - diskqueue[i + 1]);
/*movement min to max disk */
s = s + abs(size - 1 );         st = size-1;
for (i=qsize-1;i>=index;i--)
        {
            s = s + abs(diskqueue[i] - st);
st = diskqueue[i];
            printf("%d ", diskqueue[i]);
        }
    }
    printf("\n Total seek time : %d", s);
}

int main() {
    int n, diskqueue[25], start, i, j, qsize, ch;
printf("\n Enter the maximum number of cylinders : ");
scanf("%d", &n);
    printf("\n Enter the number of disk queue elements :");
scanf("%d", &qsize);
    printf("\n Enter the disk queue elements : ");
    for (i = 0; i < qsize; i++)
    {
        scanf("%d", &diskqueue[i]);
    }
    printf("\n Enter the disk start starting posision : ");
scanf("%d", &start);
    while (1)
    {
        printf("\n\n-----MENU------ ");
printf("\n 1. FCFS ");
printf("\n 2. SCAN ");
printf("\n 3. C-SCAN ");
printf("\n 4. EXIT ");
printf("\nEnter your choice: ");
scanf("%d", &ch);
```

```
        switch (ch)
{         case 1:
           printf("\n FCFS disk scheduling \n");
fcfs(qsize, diskqueue, start);
           break;
case 2:
           printf("\n SCAN disk scheduling \n");
scan(qsize, diskqueue, start, n);
           break;
case 3:
           printf("\n C-SCAN disk scheduling \n");
CSCAN(qsize, diskqueue, start, n);
break;
case 4:
exit(0);
        }
    }
}
```

OUTPUT

```
 Enter the maximum number of cylinders : 200

 Enter the number of disk queue elements :8

 Enter the disk queue elements : 98 183 37 122 14 124 65 67

 Enter the disk start starting posision : 53


-----MENU------
 1. FCFS
 2. SCAN
 3. C-SCAN
 4. EXIT
Enter your choice: 1

 FCFS disk scheduling

Disk traversal order
 98  183  37  122  14  124  65  67
 Total seek time :640
```

```
 -----MENU------
  1. FCFS
  2. SCAN
  3. C-SCAN
  4. EXIT
 Enter your choice: 2

  SCAN disk scheduling

 ****Direction of traversal****
 1.Increasing order
 2.Decreasing order
 Enter the direction of traversal : 1

 Disk traversal order
 65 67 98 122 124 183 37 14
  Total seek time : 331
```

```
 -----MENU------
  1. FCFS
  2. SCAN
  3. C-SCAN
  4. EXIT
 Enter your choice: 3

  C-SCAN disk scheduling

 ****Direction of traversal****
 1.Increasing order
 2.Decreasing order
 Enter the direction of traversal : 1

 Disk traversal order
 65 67 98 122 124 183 14 37
  Total seek time : 382

 -----MENU------
  1. FCFS
  2. SCAN
  3. C-SCAN
  4. EXIT
 Enter your choice: 4
```

RESULT

The program was successfully executed and the desired output was obtained.

SS EXPERIMENTS

EXPERIMENT NO: 04 DATE:24/01/2022

PASS 1 ALGORITHM

AIM

To implement pass 1 of 2 pass assembler

INPUT

Source file in Assembly Language

OUTPUT

Intermediate file, Symbol table, Program length

ALGORITHM

```
Begin
Read input line
if OPCODE = ‘START’, then
        Set Starting Address as #[Operand]
        Initialize LOCCTR to Starting Address
        Write line to Intermediate file
        Read next line
else
        Initialize LOCCTR to 0
        Set Starting Address to 0

while OPCODE != ‘END’, do
  if line is not a comment, then
                if there is a symbol in the LABEL field, then
                  Search SYMTAB for LABEL
           if found, then
                                Set error flag(duplicate symbol)
                        else
                                Add symbol to SYMTAB with it’s address
                  end if
  end if

          Search OPTAB for OPCODE
  if found, then
                        Add 3 to LOCCTR // 3 = Instruction length
                else if OPCODE = ‘WORD’, then
```

```
                    Add 3 to LOCCTR
                else if OPCODE = ‘RESW’, then
                    Add 3 * #[Operand] to LOCCTR
                else if OPCODE = ‘RESB’, then
                  Add #[Operand] to LOCCTR
  else if OPCODE = ‘BYTE’, then
  Find length of constant in bytes
  Add length to LOCCTR
                else
                    Set error flag(invalid Operation Code)
                end if
        end if

  Write line to Intermediate file
  Read next input line end while

Write last line to Intermediate file
Save (LOCCTR - Starting Address) as Program Length

End Pass 1
```

PASS 2 ALGORITHM

AIM

To implement pass 2 of 2 pass assembler.

INPUT

Intermediate file from Pass 1, Program length, Symbol table

OUTPUT

Object program, Assembly listing

ALGORITHM

```
Begin
Read first input line(from intermediate file)
if OPCODE = 'START', then          Write
Listing line       Read next input line
end if

Write Header record to object program
Initialize first Text record

While OPCODE != 'END', do
  if line is comment line, then
                Read next input line
                continue
          end if

          Search OPTAB for OPCODE
          if found then
                if there is a symbol in OPERAND field then
                  Search SYMTAB for OPERAND
            if found, then
                                  Search symbol value as operand address
                        else
                          Store 0 as operand address
                  Set error flag (undefined symbol)
                        end if
                else
   Store 0 as operand address    end if
            Assemble object code instruction
  else if OPCODE = 'BYTE' or 'WORD', then
```

```
  Convert constant to object code else if
OPCODE = ‘RESB’ or ‘RESW’, then
                if current Text record is not empty, then
                        Write Text record to object program
                end if

                Write Listing line
          Read next input line
  Initialize new Text record
        end if

        if object code will not fit into the current Text record, then
          Write Text record to object program
  Initialize new Text record
        end if

        Add object code to Text record
  Write Listing line      Read
next input line end while

Write last Text record to object program
Write End record to object program
Write last Listing line

