INDEX

EXPERIMENTS FROM OPERATING SYSTEM

DATE :

EXPERIMENT NO – 1.1(i)
CPU SCHEDULING ALGORITHM - FCFS

AIM

To simulate the First Come First Serve scheduling algorithm to find turn around time and waiting time .

ALGORITHM

1. Start the program
2. **Define Process Structure:**
    - ➢ Create a structure named **Process** with attributes like **pid** (Process ID), **burst_time** (Burst time), **at** (Arrival Time), **tat** (Turnaround Time), and **waittime** (Waiting Time).
3. **Function fcfsScheduling:**
    - ➢ Inputs an array of **Process** structures (**proc[]**) and the total number of processes **n**.
    - ➢ Sorts the processes based on their Arrival Time in ascending order using the Bubble Sort algorithm:
        - Iterates through the array comparing the arrival time of adjacent processes.
        - Swaps the processes if their arrival time is not in the correct order.
    - ➢ Initializes variables for calculating total waiting time, total turnaround time, and the current time.
    - ➢ Iterates through the processes:
        - Checks if the current process's Arrival Time (**at**) is less than or equal to the current time (**time**):
            - o If true, increments the time by the process's burst time (**burst_time**).
            - o Calculates Turnaround Time (**tat**) and Waiting Time (**waittime**) for the process.
            - o Updates the total turnaround time and total waiting time accordingly.
            - o Moves to the next process.
            - o If the Arrival Time condition is not met, increments time by 1.
    - ➢ Displays the details of each process: Process ID, Burst Time, Arrival Time, Waiting Time, and Turnaround Time.

OUTPUT

```
Enter the number of processes: 5

Enter process details (Process ID, Burst Time, Arrival Time):
Process 1: 4
9
0
Process 2: 3
6
2
Process 3: 5
4
3
Process 4: 1
8
0
Process 5: 2
11
3
Process         BT      AT      WT       TAT
4               9       0       0       9
1               8       0       9       17
3               6       2       15      21
5               4       3       20      24
2               11      3       24      35

Average Waiting Time: 13.60
Average Turnaround Time: 21.20
```

- Calculates and displays the Average Waiting Time and Average Turnaround Time for all processes.

4. **Function main:**
   - Declares variables **n** and **i** to store the number of processes and loop counters, respectively.
   - Prompts the user to input the number of processes.
   - Creates an array **proc[]** of **Process** structures with a size of **n**.
   - Prompts the user to input details (Process ID, Burst Time, Arrival Time) for each process.
   - Calls the **fcfsScheduling** function passing the array of processes and the total number of processes.
5. Stop the program .

PROGRAM

```
#include<stdio.h>
struct Process {
  int pid;
  int burst_time;
  int at;
  int tat;
  int waittime;
};
void fcfsScheduling(struct Process proc[], int n) {
      int i, j;
      struct Process temp;
      for (i = 0; i < n - 1; i++) {
          for (j = 0; j < n - i - 1; j++) {
                    if (proc[j].at > proc[j + 1].at) {
                        temp = proc[j];
                        proc[j] = proc[j + 1];
                        proc[j + 1] = temp;
                    }
          }
     }
     int total_waiting_time = 0;
     int total_turnaround_time = 0;
     int time=0;
     i=0;
```

```
    while (i<n) {
        if(proc[i].at<=time){
             time+=proc[i].burst_time;
             proc[i].tat=time-proc[i].at;
             proc[i].waittime=proc[i].tat-proc[i].burst_time;
             total_turnaround_time+=proc[i].tat;
             total_waiting_time+=proc[i].waittime;
             i++;
        }
        Else{
              time++;
        }
   }
   printf("Process\tBT\tAT\tWT \t TAT\n");
   for (i = 0; i < n; i++) {
       printf("%d\t%d\t%d\t%d\t%d\n", proc[i].pid, proc[i].burst_time,
       proc[i].at,proc[i].waittime,proc[i].waittime+proc[i].burst_time);
   }
    printf("\nAverage Waiting Time: %.2f\n", (float) total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float) total_turnaround_time / n);
}
void main() {
 int n, i;
 printf("Enter the number of processes: ");
 scanf("%d", &n);
 struct Process proc[n];
 printf("Enter process details (Process ID, Burst Time, Arrival Time):\n");
 for (i = 0; i < n; i++) {
 printf("Process %d: ", i + 1);
 scanf("%d %d %d", &proc[i].pid, &proc[i].burst_time, &proc[i].at);
 }
 fcfsScheduling(proc, n);
}
```

RESULT

C program to implement FCFS scheduling algorithm has been executed successfully .

DATE :

EXPERIMENT NO – 1.1(ii)
CPU SCHEDULING ALGORITHM - SJF

AIM

To simulate the Shortest Job First scheduling algorithm to find turn around time and waiting time .

ALGORITHM

1. Start the program
2. **Include Necessary Libraries:** The algorithm starts by including necessary header files like **<stdio.h>** and **<limits.h>**.
3. **Define a Process Structure:** It defines a structure named **process** that holds various attributes of a process - arrival time (**at**), burst time (**bt**), waiting time (**wt**), turnaround time (**tat**), completion time (**ct**), and a status flag (**status**) indicating whether the process has been executed or not.
4. **Main Function:**
   - Declare variables **n**, **i**, **totalTAT**, **totalWT**, **completed**, and **currentTime**.
   - Prompt the user to enter the number of processes (**n**).
   - Create an array of **process** structures (**a[n]**).
   - Ask the user to input arrival time and burst time for each process while calculating the total burst time (**sum**) of all processes.
   - Initialize the status of each process as **0** (not executed) initially.
5. **Scheduling Algorithm:**
   - Enter a while loop that runs until all processes are completed.
   - Initialize variables **sJob** and **sBt**.
   - Iterate through each process and find the process with the shortest burst time (**sBt**) among the processes that have arrived (**at <= currentTime**) and have not yet been executed (**status=0**).
   - If a suitable process is found (**sJob != -1**), update its completion time, calculate its turnaround time and waiting time, update total turnaround time and waiting time, mark it as completed (**status=1**), and increment **completed**.
   - If no suitable process is found, increment **currentTime**.
6. **Calculate Averages and Display Results:**
   - Calculate the average turnaround time (**avgTAT**) and average waiting time (**avgWT**).
   - Display a table with process details including arrival time, burst time, completion time, turnaround time, and waiting time for each process.

OUTPUT

```
Enter the number of processes: 5
Enter the arrival time and burst time
2
6
5
2
1
8
0
3
4
4

PNo     AT      BT      CT      TAT     WT
P1      2       6       9       7       1
P2      5       2       11      6       4
P3      1       8       23      22      14
P4      0       3       3       3       0
P5      4       4       15      11      7

Average Waiting time = 5.200000
Average Turn Around time = 9.800000
```

- Print the calculated average waiting time and average turnaround time.

7. **End of Algorithm.**

PROGRAM

```
#include <stdio.h>
#include <limits.h>
typedef struct{
   int at,bt,wt,tat,ct,status;
}process;
int main()
{
   int n,i;
   int totalTAT=0,totalWT=0,completed=0,currentTime=0;
   printf("Enter the number of processes: ");
   scanf("%d",&n);
   process a[n];
   printf("Enter the arrival time and burst time\n");
   int sum=0;
   for(i=0;i<n;i++)
   {
      scanf("%d",&a[i].at);
      scanf("%d",&a[i].bt);
      sum+=a[i].bt;
      a[i].status=0;
   }
   while(completed<n)
   {
      int sJob=-1;
      int sBt = INT_MAX;
      for(int i=0;i<n;i++)
      {
         if(a[i].status==0 && a[i].bt<sBt && a[i].at<=currentTime)
         {
            sJob = i;
            sBt = a[i].bt;
         }
      }
      if(sJob==-1)
```

```
        {
           currentTime++;
        }
        else
        {
           a[sJob].ct=currentTime+a[sJob].bt;
           currentTime = a[sJob].ct;
           a[sJob].tat = a[sJob].ct-a[sJob].at;
           a[sJob].wt = a[sJob].tat-a[sJob].bt;
           totalTAT+=a[sJob].tat;
           totalWT+=a[sJob].wt;
           completed++;
              a[sJob].status=1;
        }
     }
float avgTAT = (float)totalTAT/n;
   float avgWT = (float)totalWT/n;
   printf("\nPNo\tAT\tBT\tCT\tTAT\tWT\n");
   for(i=0;i<n;i++)
   {
      printf("P%d\t%d\t%d\t%d\t%d\t%d\n",i+1,a[i].at,a[i].bt,a[i].ct,a[i].tat,a[i].wt);
   }
   printf("\n");
   printf("Average Waiting time = %f\n",avgWT);
   printf("Average Turn Around time = %f\n",avgTAT);
}
```

RESULT

C program to implement SJF scheduling algorithm has been executed successfully .

DATE :

EXPERIMENT NO – 1.1(iii)
CPU SCHEDULING ALGORITHM – ROUND ROBIN

AIM

To simulate the Round Robin scheduling algorithm to find turn around time and waiting time .

ALGORITHM

1. Start the program
2. Initialize arrays and variables: Initialize arrays to store arrival time (at), burst time (bt), remaining time (rt), process IDs (id), waiting time (wt), turnaround time (tat), and a queue for the ready processes. Initialize front and rear of the queue.
3. Insert and Delete functions: Include functions to insert and delete elements into/from the queue.
4. Main function:
    - ➢ Take input for the number of processes (n), arrival time, and burst time for each process.
    - ➢ Input the time quantum (TQ).
5. Round Robin Scheduling:
    - ➢ Insert the first process into the ready queue and mark it as existing.
    - ➢ While the ready queue is not empty:
        - Remove a process from the queue.
        - If the remaining time of the process is greater than or equal to the time quantum (TQ), reduce its remaining time by TQ and increase the total time by TQ.
        - If the remaining time of the process is less than TQ, execute the remaining time and increase the total time accordingly.
        - Check for other processes that have arrived and insert them into the queue if they haven't been added yet.
        - Calculate turnaround time (TAT) and waiting time (WT) for each process.
        - Update the total turnaround time and waiting time.
6. Calculate averages: Calculate the average waiting time and average turnaround time for all processes.
7. Output: Display the process IDs, burst time, arrival time, waiting time, and turnaround time for each process. Also, print the average waiting time and average turnaround time for all processes.
8. Return: Return 0 to indicate successful execution.
9. Stop the program

OUTPUT

```
Enter the number of the process
3
Enter the arrival time and burst time of the process
AT      BT
0       10
1       8
2       7
Enter the time quantum
5
ID      BT      AT      WT      TAT
0       10      0       10      20
1       8       1       14      22
2       7       2       16      23
Average waiting time of the processes is : 13.333333
Average turn around time of the processes is : 21.666666
```

PROGRAM

```
#include<stdio.h>
int at[10],bt[10],rt[10],TQ,id[10],wt[10],tat[10];
// declaration of the ready queue
int queue[100];
int front=-1;
int rear=-1;
void insert(int n)
{
 if(front==-1)
 front=0;
 rear=rear+1;
 queue[rear]=n;
}
int delete()
{
 int n;
 n=queue[front];
 front=front+1;
 return n;
}
int main()
{
 int n,TQ,p,TIME=0;
 int temp[10],exist[10]={0};
 float total_wt=0,total_tat=0,Avg_WT,Avg_TAT;
 printf("Enter the number of the process\n");
 scanf("%d",&n);
 printf("Enter the arrival time and burst time of the process\n");
 printf("AT BT\n");
 for(int i=0;i<n;i++)
 {
 scanf("%d%d",&at[i],&bt[i]);
 id[i]=i;
 rt[i]=bt[i];
 }
 printf("Enter the time quantum\n");
 scanf("%d",&TQ);
```

```
insert(0);
exist[0]=1;
while(front<=rear)
{
p=delete();
if(rt[p]>=TQ)
{
rt[p]-=TQ;
TIME=TIME+TQ;
}
else
{
TIME=TIME+rt[p];
rt[p]=0;
}
for(int i=0;i<n;i++)
{
if(exist[i]==0 && at[i]<=TIME)
{
insert(i);
exist[i]=1;
}
}
if(rt[p]==0)
{
tat[p]=TIME-at[p];
wt[p]=tat[p]-bt[p];
total_tat=total_tat+tat[p];
total_wt=total_wt+wt[p];
}
else
{
insert(p);
}
}
Avg_TAT=total_tat/n;
Avg_WT=total_wt/n;
printf("ID BT AT WT TAT\n");
```

```
 for(int i=0;i<n;i++)
 {
 printf("%d  %d  %d  %d  %d\n",id[i],bt[i],at[i],wt[i],tat[i]);
 }
 printf("Average waiting time of the processes is : %f\n",Avg_WT);
 printf("Average turn around time of the processes is : %f\n",Avg_TAT);
 return 0;
}
```

RESULT

C program to implement Round Robin scheduling algorithm has been executed successfully .