End Pass 2
```

PROGRAM

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

#define DIRLEN 81 // Including '\0'
#define ADRLEN 5 // Including '\0'
#define CONSTLEN 17 // Including '\0'
#define LABLEN 7 // Including '\0'. Excluding ':'
#define MNEMLEN 6 // Including '\0'
#define OPLEN 12 // Including '\0'
#define SROWS 25
#define OROWS 10
#define TEXTLEN 70 // Including '\0'

typedef struct
{
   char label[LABLEN];
   int address;
}STAB_Row;

typedef struct
{    int size;
   STAB_Row row[SROWS];
}STAB;

typedef struct
{
   char mnemonic[MNEMLEN];    int
hexacode;
}OPTAB_Row;

typedef struct
{    int size;
   OPTAB_Row row[OROWS];
}OPTAB;

void getLine(char str[], int size) // Accept a line of upto given size
{
   char ch = 0;    int
i = 0;

   while((ch = getchar())!='\n')
      if(i < size)
str[i++] = ch;    str[i] = 0;
}
int strToDec(const char str[OPLEN]) // Returns the Decimal of a Decimal string
{
   int i, num = 0;

   for(i = 0; str[i]; i++)
      num = num*10 + str[i] - '0';

   return num;
}

int hexAToDec(const char hexstr[ADRLEN]) // Returns the Decimal of a
```

```
Hexadecimal string
{
   int decimal = 0, i;

   if(hexstr[0] == 0)
      return -1; // hexstr = "\0"

   for(i = 0; i < ADRLEN -  1; i++)
   {
      decimal *= 16;
      if(hexstr[i] >= '0' && hexstr[i] <= '9')            decimal
+= hexstr[i] - '0';
      else if(hexstr[i] >= 'A' && hexstr[i] <= 'F')
decimal += hexstr[i] - 'A' + 10;         else
         return -2; // hexstr does not represent a valid Hexadecimal number
   }

   return decimal;
}

void decToHexA(char hex[ADRLEN], const int decimal) // Sets hex to a Hexadecimal string of
Decimal
{
   int i, digit, dec = decimal;
   for(i = 0; i < ADRLEN - 1; i++)         hex[i] =
'0';
   hex[i] = 0;

   if(decimal < 0)
      hex[0] = 0;

   for(i = ADRLEN - 2; decimal && i >= 0; i--)
   {
      digit = dec % 16;
dec /= 16;         if(digit < 10)
hex[i] = digit + '0';         else
         hex[i] = digit - 10 + 'A';
   }
}

void decToHex6(char hex[7], const int decimal) // Sets hex to a Hexadecimal string of Decimal
{
   int i, digit, dec = decimal;
   strcpy(hex, "000000");

   if(decimal < 0)
      hex[0] = 0;

   for(i = 5; decimal && i >= 0; i--)
   {
      digit = dec % 16;
dec /= 16;         if(digit < 10)
hex[i] = digit + '0';         else
         hex[i] = digit - 10 + 'A';
   }
}

int constBLen(char str[]) // Returns length of constants of the form C'...' and X'...'in bytes
```

```
{
   int i, len = 0;

   if(str[0] == 'C' || str[0] == 'c')
   {
      for(i = 2; str[i] != '\''; i++, len++);         return
len;
   }
   else if(str[0] == 'X' || str[0] == 'x')
   {
      for(i = 2; str[i] != '\''; i++, len++);
      return (len + 1)/2;
   }
   return -1;
}

void constToHex(char hex[CONSTLEN], const char operand[OPLEN]) // Sets hex to a Hexidecimal
string  of Constant Equivalent
{    int i, j;
   char temp[CONSTLEN];
   hex[0] = 0;

    if(operand[0] == 'X' && operand[1] == '\'')
   {
      for(i = 0; operand[i + 2] != '\'' && i < (CONSTLEN - 1)/2; i++)            hex[i] =
operand[i + 2];
      hex[i] = 0;
if(i % 2)
      {
         strcpy(temp, hex);            strcpy(hex,
"0");
         strcat(hex, temp);
      }
   }
   else if(operand[0] == 'C' && operand[1] == '\'')
   {
      for(i = 0, j = 2; operand[j] != '\'' && i < 16; j++, i+=2)
      {
         hex[i + 1] = operand[j]%16;
if(hex[i + 1] > 9)               hex[i + 1] +=
'A' - 10;
         else
            hex[i + 1] += '0';

         hex[i] = operand[j]/16;
if(hex[i] > 9)               hex[i] += 'A'
- 10;            else
            hex[i] += '0';
      }
      hex[i] = 0;
   }
}

void parseToken(char token[], char str[], int size) // Parses a token of upto size specified from string
str. Uses delimiters ' ', '\0', ':', '\n'
{    int i;

   for(i = 0; i < size && str[i] && str[i] != ' ' && str[i] != ':' && str[i] != '\n'; i++)         token[i] = str[i];
```

```
    token[i] = 0;
}

int searchSTAB(STAB stab, char label[LABLEN]) // Returns index of row if found, -1 otherwise
{     int i;

    for(i = 0; i < stab.size; i++)
if(!strcmp(stab.row[i].label, label))             return i;
     return -1;
}

int insertSTAB(STAB *stab, char label[LABLEN], int address) // Returns 0 if overflow occurs, 1 on
success
{
    if(stab->size >= SROWS)
        return 0;

    stab->row[stab->size].address = address;     strcpy(stab->row[stab-
>size].label, label);     stab->size++;

    return 1;
}

int searchOPTAB(OPTAB optab, char mnemonic[MNEMLEN]) // Returns index of row if found, -1
otherwise
{     int i;

    for(i = 0; i < optab.size; i++)
        if(!strcmp(optab.row[i].mnemonic,                  mnemonic))
return i;     return -1;
}

void setOPTAB(OPTAB *optab) // Sets optab to Predefined OPTAB values
{
    optab->size = 25;

    strcpy(optab->row[0].mnemonic, "LDA");     optab->row[0].hexacode = 0;

    strcpy(optab->row[1].mnemonic, "LDX");     optab->row[1].hexacode = 4;

    strcpy(optab->row[2].mnemonic, "LDL");     optab->row[2].hexacode = 8;

    strcpy(optab->row[3].mnemonic, "STA");     optab->row[3].hexacode = 12;

    strcpy(optab->row[4].mnemonic, "STX");     optab->row[4].hexacode = 16;

    strcpy(optab->row[5].mnemonic, "STL");     optab->row[5].hexacode = 20;

    strcpy(optab->row[6].mnemonic, "ADD");
     optab->row[6].hexacode = 24;

    strcpy(optab->row[7].mnemonic, "SUB");
    optab->row[7].hexacode = 28;

    strcpy(optab->row[8].mnemonic, "MUL");
    optab->row[8].hexacode = 32;

    strcpy(optab->row[9].mnemonic, "DIV");
```

```
    optab->row[9].hexacode = 36;

    strcpy(optab->row[10].mnemonic, "COMP");
    optab->row[10].hexacode = 40;

    strcpy(optab->row[11].mnemonic, "TIX");
    optab->row[11].hexacode = 44;

    strcpy(optab->row[12].mnemonic, "JEQ");
    optab->row[12].hexacode = 48;

    strcpy(optab->row[13].mnemonic, "JGT");
    optab->row[13].hexacode = 52;

    strcpy(optab->row[14].mnemonic, "JLT");
    optab->row[14].hexacode = 56;

    strcpy(optab->row[15].mnemonic, "J");
    optab->row[15].hexacode = 60;

    strcpy(optab->row[16].mnemonic, "AND");
    optab->row[16].hexacode = 64;

    strcpy(optab->row[17].mnemonic, "OR");
    optab->row[17].hexacode = 68;

    strcpy(optab->row[18].mnemonic, "JSUB");
    optab->row[18].hexacode = 72;

    strcpy(optab->row[19].mnemonic, "RSUB");
    optab->row[19].hexacode = 76;

    strcpy(optab->row[20].mnemonic, "LDCH");
    optab->row[20].hexacode = 80;

    strcpy(optab->row[21].mnemonic, "STCH");
    optab->row[21].hexacode = 84;

    strcpy(optab->row[22].mnemonic, "RD");
    optab->row[22].hexacode = 216; strcpy(optab->row[23].mnemonic, "WD");
    optab->row[23].hexacode = 220;

    strcpy(optab->row[24].mnemonic, "TD");
    optab->row[24].hexacode = 224;
}

int indexedMode(char operand[OPLEN]) // Returns 1 if given operand is in Indexed Addressing Mode,
0 otherwise
{     int i;

    if(!operand[0])
 return 0;
    for(i = 0; operand[i + 1]; i++)
    {
        if(operand[i] == ',' && operand[i + 1] == 'X')            return 1;
    }

    return 0;
```

```
}

int pass1(char fileloc[DIRLEN], STAB *stab, OPTAB optab)
{
   FILE *pgmptr, *intrptr;
   char intrloc[DIRLEN + 5], label[LABLEN], mnemonic[MNEMLEN], operand[OPLEN];
   char inLine[LABLEN + MNEMLEN + OPLEN + 1], address[ADRLEN], firstaddr[OPLEN];
   int i, startaddr = 0, locctr = 0, addrshift;     int
found;

   pgmptr = fopen(fileloc, "r");

   if(pgmptr == NULL)
      return -1; // Program file not found     else
if(feof(pgmptr))
      return -2; // Program file is empty

   strcpy(intrloc, fileloc);
   strcat(intrloc, ".intr");

   intrptr = fopen(intrloc, "w");

   stab->size = 0;
   firstaddr[0] = 0;

   fgets(inLine, LABLEN + MNEMLEN + OPLEN + 1, pgmptr);     fgetc(pgmptr);
    parseToken(label, inLine, LABLEN);
   parseToken(mnemonic, inLine + LABLEN + 1, MNEMLEN);     parseToken(operand, inLine +
LABLEN + MNEMLEN + 1, OPLEN);

   if(!strcmp(mnemonic, "START"))
   {
      startaddr = locctr = hexAToDec(operand);         fprintf(intrptr, "
%s\n", inLine);

      fgets(inLine, LABLEN + MNEMLEN + OPLEN + 1, pgmptr);         fgetc(pgmptr);
      parseToken(label, inLine, LABLEN);         parseToken(mnemonic, inLine +
LABLEN + 1, MNEMLEN);         parseToken(operand, inLine + LABLEN + MNEMLEN
+ 1, OPLEN);
   }

   while(strcmp(mnemonic, "END"))
   {
      if(label[0] == '/' && label[1] == '/') // If line is a comment
      {
         fgets(inLine, LABLEN + MNEMLEN + OPLEN + 1, pgmptr);             fgetc(pgmptr);
         parseToken(label, inLine, LABLEN);             parseToken(mnemonic,
inLine + LABLEN + 1, MNEMLEN);
         parseToken(operand, inLine + LABLEN + MNEMLEN + 1, OPLEN);             continue;
      }

      if(label[0]) // If there is a Sybmol in Label column
      {
         if(searchSTAB(*stab, label) > -1)
return -3;
         insertSTAB(stab, label, locctr);             stab->size++;
      }
```

```
        //Calculating next LOCCTR value
        if(!strcmp(mnemonic, "WORD"))            addrshift
= 3;
        else if(!strcmp(mnemonic, "RESW"))            addrshift =
3*strToDec(operand);         else if(!strcmp(mnemonic,
"RESB"))            addrshift = strToDec(operand);         else
if(!strcmp(mnemonic, "BYTE"))            addrshift =
constBLen(operand);         else if(searchOPTAB(optab,