DATE :

EXPERIMENT NO – 1.1(iv)
CPU SCHEDULING ALGORITHM - PRIORITY

AIM

To simulate the Priority scheduling algorithm to find turn around time and waiting time .

ALGORITHM

1. Start the program
2. **Input**: Receive the number of processes **n**.
3. Create a **process** struct with attributes: **id**, **at** (arrival time), **bt** (burst time), **ct** (completion time), **tat** (turnaround time), **wt** (waiting time), **rt** (remaining time), and **priority**.
4. Input the details of each process: id, arrival time, burst time, and priority.
5. Initialize variables **completed**, **totalWT**, **totalTAT**, and **currentTime** to keep track of completed processes and total waiting and turnaround times.
6. Perform scheduling until all processes are completed:
    - Inside the loop:
        - Initialize variables **sJob** and **sPriority**.
        - Find the suitable job based on priority that meets the following criteria:
            - Arrival time is less than or equal to the current time.
            - Priority is the highest among the available processes.
            - The process hasn't finished executing yet.
        - If no suitable job is found, increment the **currentTime**.
        - If a suitable job is found:
            - Execute the selected job by decrementing its remaining time.
            - Update **currentTime**.
            - If the remaining time for the job becomes zero:
                - Update completion time, turnaround time, and waiting time.
                - Increment the **completed** count.
7. Calculate average waiting time (**avgWT**) and average turnaround time (**avgTAT**).
8. Display process details and averages:
    - Print the details of each process: id, arrival time, burst time, priority, completion time, turnaround time, and waiting time.
    - Display the calculated average waiting time and average turnaround time.
9. **Output**: Display the process details and calculated averages.
10. Stop the program

OUTPUT

```
Enter the number of processes: 3
Enter the id, arrival time, burst time, and priority of 3processes:
1 0 5 3
2 3 3 1
3 5 2 2

ID      AT      BT      Prio    CT      TAT     WT
P1      0       5       3       10      10      5
P2      3       3       1       6       3       0
P3      5       2       2       8       3       1
Average Waiting Time = 2.000000
Average Turnaround Time = 5.333333
```

PROGRAM

```
#include <stdio.h>
typedef struct {
int id, at, bt, ct, tat, wt, rt, priority;
} process;
int main() {
int n, currentTime = 0;
float avgTAT, avgWT;
printf("Enter the number of processes: ");
scanf("%d", &n);
process a[n];
printf("Enter the id, arrival time, burst time, and priority of %dprocesses:\n", n);
for (int i = 0; i < n; i++) {
scanf("%d", &a[i].id);
scanf("%d", &a[i].at);
scanf("%d", &a[i].bt);
scanf("%d", &a[i].priority);
a[i].rt = a[i].bt;
}
int completed = 0;
int totalWT = 0;
int totalTAT = 0;
while (completed < n) {
int sJob = -1;
int sPriority = 99999;
for (int i = 0; i < n; i++) {
if (a[i].at <= currentTime && a[i].priority < sPriority &&
a[i].rt > 0) {
sJob = i;
sPriority = a[i].priority;
}
}
if (sJob == -1) {
currentTime++;
} else {
a[sJob].rt--;
currentTime++;
if (a[sJob].rt == 0) {
```

```
completed++;
a[sJob].ct = currentTime;
a[sJob].tat = currentTime - a[sJob].at;
a[sJob].wt = a[sJob].tat - a[sJob].bt;
totalTAT += a[sJob].tat;
totalWT += a[sJob].wt;
}
}
}
avgTAT = (float)totalTAT / n;
avgWT = (float)totalWT / n;
printf("\nID\tAT\tBT\tPrio\tCT\tTAT\tWT");
for (int i = 0; i < n; i++) {
printf("\nP%d\t%d\t%d\t%d\t%d\t%d\t%d", a[i].id, a[i].at, a[i].bt,
a[i].priority, a[i].ct, a[i].tat, a[i].wt);
}
printf("\nAverage Waiting Time = %f", avgWT);
printf("\nAverage Turnaround Time = %f\n", avgTAT);
return 0;
}
```

RESULT

C program to implement Priority scheduling algorithm has been executed successfully .

DATE :

EXPERIMENT NO – 1.2(i)
SEQUENTIAL FILE ALLOCATION

AIM

To simulate the sequential file allocation strategy .

ALGORITHM

```
1. Start the program
2. Declare variables: n, i, j, x as integers, b[20], sb[20], t[20], c[20][20] as integer arrays
3. Read the number of files 'n' from the user
4. For i = 0 to n-1:
   a. Read the number of blocks occupied by file[i] from the user and store it in b[i]
   b. Read the starting block of file[i] from the user and store it in sb[i]
   c. Set t[i] = sb[i]
   d. For j = 0 to b[i]-1:
      i. Assign sb[i] to c[i][j] and increment sb[i] by 1
5. Display the table header: "Filename    Start block    Length"
6. For i = 0 to n-1:
   a. Display file details: file[i]+1, t[i], b[i]
7. Read the file name 'x' from the user
8. Display "File name is x" and "Length is b[x-1]"
9. Display "Blocks occupied:" followed by blocks stored in c[x-1][i] for i = 0 to b[x-1]-1
10. End
```

PROGRAM

```
#include<stdio.h>
void main()
{
 int n,i,j,b[20],sb[20],t[20],x,c[20][20];
 printf("Enter the no.of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 printf("Enter the number of blocks occupied by file%d:",i+1);
 scanf("%d",&b[i]);
 printf("Enter the starting block of file%d:",i+1);
 scanf("%d",&sb[i]);
```

OUTPUT

```
Enter the no.of files:3
Enter the number of blocks occupied by file1:3
Enter the starting block of file1:5
Enter the number of blocks occupied by file2:2
Enter the starting block of file2:9
Enter the number of blocks occupied by file3:3
Enter the starting block of file3:15
Filename        Start block      length
1                    5              3
2                    9              2
3                   15              3
Enter file name:2
File name is 2
Length is 2
Blocks occupied:   9  10
```

```
t[i]=sb[i];
for(j=0;j<b[i];j++)
{
c[i][j]=sb[i]++;
}
}
printf("Filename\tStart block\tlength\n");
for(i=0;i<n;i++)
printf("%d\t %d\t %d\n",i+1,t[i],b[i]);
printf("Enter file name:");
scanf("%d",&x);
printf("File name is %d\n",x);
printf("Length is %d\n",b[x-1]);
printf("Blocks occupied:");
for(i=0;i<b[x-1];i++)
printf("%4d",c[x-1][i]);
}
```

RESULT

Program to implement sequential file allocation strategy implemented successfully .

DATE :

EXPERIMENT NO – 1.2(ii)
INDEXED FILE ALLOCATION

AIM

To simulate the indexed file allocation strategy .

ALGORITHM

```
1. Start the program
2. Declare variables: n, m[20], i, j, sb[20], s[20], b[20][20], x
3. Display "Enter no. of files:"
4. Input the number of files n
5. Loop from i = 0 to n - 1:
      ➢ Display "Enter starting block and size of file i + 1:"
      ➢ Input starting block sb[i] and size s[i] of the file
      ➢ Display "Enter blocks occupied by file i + 1:"
      ➢ Input the number of blocks occupied by the file m[i]
      ➢ Display "Enter blocks of file i + 1:"
      ➢ Loop from j = 0 to m[i] - 1:
            • Input block numbers b[i][j] occupied by the file
6. Display "\nFile\t index\tlength"
7. Loop from i = 0 to n - 1:
      ➢ Display file index, starting block, and length: i + 1, sb[i], m[i]
8. Display "Enter file name:"
9. Input the file name x
10. Display "file name is: x"
11. Set i = x - 1
12. Display "Index is: sb[i]"
13. Display "Block occupied are:"
14. Loop from j = 0 to m[i] - 1:
      ➢ Display blocks occupied by file b[i][j]
15. End
```

PROGRAM

```
#include<stdio.h>
void main()
{
 int n,m[20],i,j,sb[20],s[20],b[20][20],x;
```

OUTPUT

```
Enter no. of files:2
Enter starting block and size of file1:2 4
Enter blocks occupied by file1:4
enter blocks of file1:7
6
8
4
Enter starting block and size of file2:3 5
Enter blocks occupied by file2:3
enter blocks of file2:1
5
9

File        index   length
1            2        4
2            3        3

Enter file name:2
file name is:2
Index is:3Block occupied are:  1  5  9
```

```
 printf("Enter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 printf("Enter starting block and size of file%d:",i+1);
 scanf("%d%d",&sb[i],&s[i]);
 printf("Enter blocks occupied by file%d:",i+1);
 scanf("%d",&m[i]);
 printf("enter blocks of file%d:",i+1);
 for(j=0;j<m[i];j++)
 scanf("%d",&b[i][j]);
 }
 printf("\nFile\t index\tlength\n");
 for(i=0;i<n;i++)
 printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);
 printf("\nEnter file name:");
 scanf("%d",&x);
 printf("file name is:%d\n",x);
 i=x-1;
 printf("Index is:%d",sb[i]);
 printf("Block occupied are:");
 for(j=0;j<m[i];j++)
 printf("%3d",b[i][j]);
}
```

<u>RESULT</u>

Program to implement indexed file allocation strategy implemented successfully .

DATE :

EXPERIMENT NO – 1.2(iii)
LINKED FILE ALLOCATION

AIM

To simulate the linked file allocation strategy .

ALGORITHM

1. Start the program
2. Structures Defined:
    - Block: Contains data and a pointer to the next block.
    - File: Contains a name and a pointer to the first block.
3. Function createFile:
    - Check if the number of files has not exceeded the limit.
    - Allocate memory for a new file.
    - Initialize the file's name and set its first block to NULL.
    - Store the file in the array of files and increment the file count.
    - Print a success message.
4. Function allocateBlock:
    - Allocate memory for a new block.
    - Set the block's data to the provided block number and its next pointer to NULL.
    - If the file doesn't have any blocks yet, assign the new block as the first block.
    - Otherwise, traverse the blocks of the file to the last one and append the new block.
    - Print a message indicating the block allocation.
5. Function displayFiles:
    - Check if no files are created; if so, print a message indicating the absence of files.
    - Otherwise, iterate through each file in the array:
        - Print the file name.
        - Traverse the blocks of the file and print their data.
        - Print a newline.
6. main Function:
    - Initialize an array to hold file pointers and other necessary variables.
    - Start an infinite loop to display a menu and take user input.
    - Provide options to:
        - Create a new file, taking its name as input.
        - Allocate a block to an existing file by index.
        - Display all created files and their allocated blocks.

OUTPUT

```
Linked File Allocation Simulation
1. Create a new file
2. Allocate a block to a file
3. Display files and allocated blocks
4. Exit
Enter your choice: 1
Enter the name of the new file: CSA
File 'CSA' created successfully.

Linked File Allocation Simulation
1. Create a new file
2. Allocate a block to a file
3. Display files and allocated blocks
4. Exit
Enter your choice: 2
Enter the index of the file (0-0): 0
Block 1 allocated to file 'CSA'.

Linked File Allocation Simulation
1. Create a new file
2. Allocate a block to a file
3. Display files and allocated blocks
4. Exit
Enter your choice: 3
List of files:
File 'CSA': Blocks -> 1

Linked File Allocation Simulation
1. Create a new file
2. Allocate a block to a file
3. Display files and allocated blocks
4. Exit
Enter your choice: 2
Enter the index of the file (0-0): 0
Block 2 allocated to file 'CSA'.

Linked File Allocation Simulation
```

- Exit the program.