mnemonic) == -1)            return -4;         else
        {
            addrshift = 3;
            if(!firstaddr[0]) // If first executable instruction
            {
                decToHexA(firstaddr, locctr);                for(i =
ADRLEN - 1; i < OPLEN - 1; i++)
                    firstaddr[i] = ' ';
firstaddr[i] = 0;
            }
        }

        decToHexA(address, locctr);

        fprintf(intrptr, "%s %s\n", address, inLine); // Wrtie to Intermediate file         locctr +=
addrshift;

        fgets(inLine, LABLEN + MNEMLEN + OPLEN + 1, pgmptr);         fgetc(pgmptr);
        parseToken(label, inLine, LABLEN);         parseToken(mnemonic, inLine +
LABLEN + 1, MNEMLEN);         parseToken(operand, inLine + LABLEN + MNEMLEN
+ 1, OPLEN);
    }

    if(!firstaddr[0])
        return -5;

    if(!operand[0]) // END has no operand
    {
        inLine[LABLEN + MNEMLEN + 1] = 0;
        strcat(inLine, firstaddr);
    }
    fprintf(intrptr, "     %s", inLine); // Write END statement to Intermediate file

    fclose(pgmptr);
    fclose(intrptr);

    return locctr - startaddr;
    /*
    Return values:
    Program Length = (locctr - startaddr) - Success
-1 - Failed to opne Program file
-2 - Empty Program file
-3 - Multiple Label definitions
-4 - Invalid Opcode
-5 - No executable instruction
    */
}

int pass2(char fileloc[DIRLEN], int pgmlen, STAB stab, OPTAB optab)
```

```
{
   char targetloc[DIRLEN + 5], record[TEXTLEN], inLine[ADRLEN + LABLEN + MNEMLEN + OPLEN
+ 1];
   char label[LABLEN], mnemonic[MNEMLEN], operand[OPLEN], address[ADRLEN],
startaddr[7], pgmlenstr[7], objcode[7], reclen[ADRLEN];     int i, codelen, lenincr, indexed,
opervalue;
   FILE *intrptr, *objptr, *alstptr;

   strcpy(targetloc, fileloc);     strcat(targetloc,
".intr");
   intrptr = fopen(targetloc, "r"); // Open Intermediate file

   if(intrptr == NULL)
      return -1; // Intermediate file not found

   strcpy(targetloc, fileloc);     strcat(targetloc,
".alst");
   alstptr = fopen(targetloc, "w"); // Open Assembly listing

   strcpy(targetloc, fileloc);     strcat(targetloc,
".objp");
   objptr = fopen(targetloc, "w"); // Open Object code

   // Reading line from Intermediate File
   fgets(inLine, ADRLEN + LABLEN + MNEMLEN + OPLEN + 1, intrptr);     fgetc(intrptr);
   parseToken(address, inLine, ADRLEN);     parseToken(label, inLine + ADRLEN, LABLEN);
parseToken(mnemonic, inLine + ADRLEN + LABLEN + 1, MNEMLEN);     parseToken(operand,
inLine + ADRLEN + LABLEN + MNEMLEN + 1, OPLEN);

   // Creating Header Record
   record[0] = 'H';     record[1] =
0;
   if(!strcmp(mnemonic, "START"))
   {
      fprintf(alstptr, "%s       \n", inLine); // Writing Start line to Assembly Listing

      for(i = 0; label[i]; i++) // Appending Program Name to Record
record[i + 1] = label[i];         for(; i < 6; i++)             record[i + 1] = ' ';
      for(i = 0; i < 7 - ADRLEN; i++) // 6 - (ADRLEN - 1) = 7 - ADRLEN             startaddr[i] = '0';
// Padding 0's to Starting Address to fit to 6 characters         startaddr[i] = 0;
strcat(startaddr, operand);
      strcat(record, startaddr); // Appending Starting Address to Record

      fgets(inLine, ADRLEN + LABLEN + MNEMLEN + OPLEN + 1, intrptr);         fgetc(intrptr);
      parseToken(address, inLine, ADRLEN);
      parseToken(mnemonic, inLine + ADRLEN + LABLEN + 1, MNEMLEN); parseToken(operand,
       inLine + ADRLEN + LABLEN + MNEMLEN + 1, OPLEN);
   }     else         strcat(record, "
000000");     decToHex6(pgmlenstr,
pgmlen);     strcat(record, pgmlenstr);
   fprintf(objptr, "%s\n", record);

   codelen = 0;     record[0] = 'T';     for(i = 0; i < 6 - strlen(address); i++) // 6 - (ADRLEN - 1)
= 7 - ADRLEN         record[i + 1] = '0'; // Padding 0's to Starting Address to fit to 6 characters
record[i + 1] = 0;     strcat(record, address);
   strcat(record, "  "); // Space for Record Size

   while(strcmp(mnemonic, "END"))
```

```
   {
      // Creating Text Record
      if(!strcmp(mnemonic, "RESW") || !strcmp(mnemonic, "RESB"))
      {
         if(codelen) // Text Record did not start with RESW or RESB
         {
            decToHexA(reclen, codelen);
            record[7] = reclen[2];
record[8] = reclen[3];
            fprintf(objptr, "%s\n", record); // Writing Current Text Record
         }

         // Writing Current line to Assembly Listing            fprintf(alstptr, "%s
\n", inLine);

         // Reading Next line from Intermediate File
         fgets(inLine, ADRLEN + LABLEN + MNEMLEN + OPLEN + 1, intrptr);            fgetc(intrptr);
         parseToken(address, inLine, ADRLEN);
         parseToken(mnemonic, inLine + ADRLEN + LABLEN + 1, MNEMLEN);
parseToken(operand, inLine + ADRLEN + LABLEN + MNEMLEN + 1,
OPLEN);

         // Start a New Text Record //Set record to 'T' + address + "  "
codelen = 0;            record[0] = 'T';
         for(i = 0; i < 6 - strlen(address); i++) // 6 - (ADRLEN - 1) = 7 - ADRLEN               record[i +
1] = '0'; // Padding 0's to Starting Address to fit to 6 characters           record[i + 1] = 0;
strcat(record, address);
         strcat(record, "  "); // Space for Record Size

         continue;
       }
      else if(!strcmp(mnemonic, "WORD"))
      {
         lenincr = 3;           i =
strToDec(operand);
         decToHex6(objcode, i);
      }
      else if(!strcmp(mnemonic, "BYTE"))
      {
         lenincr = constBLen(operand);
         constToHex(objcode, operand);
      }
else       {
         lenincr = 3;
         i = searchOPTAB(optab, mnemonic);           objcode[1] =
optab.row[i].hexacode%16;
         if(objcode[1] > 9)
objcode[1] += 'A' - 10;
         else
            objcode[1] += '0';
         objcode[0] = optab.row[i].hexacode/16;
         if(objcode[0] > 9)
objcode[0] += 'A' - 10;
         else
            objcode[0] += '0';

         if(!operand[0])
opervalue = 0;
```

```
            else
            {
                if(indexed = indexedMode(operand))
                {
                    for(i = 0; operand[i] != ','; i++);
operand[i] = 0;
                }
                // Get the value of the operand
                if((i = searchSTAB(stab, operand)) == -1)
                {
                    printf("\ninLine :%s\nAddress: %s, Label: %s, Opcode: %s, Operand:
%s", inLine, address, label, mnemonic, operand);
return -2; // Invalid Symbol
                }
                opervalue = stab.row[i].address;
                if(indexed)
                    opervalue += 32768;
            }

           decToHexA(objcode + 2, opervalue); // Append New Operand Value to Objcode
          }

        //If New Object Code exceeds Text Record limit
        if(codelen + lenincr > 30)
        {
            // Writing to Object File              decToHexA(reclen,
codelen);
            record[7] = reclen[2];              record[8]
= reclen[3];              fprintf(objptr, "%s\n",
record);

            // Starting a New Text Record
            codelen = 0;
record[0] = 'T';
            for(i = 0; i < 6 - strlen(address); i++) // 6 - (ADRLEN - 1) = 7 - ADRLEN                  record[i +
1] = '0'; // Padding 0's to Starting Address to fit to 6 characters              record[i + 1] = 0;
strcat(record, address);
            strcat(record, "  "); // Space for Record Size
        }

        fprintf(alstptr, "%s %s\n", inLine, objcode); // Write the Current line + Object Code to Assembly
Listing
        strcat(record, objcode); // Appennding Current Object Code to Record          codelen += lenincr;

        // Reading Next line from Intermediate File
        fgets(inLine, ADRLEN + LABLEN + MNEMLEN + OPLEN + 1, intrptr);          fgetc(intrptr);
        parseToken(address, inLine, ADRLEN);          parseToken(mnemonic, inLine + ADRLEN +
LABLEN + 1, MNEMLEN);          parseToken(operand, inLine + ADRLEN + LABLEN + MNEMLEN +
1, OPLEN);
    }

    if(codelen) // Text Record is not Empty
    {
        decToHexA(reclen, codelen);
record[7] = reclen[2];          record[8] =
reclen[3];
        fprintf(objptr, "%s\n", record); // Writing Current Text Record
    }
```

```
    fprintf(alstptr, "%s       ", inLine); // Writing End line to Assembly Listing

    // Write End record
record[0] = 'E';     record[1] = 0;
    for(i = 0; i < 7 - ADRLEN; i++) // 6 - (ADRLEN - 1) = 7 - ADRLEN          startaddr[i] = '0'; //
Padding 0's to Starting Address to fit to 6 characters     startaddr[i] = 0;     strcat(startaddr,
operand);
    strcat(record, startaddr);
    fprintf(objptr, "%s", record); // Writing End Record

    fclose(intrptr);     fclose(alstptr);
    fclose(objptr);

    return 1;
    /*
    Return values:
    1  - Success
-1 - Failed to open Intermediate file
-2 - Invalid Symbol
    */
}

int main()
{
    char fileloc[DIRLEN];     int
pgmlen, errcode;     STAB stab;
    OPTAB optab;

    setOPTAB(&optab);

    printf("Enter file location (with name): ");     getLine(fileloc, DIRLEN
- 1);

    pgmlen = pass1(fileloc, &stab, optab);
    if(pgmlen < 0)
    {
        printf("\nError in pass 1: Errcode %d", pgmlen);         return 0;
    }

    printf("\nPass 1 successful wih length(bytes): %d", pgmlen);

    if((errcode = pass2(fileloc, pgmlen, stab, optab)) < 0)
printf("\nError in pass 1: Errcode %d", errcode);     else
        printf("\nPass 2 successful");

    return 0;
}
```