7. Menu Implementation in main:
    - ➢ Display a menu to the user.
    - ➢ Accept user choice using scanf.
    - ➢ Use a switch-case statement to perform actions based on the user's choice:
        - Create a file with the given name.
        - Allocate a block to a specified file (if files exist).
        - Display all files and their allocated blocks.
        - Exit the program if the user chooses.
8. Error Handling:
    - ➢ Check for invalid inputs (such as invalid file indices or invalid choices).
    - ➢ Display appropriate error messages.
9. Loop Continuity:
    - ➢ Keep the program running until the user chooses to exit.
10. End

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
struct Block {
  int data;
  struct Block* next;
};
struct File {
  char name[20];
  struct Block* firstBlock;
};
void createFile(struct File** files, int* numFiles, char* name) {
  if (*numFiles >= 50) {
    printf("Cannot create more files. Limit reached.\n");
    return;
  }
  struct File* newFile = (struct File*)malloc(sizeof(struct File));
  if (!newFile) {
    printf("Memory allocation failed.\n");
    return;
  }
```

```
1. Create a new file
2. Allocate a block to a file
3. Display files and allocated blocks
4. Exit
Enter your choice: 2
Enter the index of the file (0-0): 0
Block 3 allocated to file 'CSA'.

Linked File Allocation Simulation
1. Create a new file
2. Allocate a block to a file
3. Display files and allocated blocks
4. Exit
Enter your choice: 3
List of files:
File 'CSA': Blocks -> 1 2 3

Linked File Allocation Simulation
1. Create a new file
2. Allocate a block to a file
3. Display files and allocated blocks
4. Exit
Enter your choice: 4
```

```
    strcpy(newFile->name, name);
    newFile->firstBlock = NULL;
    files[*numFiles] = newFile;
    (*numFiles)++;
    printf("File '%s' created successfully.\n", name);
}
void allocateBlock(struct File* file, int blockNumber) {
    struct Block* newBlock = (struct Block*)malloc(sizeof(struct Block));
    if (!newBlock) {
        printf("Memory allocation failed.\n");
        return;
    }
    newBlock->data = blockNumber;
    newBlock->next = NULL;
    if (!file->firstBlock) {
        file->firstBlock = newBlock;
    } else {
        struct Block* current = file->firstBlock;
        while (current->next) {
            current = current->next;
        }
        current->next = newBlock;
    }
    printf("Block %d allocated to file '%s'.\n", blockNumber, file->name);
}
void displayFiles(struct File** files, int numFiles) {
    if (numFiles == 0) {
        printf("No files created yet.\n");
        return;
    }
    printf("List of files:\n");
    for (int i = 0; i < numFiles; i++) {
        struct File* file = files[i];
        printf("File '%s': Blocks -> ", file->name);
        struct Block* current = file->firstBlock;
        while (current) {
            printf("%d ", current->data);
            current = current->next;
```

```
        }
        printf("\n");
    }
}
int main() {
    struct File* files[50];
    int numFiles = 0;
    int blockNumber = 1;
    int choice;
    char fileName[20];
    while (1) {
        printf("\nLinked File Allocation Simulation\n");
        printf("1. Create a new file\n");
        printf("2. Allocate a block to a file\n");
        printf("3. Display files and allocated blocks\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the name of the new file: ");
                scanf("%s", fileName);
                createFile(files, &numFiles, fileName);
                break;
            case 2:
                if (numFiles == 0) {
                    printf("No files created yet. Please create a file first.\n");
                } else {
                    int fileIndex;
                    printf("Enter the index of the file (0-%d): ", numFiles - 1);
                    scanf("%d", &fileIndex);
                    if (fileIndex >= 0 && fileIndex < numFiles) {
                        allocateBlock(files[fileIndex], blockNumber);
                        blockNumber++;
                    } else {
                        printf("Invalid file index.\n");
                     }
                }
```

```
            break;
          case 3:
            displayFiles(files, numFiles);
            break;
          case 4:
            exit(0);
          default:
            printf("Invalid choice. Please enter a valid option.\n");
        }
    }
    return 0;
}
```

RESULT

Program to implement linked file allocation strategy implemented successfully .

DATE :

EXPERIMENT NO – 1.3(i)
FILE ORGANIZATION – SINGLE LEVEL DIRECTORY

AIM

To simulate the single level directory file organization technique .

ALGORITHM

1. **Variables Initialization:**
    - Initialize variables: **nf** (number of files), **i** (loop iterator), **ch** (choice), **mdname** (directory name), **fname** (array to store file names).
2. **Input Directory Name:**
    - Prompt the user to input a directory name using **printf**.
    - Use **scanf** to read and store the directory name in the variable **mdname**.
3. **File Creation Loop:**
    - Start a do-while loop to repeatedly create files based on user input.
    - Inside the loop:
        - Prompt the user to enter the name of the file to be created using **printf**.
        - Read the file name entered by the user using **scanf** and store it in the variable **name**.
        - Iterate through the existing file names stored in the **fname** array to check if the entered file name already exists.
            - Use a **for** loop to check each element of the **fname** array.
            - Compare the entered file name (**name**) with existing file names (**fname[i]**) using **strcmp**.
            - If a match is found (**strcmp** returns 0), break the loop.
        - If the entered file name doesn't match any existing file name:
            - Store the new file name in the **fname** array at index **nf**.
            - Increment **nf** (number of files) to keep track of the number of files created.
        - If the entered file name matches an existing file name, display an appropriate message indicating the file already exists.
        - Prompt the user to choose whether to enter another file (yes - 1 or no - 0) using **printf**.
        - Read the user's choice using **scanf** and store it in the variable **ch**.
4. **Display Directory Name and File Names:**
    - After the loop exits (when the user chooses not to enter another file), display the directory name and the list of created file names.

<u>OUTPUT</u>

```
Enter the directory name: dir1
Enter file name to be created:file1
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created:file2
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created:file3
Do you want to enter another file(yes - 1 or no - 0):1
Enter file name to be created:file1
There is already file1
Do you want to enter another file(yes - 1 or no - 0):0
Directory name is:dir1
Files names are...
file1
file2
file3
```

- Print the directory name (**mdname**) using **printf**.
- Display a message indicating the list of file names using **printf**.
- Use a **for** loop to iterate through the **fname** array and print each file name (**fname[i]**) on a new line using **printf**.

5. **End of Program**

PROGRAM

```
#include<stdio.h>
#include<string.h>
void main()
{
int nf=0,i=0,ch;
char mdname[10],fname[10][10],name[10];
 printf("Enter the directory name: ");
scanf("%s",mdname);
 do
{
printf("Enter file name to be created:");
scanf("%s",name);
for(i=0;i<nf;i++)
{
if(!strcmp(name,fname[i]))
break;
}
if(i==nf)
strcpy(fname[nf++],name);
 else
printf("There is already %s\n",name);
printf("Do you want to enter another file(yes - 1 or no - 0):");
scanf("%d",&ch);
} while(ch==1);
printf("Directory name is:%s\n",mdname);
printf("Files names are...\n");
for(i=0;i<nf;i++)
printf("%s\n",fname[i]);
}
```

RESULT

Program to implement single level file organization technique has been executed successfully .

DATE :

EXPERIMENT NO – 1.3(ii)
FILE ORGANIZATION – TWO LEVEL DIRECTORY

AIM

To simulate the two level directory file organization technique .

ALGORITHM

1. Start of the program
2. Structure Definition:
    - Define a structure to hold directory names (dname), file names (fname), and file count (fcnt).
3. Variables Initialization:
    - Initialize variables: dir (an array of structures to hold directories and files), dcnt (directory count), i, ch, k, f, d.
4. Menu-driven System:
    - Start an infinite loop (while (1)) to display a menu and take user input.
    - Display a menu with options for directory and file operations: create directory, create file, delete file, search file, display, and exit.
    - Use scanf to accept the user's choice (ch).
5. Switch-case Statement:
    - Use a switch statement to perform actions based on the user's choice:
    - Case 1 (Create Directory):
        - Prompt the user to enter the name of the directory.
        - Read the directory name using scanf and store it in dir[dcnt].dname.
        - Initialize the file count (dir[dcnt].fcnt) for this directory to zero.
        - Increment dcnt (directory count) to keep track of the number of directories created.
        - Display a message indicating the directory creation.
    - Case 2 (Create File):
        - Prompt the user to enter the directory name.
        - Read the directory name using scanf and store it in the variable d.
        - Search for the directory in the dir array.
        - If found, prompt the user to enter the name of the file to create within that directory.
        - Read the file name using scanf and store it in the appropriate directory's file list (dir[i].fname[dir[i].fcnt]).
        - Increment the file count (dir[i].fcnt) for that directory.

<u>OUTPUT</u>

```
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:1

Enter Name of Directory:dir1

Directory created
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:2

Enter Name of the Directory:dir1

Enter Name of the File to Create:file1

File created
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:2

Enter Name of the Directory:dir1

Enter Name of the File to Create:file2
```

        - Display a message indicating the file creation.
    - Case 3 (Delete File):
        - Prompt the user to enter the directory name.
        - Read the directory name using scanf and store it in the variable d.
        - Search for the directory in the dir array.
        - If found, prompt the user to enter the name of the file to delete within that directory.
        - Search for the file in the directory's file list.
        - If found, delete the file by shifting the elements in the array and decrementing the file count.
        - Display a message indicating the file deletion.
    - Case 4 (Search File):
        - Prompt the user to enter the directory name.
        - Read the directory name using scanf and store it in the variable d.
        - Search for the directory in the dir array.
        - If found, prompt the user to enter the name of the file to search within that directory.
        - Search for the file in the directory's file list.
        - Display a message indicating whether the file is found or not.
    - Case 5 (Display):
        - Display the list of directories and their respective files (if any).
        - If no directories exist, display a message indicating the absence of directories.
        - Iterate through each directory in the dir array and print its name along with the associated file names.
    - Default Case and Exit:
        - If the user selects an invalid option or chooses to exit, the program terminates using the exit(0) function.

6. End of Program

PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct
{
 char dname[10], fname[10][10];
 int fcnt;
```

```
File created
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:2

Enter Name of the Directory:dir1

Enter Name of the File to Create:file2

File created
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:2

Enter Name of the Directory:dir2

Directory dir2 Not Found!
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:1

Enter Name of Directory:dir2

Directory created
1.Create Directory
2.Create File
```

```
} dir[10];
void main()
{
 int i, ch, dcnt, k;
 char f[30], d[30];
 dcnt = 0;
 while (1)
 {
 printf("\n1.Create Directory\n2.Create File\n3.Delete File\n4.Search
File\n5.Display\n6.Exit\nEnter Your Choice:");
 scanf("%d", &ch);
 switch (ch)
 {
 case 1:
 printf("\nEnter Name of Directory:");
 scanf("%s", dir[dcnt].dname);
 dir[dcnt].fcnt = 0;
 dcnt++;
 printf("\nDirectory created");
 break;
 case 2:
 printf("\nEnter Name of the Directory:");
 scanf("%s", d);
 for (i = 0; i < dcnt; i++)
 if (strcmp(d, dir[i].dname) == 0)
 {
 printf("\nEnter Name of the File to Create:");
 scanf("%s", dir[i].fname[dir[i].fcnt]);
 dir[i].fcnt++;
 printf("\nFile created");
 break;
 }
 if (i == dcnt)
 printf("\nDirectory %s Not Found!", d);
 break;
 case 3:
 printf("\nEnter Name of the Directory:");
 scanf("%s", d);
```

```
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:2

Enter Name of the Directory:dir2

Enter Name of the File to Create:file3

File created
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:5

Directory       Files
dir1            file1    file2    file2
dir2            file3
1.Create Directory
2.Create File
3.Delete File
4.Search File
5.Display
6.Exit
Enter Your Choice:4

Enter Name of the Directory:dir1

Enter the Name of the File to Search:file2

File file2 Found
1.Create Directory
2.Create File
3.Delete File
```

```
for (i = 0; i < dcnt; i++)
{
if (strcmp(d, dir[i].dname) == 0)
{
printf("\nEnter Name of the File to Delete:");
scanf("%s", f);
for (k = 0; k < dir[i].fcnt; k++)
{
if (strcmp(f, dir[i].fname[k]) == 0)
{
printf("\nFile %s Deleted", f);
dir[i].fcnt--;
strcpy(dir[i].fname[k], dir[i].fname[dir[i].fcnt]);
goto jmp;
}
}
printf("\nFile %s Not Found!", f);
goto jmp;
}
}
printf("\nDirectory %s Not Found!", d);
jmp:
break;
case 4:
printf("\nEnter Name of the Directory:");
scanf("%s", d);
for (i = 0; i < dcnt; i++)
{
if (strcmp(d, dir[i].dname) == 0)
{
printf("\nEnter the Name of the File to Search:");
scanf("%s", f);
for (k = 0; k < dir[i].fcnt; k++)
{
if (strcmp(f, dir[i].fname[k]) == 0)
{
printf("\nFile %s Found", f);
goto jmp1;
```

```
4.Search File
5.Display
6.Exit
Enter Your Choice:6
```

```
 }
 }
 printf("\nFile %s Not Found!", f);
 goto jmp1;
 }
 }
 printf("\nDirectory %s Not Found!", d);
 jmp1:
 break;
 case 5:
 if (dcnt == 0)
 printf("\nNo Directories!");
 else
 {
 printf("\nDirectory\tFiles");
 for (i = 0; i < dcnt; i++)
 {
 printf("\n%s\t\t", dir[i].dname);
 for (k = 0; k < dir[i].fcnt; k++)
 printf("\t%s", dir[i].fname[k]);
 }
 }
 break;
 default:
 exit(0);
 }
 }
}
```