INPUT PROGRAM (SOURCE PROGRAM)

```
PGM1:   START 1000
            LDA   ALPHA
            MUL   BETA
            STA   GAMMA
ALPHA  WORD  2
BETA      WORD  4
GAMMA RESW  1
            END
```

OUTPUT PROGRAM OF PASS 1 (INTERMEDIATE FILE)

```
       PGM1:   START 1000
1000             LDA   ALPHA
1003             MUL   BETA
1006             STA   GAMMA
1009 ALPHA  WORD  2
100C BETA    WORD  4
100F GAMMA  RESW  1
                 END   1000
```

OUTPUT PROGRAM OF PASS 2

ASSEMBLY LISTING FILE

```
      PGM1:   START 1000
1000             LDA   ALPHA    001009
1003             MUL   BETA      20100C
1006             STA   GAMMA   0C100F
1009 ALPHA  WORD  2          000002
100C BETA     WORD  4          000004 100F
GAMMA  RESW  1
                   END   1000
```

OBJECT PROGRAM FILE

```
HPGM1  001000000012
T0010000F00100920100C0C100F000002000004
E001000
```

RESULT

The program has been executed successfully and output verified.

EXPERIMENT NO : 05 DATE :12/02/2022

SINGLE PASS ASSEMBLER

AIM

To implement single pass algorithm

INPUT

Source file in Assembly Language

OUTPUT

Assembly Listing file, Object Program file

ALGORITHM

```
Begin
Read input line
if OPCODE = 'START', then
                Set Starting Address as #[Operand]
                 Initialize LOCCTR to Starting Address
               Write line to Intermediate file
             Read next line
else
  Initialize LOCCTR to 0   Set Starting
Address to 0 end if

Write Header record to Object program file
Initialize first Text record

while OPCODE != 'END', do
  if line is a comment, then
  Read next input line
                      continue
           end if

                if there is a symbol in the LABEL field, then
             Search SYMTAB for LABEL                  if
found, then
                                  Set error flag (duplicate symbol)
                     else
                                   Add symbol to SYMTAB with it's address
                     end if
           end if

  Search OPTAB for OPCODE     if found,
then
                       Set LOCCTRincr to 3
```

```
  else if OPCODE = ‘WORD’, then
  Set LOCCTRincr to 3
              else if OPCODE = ‘RESW’, then
                        Set LOCCTRincr to 3*#[OPERAND]
              else if OPCODE = ‘REWB’, then
            Set LOCCTRincr to #[OPERAND]          else if
OPCODE = ‘BYTE’, then             Find length of
constant in bytes                 Set LOCCTRincr to
length      else
            Set error flag(invalid operation code)      end if

  if OPCODE = ‘RESB’ or ‘RESW’, then              if current Text record
is not empty, then                          Write Text record to Object
program file
                    end if
                     Write Listing line
                     Read next input line
                      Add LOCCTRincr to LOCCTR
                     Initialize new Text record

  else if OPCODE = ‘RESB’ or ‘RESW’, then
  Convert constant to object code
          else
                        if there is a symbol in OPERAND field, then
                     Search SYMTAB for OPERAND
            if found, then
                                        Store symbol value as operand address
                          else
                            Store 0 as operand address
            Set error flag(undefined symbol)                              end
if
                    else
  Store 0 as operand address   end if

                        Assemble object code instruction
          end if

               if object code will not into the current Text record, then
             Write Text record to object
  Initialize new Text record
          end if

          Add object code to Text record
          Write line to Intermediate file
             Read next input line
              Add LOCCTRincr ro LOCCTR
end while

Write last Tet record to Object progra file
Write last Listing line
Wrte End record to Object program file
Write (LOCCTR] – Starting address) to Program length field in Header record in Object program file

End Assemble
```

PROGRAM

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

#define DIRLEN 81 // Including '\0'
#define ADRLEN 5 // Including '\0'
#define CONSTLEN 17 // Including '\0'
#define LABLEN 7 // Including '\0'. Excluding ':'
#define MNEMLEN 6 // Including '\0'
#define OPLEN 12 // Including '\0'
#define SROWS 25
#define OROWS 10
#define TEXTLEN 70 // Including '\0'

typedef struct
{
   char label[LABLEN];
   int address;
}STAB_Row;

typedef struct
{    int size;
   STAB_Row row[SROWS];
}STAB;

typedef struct
{
   char mnemonic[MNEMLEN];    int
hexacode;
}OPTAB_Row;

typedef struct
{    int size;
   OPTAB_Row row[OROWS];
}OPTAB;

void getLine(char str[], int size) // Accept a line of upto given size
{
   char ch = 0;    int
i = 0;

   while((ch = getchar())!='\n')
      if(i < size)
str[i++] = ch;    str[i] = 0;
}
int strToDec(const char str[OPLEN]) // Returns the Decimal of a Decimal string
{
   int i, num = 0;

   for(i = 0; str[i]; i++)
      num = num*10 + str[i] - '0';

   return num;
}

int hexAToDec(const char hexstr[ADRLEN]) // Returns the Decimal of a
```

```
Hexadecimal string
{
   int decimal = 0, i;

   if(hexstr[0] == 0)
      return -1; // hexstr = "\0"

   for(i = 0; i < ADRLEN -  1; i++)
   {
      decimal *= 16;
      if(hexstr[i] >= '0' && hexstr[i] <= '9')            decimal
+= hexstr[i] - '0';
      else if(hexstr[i] >= 'A' && hexstr[i] <= 'F')
decimal += hexstr[i] - 'A' + 10;         else
         return -2; // hexstr does not represent a valid Hexadecimal number
   }

   return decimal;
}

void decToHexA(char hex[ADRLEN], const int decimal) // Sets hex to a Hexadecimal string of
Decimal
{
   int i, digit, dec = decimal;
   for(i = 0; i < ADRLEN - 1; i++)         hex[i] =
'0';
   hex[i] = 0;

   if(decimal < 0)
      hex[0] = 0;

   for(i = ADRLEN - 2; decimal && i >= 0; i--)
   {
      digit = dec % 16;
dec /= 16;         if(digit < 10)
hex[i] = digit + '0';         else
         hex[i] = digit - 10 + 'A';
   }
}

void decToHex6(char hex[7], const int decimal) // Sets hex to a Hexadecimal string of Decimal
{
   int i, digit, dec = decimal;
   strcpy(hex, "000000");

   if(decimal < 0)
      hex[0] = 0;

   for(i = 5; decimal && i >= 0; i--)
   {
      digit = dec % 16;
dec /= 16;         if(digit < 10)
hex[i] = digit + '0';         else
         hex[i] = digit - 10 + 'A';
   }
}

int constBLen(char str[]) // Returns length of constants of the form C'...' and X'...'in bytes
```

```
{
   int i, len = 0;

   if(str[0] == 'C' || str[0] == 'c')
   {
      for(i = 2; str[i] != '\''; i++, len++);         return
len;
   }
   else if(str[0] == 'X' || str[0] == 'x')
   {
      for(i = 2; str[i] != '\''; i++, len++);
      return (len + 1)/2;
   }
   return -1;
}

void constToHex(char hex[CONSTLEN], const char operand[OPLEN]) // Sets hex to a Hexidecimal
string  of Constant Equivalent
{     int i, j;
   char temp[CONSTLEN];
    hex[0] = 0;

    if(operand[0] == 'X' && operand[1] == '\'')
   {
      for(i = 0; operand[i + 2] != '\'' && i < (CONSTLEN - 1)/2; i++)             hex[i] =
operand[i + 2];
      hex[i] = 0;
if(i % 2)
      {
         strcpy(temp, hex);             strcpy(hex,
"0");
         strcat(hex, temp);
      }
   }
   else if(operand[0] == 'C' && operand[1] == '\'')
   {
      for(i = 0, j = 2; operand[j] != '\'' && i < 16; j++, i+=2)
      {
         hex[i + 1] = operand[j]%16;
if(hex[i + 1] > 9)                hex[i + 1] +=
'A' - 10;
         else
            hex[i + 1] += '0';

         hex[i] = operand[j]/16;
if(hex[i] > 9)                hex[i] += 'A'
- 10;            else
            hex[i] += '0';
      }
      hex[i] = 0;
   }
}

void parseToken(char token[], char str[], int size) // Parses a token of upto size specified from string
str. Uses delimiters ' ', '\0', ':', '\n'
{     int i;

   for(i = 0; i < size && str[i] && str[i] != ' ' && str[i] != ':' && str[i] != '\n'; i++)         token[i] = str[i];
```