RESULT

Program to implement two level file organization technique has been executed successfully .

DATE :

EXPERIMENT NO – 1.3(iii)
FILE ORGANIZATION – HIERARCHICAL

AIM

To simulate the hierarchical file organization technique .

ALGORITHM

1. Start of program
2. **Structures Defined:**
    - **File**: Contains a name and content.
    - **Directory**: Contains a name, arrays for subdirectories and files, counters for the number of subdirectories and files.
3. **Function createDirectory:**
    - Check if the number of subdirectories in the parent directory has not reached the limit (**MAX_FILES**).
    - Allocate memory for a new directory.
    - Initialize the new directory's name, number of subdirectories, and number of files.
    - Add the new directory to the parent's subdirectory list and increment the count of subdirectories.
    - Print a message indicating successful directory creation or an error if the limit is reached.
4. **Function createFile:**
    - Check if the number of files in the parent directory has not reached the limit (**MAX_FILES**).
    - Create a new file structure with a given name and content.
    - Add the new file to the parent directory's file list and increment the count of files.
    - Print a message indicating successful file creation or an error if the limit is reached.
5. **Function listContents:**
    - Display the contents of a directory:
        - Print the directory's name.
        - Iterate through the subdirectories and print their names.
        - Iterate through the files and print their names.
6. **main Function:**
    - Create a root directory.

<u>OUTPUT</u>

```
Enter the name of the directory: dir1
Directory 'dir1' created.
Enter the name of a file: file1
Enter the content of the file: hello i am CSA
File 'file1' created.
Contents of directory 'root':
Subdirectories:
- dir1
Files:
- file1
```

- Prompt the user to input the name of a directory (**dirName**) and create it using **createDirectory**.
- Prompt the user to input the name of a file (**fileName**) and its content (**fileContent**), then create the file using **createFile**.
- Display the contents of the root directory using **listContents**.

7. **Input Validation and Display:**
    - The program takes user input for directory and file creation.
    - It checks for limitations on the number of subdirectories and files.
    - Displays the created directory and its contents including subdirectories and files.
8. **End of program**

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_NAME_LENGTH 50
#define MAX_FILES 100
typedef struct File {
char name[MAX_NAME_LENGTH];
char content[1024];
} File;
typedef struct Directory {
char name[MAX_NAME_LENGTH];
struct Directory* subdirectories[MAX_FILES];
File files[MAX_FILES];
int num_subdirectories;
int num_files;
} Directory;
void createDirectory(Directory* parent, const char* name) {
if (parent->num_subdirectories < MAX_FILES) {
Directory* newDirectory = (Directory*)malloc(sizeof(Directory));
strcpy(newDirectory->name, name);
newDirectory->num_subdirectories = 0;
newDirectory->num_files = 0;
parent->subdirectories[parent->num_subdirectories++] = newDirectory;
 printf("Directory '%s' created.\n", name);
 } else
printf("Directory limit reached. Cannot create '%s'.\n", name);
```

```
}
void createFile(Directory* parent, const char* name, const char* content) {
 if (parent->num_files < MAX_FILES) {
File newFile;
strcpy(newFile.name, name);
strcpy(newFile.content, content);
 parent->files[parent->num_files++] = newFile;
printf("File '%s' created.\n", name);
 } else
 printf("File limit reached. Cannot create '%s'.\n", name);
}
void listContents(Directory* dir) {
 printf("Contents of directory '%s':\n", dir->name);
 printf("Subdirectories:\n");
 for (int i = 0; i < dir->num_subdirectories; i++) {
 printf("- %s\n", dir->subdirectories[i]->name);
 }
 printf("Files:\n");
 for (int i = 0; i < dir->num_files; i++) {
printf("- %s\n", dir->files[i].name);
 }
}
int main() {
 Directory root;
 strcpy(root.name, "root");
 root.num_subdirectories = 0;
 root.num_files = 0;
 char dirName[MAX_NAME_LENGTH];
 printf("Enter the name of the directory: ");
 scanf("%s", dirName);
 createDirectory(&root, dirName);
 char fileName[MAX_NAME_LENGTH];
 printf("Enter the name of a file: ");
 scanf("%s", fileName);
 char fileContent[1024];
 printf("Enter the content of the file: ");
 scanf("%s", fileContent);
 createFile(&root, fileName, fileContent);
```