```
    token[i] = 0;
}

int searchSTAB(STAB stab, char label[LABLEN]) // Returns index of row if found, -1 otherwise
{     int i;

    for(i = 0; i < stab.size; i++)
if(!strcmp(stab.row[i].label, label))
return i; return -1;
}

int insertSTAB(STAB *stab, char label[LABLEN], int address) // Returns 0 if overflow occurs, 1 on
success
{
    if(stab->size >= SROWS)
        return 0;

    stab->row[stab->size].address = address;     strcpy(stab->row[stab-
>size].label, label);     stab->size++;

    return 1;
}

int searchOPTAB(OPTAB optab, char mnemonic[MNEMLEN]) // Returns index of row if found, -1
otherwise
{     int i;

    for(i = 0; i < optab.size; i++)
        if(!strcmp(optab.row[i].mnemonic,                  mnemonic))
return i;     return -1;
}

void setOPTAB(OPTAB *optab) // Sets optab to Predefined OPTAB values
{
    optab->size = 25;

    strcpy(optab->row[0].mnemonic, "LDA");     optab->row[0].hexacode = 0;

    strcpy(optab->row[1].mnemonic, "LDX");     optab->row[1].hexacode = 4;

    strcpy(optab->row[2].mnemonic, "LDL");     optab->row[2].hexacode = 8;

    strcpy(optab->row[3].mnemonic, "STA");     optab->row[3].hexacode = 12;

    strcpy(optab->row[4].mnemonic, "STX");     optab->row[4].hexacode = 16;

    strcpy(optab->row[5].mnemonic, "STL");     optab->row[5].hexacode = 20;

    strcpy(optab->row[6].mnemonic,          "ADD");          optab-
     >row[6].hexacode = 24;     strcpy(optab->row[7].mnemonic,
     "SUB");
    optab->row[7].hexacode = 28;

    strcpy(optab->row[8].mnemonic, "MUL");
    optab->row[8].hexacode = 32;

    strcpy(optab->row[9].mnemonic, "DIV");
    optab->row[9].hexacode = 36;
```

```
        strcpy(optab->row[10].mnemonic, "COMP");
        optab->row[10].hexacode = 40;

        strcpy(optab->row[11].mnemonic, "TIX");
        optab->row[11].hexacode = 44;

        strcpy(optab->row[12].mnemonic, "JEQ");
        optab->row[12].hexacode = 48;

        strcpy(optab->row[13].mnemonic, "JGT");
        optab->row[13].hexacode = 52;

        strcpy(optab->row[14].mnemonic, "JLT");
        optab->row[14].hexacode = 56;

        strcpy(optab->row[15].mnemonic, "J");
        optab->row[15].hexacode = 60;

        strcpy(optab->row[16].mnemonic, "AND");
        optab->row[16].hexacode = 64;

        strcpy(optab->row[17].mnemonic, "OR");
        optab->row[17].hexacode = 68;

        strcpy(optab->row[18].mnemonic, "JSUB");
        optab->row[18].hexacode = 72;

        strcpy(optab->row[19].mnemonic, "RSUB");
        optab->row[19].hexacode = 76;

        strcpy(optab->row[20].mnemonic, "LDCH");
        optab->row[20].hexacode = 80;

        strcpy(optab->row[21].mnemonic, "STCH");
        optab->row[21].hexacode = 84;

        strcpy(optab->row[22].mnemonic, "RD");
        optab->row[22].hexacode = 216;

strcpy(optab->row[23].mnemonic, "WD"); optab->row[23].hexacode
                                    = 220;

        strcpy(optab->row[24].mnemonic, "TD");
        optab->row[24].hexacode = 224;
    }

    int indexedMode(char operand[OPLEN]) // Returns 1 if given operand is in Indexed Addressing Mode,
    0 otherwise
    {   int i;

        if(!operand[0])
    return 0;
        for(i = 0; operand[i + 1]; i++)
        {
            if(operand[i] == ',' && operand[i + 1] == 'X')          return 1;
        }
```

```
    return 0;
}

int assemble(char fileloc[DIRLEN])
{
    FILE *pgmptr, *alstptr, *objptr;     char
targetloc[DIRLEN + 5];
    char inLine[LABLEN + MNEMLEN + OPLEN + 1], label[LABLEN], mnemonic[MNEMLEN],
operand[OPLEN], address[ADRLEN];
    char record[TEXTLEN], objcode[7], startaddrstr[ADRLEN], firstaddr[OPLEN], reclen[ADRLEN],
pgmlenstr[7];
    int i, startaddr = 0, locctr = 0, addrshift, codelen, lenincr;     int  indexed,
opervalue;
    OPTAB optab;
    STAB stab;

    pgmptr = fopen(fileloc, "r");     if(pgmptr ==
NULL)
        return -1; // Program file not found     else
if(feof(pgmptr))
        return -2; // Program file is empty


    strcpy(targetloc, fileloc);     strcat(targetloc,
".alst");
    alstptr = fopen(targetloc, "w"); // Open Assembly listing

    strcpy(targetloc, fileloc);     strcat(targetloc,
".objp");
    objptr = fopen(targetloc, "w"); // Open Object code
     setOPTAB(&optab);
    stab.size = 0;
    firstaddr[0] = 0;

    fgets(inLine, LABLEN + MNEMLEN + OPLEN + 1, pgmptr);     fgetc(pgmptr);
    parseToken(label, inLine, LABLEN);     parseToken(mnemonic, inLine + LABLEN
+ 1, MNEMLEN);     parseToken(operand, inLine + LABLEN + MNEMLEN + 1,
OPLEN);     decToHexA(address, locctr);

    // Creating Header Record
    record[0] = 'H';     record[1] =
0;
    if(!strcmp(mnemonic, "START"))
    {
        startaddr = locctr = hexAToDec(operand);

        fprintf(alstptr, "     %s       \n", inLine); // Writing Start line to Assembly Listing

        for(i = 0; label[i]; i++) // Appending Program Name to Record
record[i + 1] = label[i];         for(; i < 6; i++)             record[i + 1] = ' ';
        for(i = 0; i < 7 - ADRLEN; i++) // 6 - (ADRLEN - 1) = 7 - ADRLEN             startaddrstr[i] = '0';
// Padding 0's to Starting Address to fit to 6 characters         startaddrstr[i] = 0;
strcat(startaddrstr, operand);
        strcat(record, startaddrstr); // Appending Starting Address to Record

        fgets(inLine, LABLEN + MNEMLEN + OPLEN + 1, pgmptr);         fgetc(pgmptr);
```

```
      parseToken(label, inLine, LABLEN);        parseToken(mnemonic, inLine +
LABLEN + 1, MNEMLEN);        parseToken(operand, inLine + LABLEN + MNEMLEN
+ 1, OPLEN);        decToHexA(address, locctr);
   }
else
      strcat(record, "      000000");
strcat(record, "      ");     fprintf(objptr,
"%s\n", record);

   codelen = 0;     record[0] = 'T';     for(i = 0; i < 6 - strlen(address); i++) // 6 - (ADRLEN - 1)
= 7 - ADRLEN         record[i + 1] = '0'; // Padding 0's to Starting Address to fit to 6 characters
record[i + 1] = 0;     strcat(record, address);
   strcat(record, "  "); // Space for Record Size
    while(strcmp(mnemonic, "END"))
   {
      if(label[0] == '/' && label[1] == '/') // If line is a comment
      {
         fgets(inLine, LABLEN + MNEMLEN + OPLEN + 1, pgmptr);            fgetc(pgmptr);
         parseToken(label, inLine, LABLEN);            parseToken(mnemonic, inLine +
LABLEN + 1, MNEMLEN);            parseToken(operand, inLine + LABLEN + MNEMLEN +
1, OPLEN);            decToHexA(address, locctr);
         continue;
      }

      if(label[0]) // If there is a Sybmol in Label column
      {
         if(searchSTAB(stab, label) > -1)
return -3;
         insertSTAB(&stab, label, locctr);
         stab.size++;
      }

      //Calculating next LOCCTR value
      if(!strcmp(mnemonic, "WORD"))            addrshift
= 3;
      else if(!strcmp(mnemonic, "RESW"))            addrshift =
3*strToDec(operand);        else if(!strcmp(mnemonic,
"RESB"))            addrshift = strToDec(operand);        else
if(!strcmp(mnemonic, "BYTE"))            addrshift =
constBLen(operand);        else if(searchOPTAB(optab,
mnemonic) == -1)            return -4;        else
      {
         addrshift = 3;
         if(!firstaddr[0]) // If first executable instruction               decToHexA(firstaddr,
locctr);
      }

      // Creating Text Record
      if(!strcmp(mnemonic, "RESW") || !strcmp(mnemonic, "RESB"))
      {
         if(codelen) // Text Record did not start with RESW or RESB
         {
            decToHexA(reclen, codelen);
record[7] = reclen[2];               record[8] =
reclen[3];
            fprintf(objptr, "%s\n", record); // Writing Current Text Record
         }
```

```
        // Writing Current line to Assembly Listing             fprintf(alstptr, "%s
%s      \n", address, inLine);

        locctr += addrshift;

        // Reading Next line from Program File
        fgets(inLine, LABLEN + MNEMLEN + OPLEN + 1, pgmptr);            fgetc(pgmptr);
        parseToken(label, inLine, LABLEN);            parseToken(mnemonic, inLine +
LABLEN + 1, MNEMLEN);            parseToken(operand, inLine + LABLEN + MNEMLEN +
1, OPLEN);            decToHexA(address, locctr);

        // Start a New Text Record //Set record to 'T' + address + "  "
codelen = 0;            record[0] = 'T';
        for(i = 0; i < 6 - strlen(address); i++) // 6 - (ADRLEN - 1) = 7 - ADRLEN                record[i +
1] = '0'; // Padding 0's to Starting Address to fit to 6 characters            record[i + 1] = 0;
strcat(record, address);
        strcat(record, "  "); // Space for Record Size

        continue;
    }
    else if(!strcmp(mnemonic, "WORD"))
    {
        lenincr = 3;            i =
strToDec(operand);
        decToHex6(objcode, i);
    }
    else if(!strcmp(mnemonic, "BYTE"))
    {
        lenincr = constBLen(operand);
        constToHex(objcode, operand);
    }
else      {
        lenincr = 3;
        i = searchOPTAB(optab, mnemonic);            objcode[1] =
optab.row[i].hexacode%16;
        if(objcode[1] > 9)
objcode[1] += 'A' - 10;
        else
            objcode[1] += '0';
        objcode[0] = optab.row[i].hexacode/16;
        if(objcode[0] > 9)
objcode[0] += 'A' - 10;
        else
            objcode[0] += '0';
        if(!operand[0])
opervalue = 0;
        else
        {
            if(indexed = indexedMode(operand))
            {
                for(i = 0; operand[i] != ','; i++);
operand[i] = 0;
            }
            // Get the value of the operand
            if((i = searchSTAB(stab, operand)) == -1)
            {
                printf("OPERAND: |%s|", operand);
                return -6; // Invalid Symbol
```

```
            }

            opervalue = stab.row[i].address;
            if(indexed)
               opervalue += 32768;
         }

        decToHexA(objcode + 2, opervalue); // Append New Operand Value to
Objcode
      }

      //If New Object Code exceeds Text Record limit
      if(codelen + lenincr > 30)
      {
         // Writing to Object File            decToHexA(reclen,
codelen);
         record[7] = reclen[2];            record[8]
= reclen[3];            fprintf(objptr, "%s\n",
record);

         // Starting a New Text Record
         codelen = 0;
record[0] = 'T';
         for(i = 0; i < 6 - strlen(address); i++) // 6 - (ADRLEN - 1) = 7 - ADRLEN               record[i +
1] = '0'; // Padding 0's to Starting Address to fit to 6 characters            record[i + 1] = 0;
strcat(record, address);
         strcat(record, "  "); // Space for Record Size
      }

      fprintf(alstptr, "%s %s %s\n", address, inLine, objcode); // Write the Current line
+ Object Code to Assembly Listing
      strcat(record, objcode); // Appennding Current Object Code to Record        codelen += lenincr;

      locctr += addrshift;

      // Reading Next line from Intermediate File
      fgets(inLine, LABLEN + MNEMLEN + OPLEN + 1, pgmptr);        fgetc(pgmptr);
      parseToken(label, inLine, LABLEN);        parseToken(mnemonic, inLine +
LABLEN + 1, MNEMLEN);        parseToken(operand, inLine + LABLEN + MNEMLEN
+ 1, OPLEN);        decToHexA(address, locctr);
   }

   if(codelen) // Text Record is not Empty
   {
      decToHexA(reclen, codelen);
record[7] = reclen[2];        record[8] =
reclen[3];
      fprintf(objptr, "%s\n", record); // Writing Current Text Record
   }

   if(!operand[0])
   {
      if(!firstaddr[0])
         return -5;

      strcpy(operand, firstaddr);        inLine[LABLEN +
MNEMLEN + 1] = 0;
      for(i = ADRLEN - 1; i < OPLEN - 1; i++)
```

```
            firstaddr[i] = ' ';         firstaddr[i]
= 0;         strcat(inLine, firstaddr);
   }

   fprintf(alstptr, "     %s       ", inLine); // Writing End line to Assembly Listing

   // Write End record
record[0] = 'E';     record[1] = 0;
   for(i = 0; i < 7 - ADRLEN; i++) // 6 - (ADRLEN - 1) = 7 - ADRLEN         startaddrstr[i] = '0'; //
Padding 0's to Starting Address to fit to 6 characters     startaddrstr[i] = 0;
strcat(startaddrstr, operand);     strcat(record, startaddrstr);
   fprintf(objptr, "%s", record); // Writing End Record

   decToHex6(pgmlenstr, locctr - startaddr);
   fseek(objptr, 13, 0);
   fprintf(objptr, "%s", pgmlenstr);

   fclose(pgmptr);     fclose(alstptr);
fclose(objptr);     return 1;

   /*Pass 1     Return
values:
   Program Length = (locctr - startaddr) - Success
-1 - Failed to open Program file
-2 - Empty Program file
-3 - Multiple Label definitions
-4 - Invalid Opcode
-5 - No executable instruction
-6 - Invalid Symbol
   1  - Success
   */
}

int main()
{     int errcode;
   char fileloc[DIRLEN];

   printf("Enter file location (with name): ");
   getLine(fileloc, DIRLEN - 1);

   errcode = assemble(fileloc);

   if(errcode < 0)
   {
      printf("\nError in assembly: Errcode %d", errcode);         return 0;
   }

   printf("\nAssembly successful");

   return 0;
}
```

INPUT PROGRAM (SOURCE PROGRAM)

```
PGM1        START       1000
ALPHA       WORD        2
BETA        WORD        4
GAMMA       RESW        1
            LDA         ALPHA
            MUL         BETA
            STA         GAMMA
            END
```

OUTPUT PROGRAM OF PASS 2

ASSEMBLY LISTING FILE

```
          PGM1 START       1000
 1000 ALPHA      WORD       2          000002
 1003 BETA       WORD       4          000004
 1006 GAMMA      RESW       1
1009             LDA        ALPHA      001000
100C             MUL        BETA       201003
100F             STA        GAMMA      0C1006
                  END       1009
```

OBJECT PROGRAM FILE

```
HPGM1  001000000012
T00100006000002000004
T0010090900100020100030C1006
E001009
```

RESULT

The program has been executed successfully and output verified.

MASM EXPERIMENTS

EXPERIMENT NO: 06 DATE: 16/02/2022

8 BIT ADDITION - MULTIPLICATION

AIM

To implement 8 bit addition and multiplication

INPUT

numbers to add

OUTPUT

sum and product

PROGRAM

1 - Addition

```
ASSUME CS:CODE, DS:DATA

DATA SEGMENT

        M1 DB 10,13,"Enter first number:$"

        M2 DB 10,13,"Enter second number:$"

       M3 DB 10,13,"Sum: $"

DATA ENDS
PRTMSG      MACRO       MESSAGE

    LEA     DX, MESSAGE

       MOVAH,09

    INT     21H
    ENDM

 GETDCM      MACRO

    MOV     AH, 01

    INT     21H

    SUB     AL, 30H

    ENDM

CODE SEGMENT
```

```
START: MOV      AX, DATA
       MOV      DS, AX

       PRTMSG          M1

       GETDCM
       MOV      BL, AL
       PRTMSG M2
       GETDCM
       ADD      AL, BL
       MOV      AH, 00H
       AAA
       MOV      BX, AX
       PRTMSG  M3
       MOV      DL, BH
       ADD      DL, 30H
       MOV      AH, 02
       INT      21H
       MOV      DL, BL
       ADD      Dl, 30H
       INT      21H
       MOV      AH,4CH
       INT      21H
CODE ENDS
END START
```

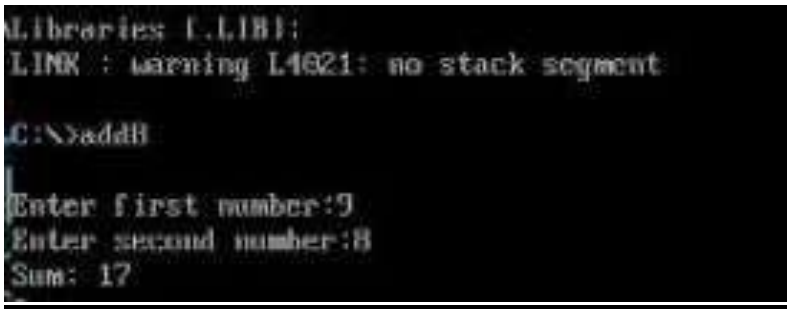
2.Multiplication

```
DATA SEGMENT
M1 DB 13,10,"ENTER 2 NUMBERS  $"
M2 DB 13,10,"PRODUCT IS $"
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE , DS:DATA
```

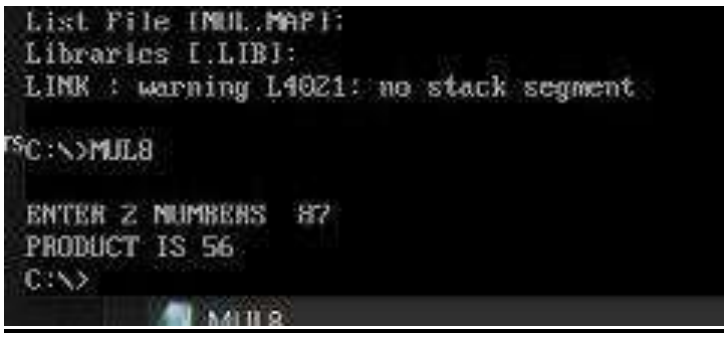
```
START:MOV AX,DATA
      MOV DS,AX
      LEA DX,M1
      MOV AH,09H
      INT 21H
      MOV AH,01H
      INT 21H
      SUB AL,30H
      MOV BL,AL
      MOV AH,01H
      INT 21H
      SUB AL,30H
      MOV AH,00H
      MUL BL
      AAM
      MOV BX,AX
            LEA DX,M2
            MOV AH,09H
           INT 21H
      MOV DL,BH
      OR DL,30H
      MOV AH,02H
      INT 21H
      MOV DL,BL
      OR DL,30H
      MOV AH,02H
      INT 21H
      MOV AH,4CH
      INT 21H

      CODE ENDS
END START
```

OUTPUT

1.Addition:



2. Multiplication:



RESULT

The program has been executed successfully and output verified.

EXPERIMENT NO: 07 DATE: 16/02/2022

EVEN OR ODD

AIM
To check whether number is even or odd

INPUT
number to check

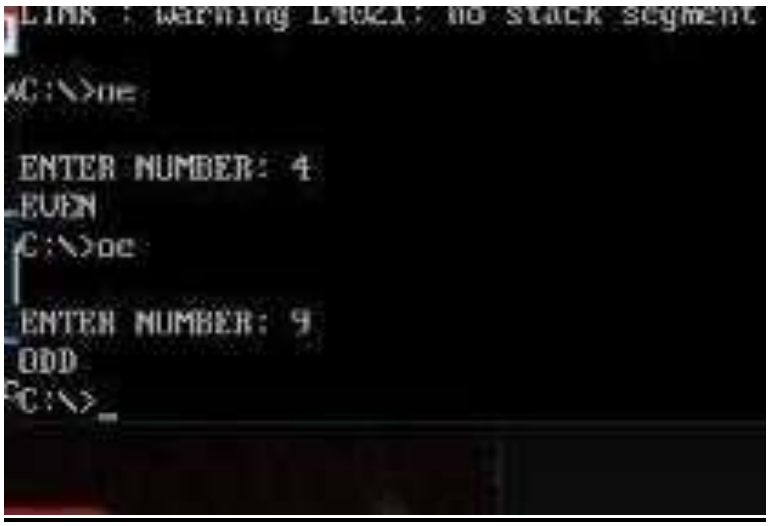
OUTPUT
even or odd

PROGRAM

```
ASSUME      CS:CODE, DS:DATA

DATA SEGMENT
        M1      DB 10,13,"ENTER NUMBER: $"
        M2      DB 10,13,"ODD$"
        M3      DB 10,13,"EVEN$"
DATA ENDS

PRTMSG      MACRO       MSG
        LEA     DX, MSG
        MOV     AH, 09
        INT     21H
        ENDM

GETDCM      MACRO
        MOV     AH, 01
        INT     21H
        SUB     AL, 30H
        ENDM
```

```
CODE SEGMENT
START: MOV    AX, DATA
       MOV    DS, AX

       PRTMSG        M1
       GETDCM
       SHR    AL, 01

       JC     ODD
       PRTMSG        M3
       JMP    DONE

 ODD: PRTMSG         M2
DONE: MOV    AH, 4CH
       INT    21H

CODE ENDS
END START
```

OUTPUT



RESULT

The program has been executed successfully and output verified.

EXPERIMENT NO: 08 DATE: 21/02/2022

16 BIT ADDITION - MULTIPLICATION

AIM

To implement 16 bit addition and multiplication

INPUT

numbers to add

OUTPUT

sum and product

PROGRAM:

1.Addition

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
        M1      DB 10,13,"ENTER FIRST NUMBER: $"
        M2      DB 10,13,"ENTER SECOND NUMBER: $"
        M3      DB 10,13,"SUM: $"

        SUM     DB 03
DATA ENDS

                                MSG
PRTMSG          MACRO
        LEA     DX, MSG
        MOV     AH, 09
        INT     21H
        ENDM
GETDCM          MACRO
        MOV     AH, 01
        INT     21H
        SUB     AL, 30H
        ENDM
PRTDCM          MACRO
        MOV     DL,[SI]
```

```
        ADD     DL, 30H
        MOV     AH, 02
        INT     21H
        ENDM

CODE SEGMENT
START: MOV      AX, DATA
        MOV     DS, AX
        PRTMSG          M1
        GETDCM
  MOV    BH, AL   GETDCM
        MOV     BL, AL
        PRTMSG          M2
        GETDCM
  MOV    CH, AL   GETDCM
        MOV     CL, AL

        ADD     BL, CL
        MOV     AL, BL
                AH, 00
        MOV
        AAA

        LEA     SI, SUM
        MOV     [SI], AL

        ADD     BH, AH
        ADD     BH, CH
        MOV     AL, BH
                AH, 00
        MOV
        AAA
```

```
        INC     SI
                [SI], AL
        MOV

        INC     SI
        MOV     [SI], AH

        PRTMSG          M3
PRTDCM          DEC
SI      PRTDCM
DEC     SI
        PRTDCM
        MOV     AH, 4CH
        INT     21H
CODE ENDS
END START
```

2. Multiplication

```
ASSUME          CS:CODE, DS:DATA

DATA SEGMENT
        M1      DB      10, 13, "ENTER FIRST NUMBER: $"
        M2      DB      10, 13, "ENTER SECOND NUMBER: $"
        M3      DB      10, 13, "PRODUCT: $"
        PROD DB         4 DUP(00H)
DATA ENDS
PRTMSG          MACRO           MESSAGE
        LEA     DX, MESSAGE
        MOV AH, 09
        INT     21H
        ENDM
GETDCM          MACRO
        MOV AH, 01
        INT     21H
        SUB     AL, 30H
        ENDM
```

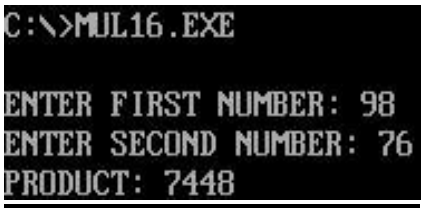
```
PRTDCM          MACRO
        MOV DL, [SI]
        ADD DL, 30H
        MOV AH, 02
        INT     21H
        ENDM
CODE SEGMENT
        START:          MOV AX, DATA
                MOV DS, AX
                PRTMSG          M1
                GETDCM
              MOV BH, AL          GETDCM
                MOV BL, AL

                PRTMSG          M2
                GETDCM
              MOV CH, AL          GETDCM
                MOV CL, AL
               LEA      SI, PROD
                MOV AH, 00H
                MUL BL
               AAM
                MOV [SI], AL
               INC      SI
                MOV [SI], AH

                MOV AH, 00H
                MOV AL, BH
                MUL CL
               AAM
                MOV DX, AX
                ADD DL, [SI]
                MOV AH, 00H
                MOV AL, CH
```

```
                MUL BL
               AAM
                ADD DX, AX
                MOV AL, DL
                MOV AH, 00H
               AAM
                ADD DH, AH
                MOV DL, DH
                MOV DH, 00H
                MOV [SI], AL
              INC   SI
                MOV AH, 00H
                MOV AL, BH
               MUL CH
              AAM
                ADD DX, AX
                MOV AL, DL
                MOV AH, 00H
              AAM
                MOV [SI], AL
             INC   SI
        ADD DH, AH  MOV AL, DH
        MOV [SI], AL    PRTMSG
                          M3
              PRTDCM
             DEC   SI
              PRTDCM
             DEC   SI
              PRTDCM
             DEC   SI
              PRTDCM
                MOV AH, 4CH
             INT   21H
CODE ENDS
END START
```

OUTPUT

1.Addition

2. Multiplication



RESULT

The program has been executed successfully and output verified.

EXPERIMENT NO: 09 DATE: 21/02/2022

LINEAR SEARCH

AIM

To implement linear search.

INPUT

Numbers to add

OUTPUT

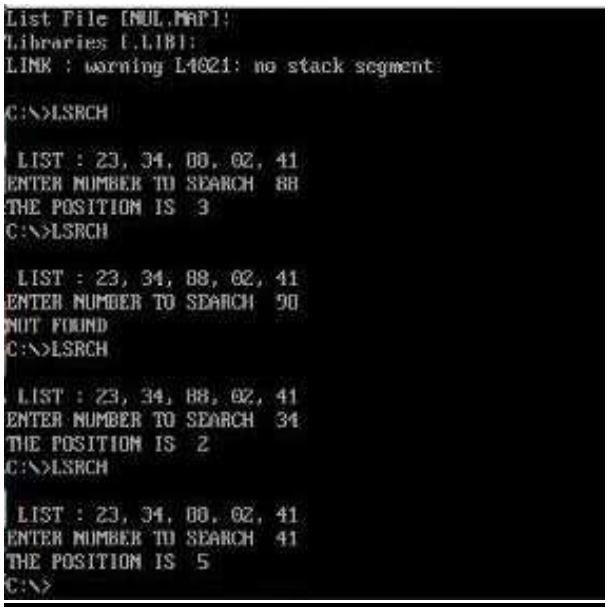
Location of key

PROGRAM:

```
START:  MOV AX,DATA
        MOV DS,AX
        LEA DX,M4
        MOV AH,09H
        INT 21H
        XOR DX,DX
        LEA DX,M1
        MOV AH,09H
        INT 21H
        MOV AH,01
        INT 21H
        SUB AL,30H
        SHL AL,01
        SHL AL,01
        SHL AL,01
        SHL AL,01
        MOV BL,AL
MOV AH,01       INT 21H
        SUB AL,30H
        ADD AL,BL
```

```
        MOV CL,01H
        MOV SI,OFFSET LIST
CLOOP:  CMP [SI],AL
        JZ FOUND
        INC SI
        INC CL
        CMP CL,06H
        JNZ CLOOP
        CMP CL,06H
        JZ XXX

FOUND:  MOV AL,CL
        ADD AL,00H
        AAA
        OR AL,30H
        MOV CL,AL
        LEA DX,M2
        MOV AH,09H
        INT 21H
        MOV DL,CL
  MOV AH,02H    INT 21H
        MOV AH,4CH
        INT 21H

XXX:    LEA DX,M3
        MOV AH,09H
        INT 21H
        MOV AH,4CH
        INT 21H

CODE ENDS
 END START
```

OUTPUT



RESULT

The program has been executed successfully and the result is verified.

EXPERIMENT NO: 10 DATE: 23/02/2022

STRING MANIPULATION

AIM

To find number of vowels, consonants and digits in string

INPUT

String

OUTPUT

count

PROGRAM:

```
ASSUME CS:CODE,DS:DATA,ES:EXTRA

DATA SEGMENT

        M1      DB      10 ,13, "ENTER STRING(DELIMITER: `): $"

        M2      DB      10, 13, "NUMBER OF VOWELS: $"

        M3      DB      10, 13, "NUMBER OF DIGITS: $"

        M4      DB      10, 13, "NUMBER OF CONSONANTS: $"


         INSTR DB       "Hello123"

        MAXLEN          DB      0AH

         DELIM DB       "`"


         VCNT DB        00H

        DGCNT DB        00H

        CNCNT DB        00H

DATA ENDS
EXTRA SEGMENT

        VWSTR DB        "aeiouAEIOU"
```

```
        DGSTR DB        "0123456789"
EXTRA ENDS

PRTMSG          MACRO           MESSAGE
        LEA     DX, MESSAGE
        MOV     AH, 09
        INT     21H
        ENDM

                                        COUNT
PRTCNT          MACRO
        MOV     DL, COUNT
        ADD     DL, 30H
        MOV     AH, 02
        INT     21H
        ENDM
CODE SEGMENT
        START: MOV      AX, DATA
               MOV      DS, AX
               MOV      AX, EXTRA
               MOV      ES, AX

               LEA      SI, INSTR
               PRTMSG   M1
               MOV      BX, 00
               MOV      CH, 00H
               MOV      CL, MAXLEN
               MOV      AH, 01
         GETC: INT      21H
               CMP      AL, DELIM
               JE       ENDGET
               INC      BL
               MOV      [SI], AL
               INC      SI
```

```
                LOOP GETC
ENDGET:         CLD
        LEA     SI, INSTR
CHKA: MOV       AX, [SI]
        INC     SI
        MOV     CL, 0AH
        LEA     DI, VWSTR
         REPNZ SCASB
        JNE     CHKD
        INC     VCNT
                ENDC
        JMP

CHKD: MOV       CL, 0AH
        LEA     DI, DGSTR
 REPNZ SCASB    JNE     CHKC
        INC     DGCNT
        JMP     ENDC

 CHKC: INC      CNCNT
 ENDC: MOV      CL, BL
        DEC     BX
          LOOP CHKA
         PRTMSG         M2
         PRTCNT         VCNT
         PRTMSG         M3
         PRTCNT         DGCNT
         PRTMSG         M4
         PRTCNT         CNCNT

        MOV     AH, 4CH
        INT     21H
CODE ENDS
 END    START
```

OUTPUT

RESULT

The program has been executed successfully and output verified.

TRAINER KIT

EXPERIMENT NO : 11 DATE : 23/02/2022

ADDITION OF TWO 16-BIT NUMBERS

AIM

To add two 16-bit numbers using a trainer kit.

PROGRAM

ADDRESS	MNEMONICS
0400	AND AX,0000H
0403	MOV BX,0600H
0406	MOV SI,0500H
0409	MOV DI,0550H
040C	MOV AX,[SI]
040E	ADD AX,[DI]
0410	MOV [BX],AX
0412	MOV AX,0000H
0415	ADC AX,0000H
0418	MOV [BX+2],AX
041B	HLT

INPUT

0500 - B5
0501 - 7A
0550 - 2A
0551 – E5

OUTPUT

0600 - DF
0601 - 5F
0602 - 01

RESULT

The program has been executed successfully and the output has been verified.

EXPERIMENT NO : 12 DATE : 23/02/2022

SUBTRACTION OF TWO 16-BIT NUMBERS

AIM

To subtract two 16-bit numbers using a trainer kit.

PROGRAM

ADDRESS	MNEMONICS
0400	CLC
0401	MOV BX,0900H
0404	MOV SI,0700H
0407	MOV DI,0800H
040A	MOV AX,[SI]
040C	SBB AX,[DI]
040E	MOV [BX],AX
0410	HLT

INPUT

0700 - 18
0701 - 08
0800 - 40
0801 - 10

OUTPUT

0900 - D8
0901 - F7

RESULT

The program has been executed successfully and the output has been verified.

EXPERIMENT NO : 13 DATE : 23/02/2022

MULTIPLICATION OF TWO 16-BIT NUMBERS

AIM

To multiply two 16-bit numbers using a trainer kit.

PROGRAM

ADDRESS	MNEMONICS
0400	CLC
0401	MOV BX,0700H
0404	MOV SI,0750H
0407	MOV DI,0800H
040A	MOV AX,[SI]
040C	MOV CX,[DI]
040E	MUL CX
0410	MOV [BX],AX
0412	MOV [BX+2],[DX]
0415	HLT

INPUT

0750 - 1A
0751 - 2B
0800 - 4B
0801 - 12

OUTPUT

0700 - 9E
0701 - 74
0702 - 14
0703 - 03

RESULT

The program has been executed successfully and the output has been verified.

EXPERIMENT NO : 14 DATE : 23/02/2022

DIVISION OF TWO 16-BIT NUMBERS

AIM

To divide two 16-bit numbers using a trainer kit.

PROGRAM

ADDRESS	MNEMONICS
0400	CLC
0401	MOV BX,0700H
0404	MOV SI,0750H
0407	MOV DI,0800H
040A	MOV AX,[SI]
040C	MOV CX,[DI]
040E	MOV CH,00H
0410	DIV CX
0412	MOV [BX],AX
0414	HLT

INPUT

0750 - 43
0751 - 12
0800 - 21

OUTPUT

0700 - 8D (Quotient)
0701 - 16 (Remainder)

RESULT

The program has been executed successfully and the output has been verified.

EXPERIMENT NO : 15 DATE : 23/02/2022

MAXIMUM OF N NUMBERS

AIM

To find the maximum of n numbers using a trainer kit.

PROGRAM

ADDRESS	MNEMONICS
0400	CLC
0401	MOV BX,0700H
0404	MOV SI,0800H
0407	MOV CX,0005H
040A	MOV AL,00H
040C	CMP AL,[SI]
040E	JA 0412H
0410	MOV CH,00H
0412	INC SI
0413	LOOPNZ 040CH
0415	MOV [BX],AL
0417	HLT

INPUT

0800 - 77
0801 - 81
0802 - B4
0803 - F1
0804 - AB

OUTPUT

0700 - F1

RESULT

The program has been executed successfully and the output has been verified.

EXPERIMENT NO : 16 DATE : 23/02/2022

SORTING NUMBERS IN ASCENDING ORDER

AIM

To sort the numbers in ascending order using a trainer kit.

PROGRAM

ADDRESS	MNEMONICS
0400	MOV SI,0700H
0403	MOV BX,[SI]
0405	DEC BX
0406	MOV CX,[SI]
0408	DEC CX
0409	MOV SI,0702H
040C	MOV AL,[SI]
040E	INC SI
040F	CMP AL,[SI]
0411	JBE 0419H
0413	XCHG AL,[SI]
0415	DEC SI
0416	MOV [SI],AL
0418	INC SI
0419	LOOP 040CH

041B	DEC BX
041C	MOV [SI],0700H
041F	JNE 0406H
0421	HLT

INPUT

0700 - 05

0702 - 12
0703 - 97
0704 - 41
0705 - 14
0706 - AA

OUTPUT

0702 - 12
0703 - 14
0704 - 41
0705 - 97
0706 - AA

RESULT

The program has been executed successfully and the output has been verified.