```
 listContents(&root);
 return 0;
}
```

RESULT

Program to implement hierarchical file organization technique has been executed successfully .

DATE :

EXPERIMENT NO – 1.4(i)
DISK SCHEDULING ALGORITHM - FCFS

AIM

To simulate First Come First Serve disk scheduling algorithm .

ALGORITHM

1. Start of program
2. **Variable Initialization:**
    - Declare variables: **diskQueue** (an array to hold disk queue elements), **n** (size of the queue), **i** (loop iterator), **seekTime** (total seek time), **diff** (difference between disk positions).
3. **Input Handling:**
    - Prompt the user to enter the size of the queue (**n**) using **printf**.
    - Read the size of the queue from the user using **scanf**.
    - Prompt the user to enter the queue elements using **printf**.
    - Read the queue elements into the **diskQueue** array using a loop and **scanf**.
    - Prompt the user to enter the initial head position using **printf**.
    - Read the initial head position into **diskQueue[0]** using **scanf**.
4. **Disk Movement Calculation:**
    - Initialize **seekTime** to zero.
    - Print a header indicating the movement of cylinders using **printf**.
    - Iterate through the queue elements to calculate disk movements and seek times:
        - Calculate the absolute difference between consecutive disk positions (**diskQueue[i+1]** and **diskQueue[i]**) and store it in **diff**.
        - Add **diff** to the **seekTime** variable to accumulate total seek time.
        - Print the movement from one disk position to another along with the seek time using **printf**.
5. **Display Results:**
    - Print the total seek time using **printf**.
    - Calculate and print the average seek time by dividing the total seek time by the number of elements (**n**) in the queue using **printf**.
6. **End of Program:**

PROGRAM

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

OUTPUT

```
Enter the size of Queue: 6
Enter the Queue: 45
67
12
34
11
89
Enter the initial head position: 40

Movement of Cylinders
Move from 40 to 45 with seek time 5
Move from 45 to 67 with seek time 22
Move from 67 to 12 with seek time 55
Move from 12 to 34 with seek time 22
Move from 34 to 11 with seek time 23
Move from 11 to 89 with seek time 78

Total Seek Time: 205
Average Seek Time = 34.166668
```

```
int main() {
   int diskQueue[20], n, i, seekTime=0, diff;
   printf("Enter the size of Queue: ");
   scanf("%d", &n);
   printf("Enter the Queue: ");
   for(i=1;i<=n;i++) {
          scanf("%d",&diskQueue[i]);
   }
   printf("Enter the initial head position: ");
   scanf("%d", &diskQueue[0]);
   printf("\nMovement of Cylinders\n");
   for(i=0;i<n;i++) {
      diff= abs(diskQueue[i+1] - diskQueue[i]);
      seekTime+= diff;
      printf("Move from %d to %d with seek time %d\n", diskQueue[i], diskQueue[i+1], diff);
   }
   printf("\nTotal Seek Time: %d", seekTime);
   printf("\nAverage Seek Time = %f",(float) seekTime/n);
   printf("\n");
   return 0;
}
```

RESULT

Program to implement First Come First Serve disk scheduling algorithm has been implemented successfully .

DATE :

EXPERIMENT NO – 1.4(ii)
DISK SCHEDULING ALGORITHM - SCAN

AIM

To simulate SCAN disk scheduling algorithm .

ALGORITHM

1. Start of program
2. **Function Definitions:**
    - **scan(int Ar[20], int n, int start)**: Performs the disk arm movement scan.
    - **sort(int Ar[20], int n)**: Sorts the disk queue.
3. **Main Function:**
    - Declare variables: **diskQueue[20]** (to store disk queue), **n** (size of the queue), **start** (initial head position), **i**.
    - Input the size of the queue (**n**) from the user using **scanf**.
    - Input the queue elements into **diskQueue** using a loop and **scanf**.
    - Input the initial head position (**start**) using **scanf**.
    - Increment the size of the queue by 1 and set **diskQueue[0]** to the initial head position.
    - Sort the **diskQueue** using the **sort** function.
    - Perform the scan using the **scan** function with **diskQueue**, **n**, and **start** as parameters.
4. **scan Function:**
    - Declare variables: **i**, **pos**, **diff**, **seekTime**, **current**.
    - Find the position of the initial head position in the queue (**start**).
    - Iterate from the initial head position to the end of the queue, calculating seek times and printing movement details.
    - Then, iterate from the position before the initial head position to the start of the queue, calculating seek times and printing movement details.
    - Calculate and print the total seek time and average seek time based on the movements.
5. **sort Function:**
    - Sorts the **Ar** array in ascending order using the Bubble Sort algorithm.
6. **End of Program:**
    - The program concludes after performing the scan and displaying the movement details, total seek time, and average seek time.

OUTPUT

```
Enter the size of Queue: 6
Enter the Queue: 34
98
12
43
11
10
Enter the initial head position: 35

Movement of Cylinders
Move from 35 to 43 with seek time 8
Move from 43 to 98 with seek time 55
Move from 98 to 34 with seek time 64
Move from 34 to 12 with seek time 22
Move from 12 to 11 with seek time 1
Move from 11 to 10 with seek time 1

Total Seek Time: 151
Average Seek Time = 25.166666
```

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
void scan(int Ar[20], int n, int start);
void sort(int Ar[20], int n);
int main() {
   int diskQueue[20], n, start, i;
   printf("Enter the size of Queue: ");
   scanf("%d", &n);
   printf("Enter the Queue: ");
   for(i=1;i<=n;i++) {
          scanf("%d",&diskQueue[i]);
   }
   printf("Enter the initial head position: ");
   scanf("%d", &start);
   diskQueue[0] = start;
   ++n;
   sort(diskQueue, n);
   scan(diskQueue, n, start);
   return 0;
}
void scan(int Ar[20], int n, int start) {
   int i, pos, diff, seekTime=0, current;
   for(i=0;i<n;i++) {
      if(Ar[i]==start) {
         pos=i;
         break;
      }
   }
   printf("\nMovement of Cylinders\n");
   for(i=pos;i<n-1;i++) {
      diff = abs(Ar[i+1] - Ar[i]);
      seekTime += diff;
      printf("Move from %d to %d with seek time %d\n", Ar[i], Ar[i+1], diff);
   }
   current=i;
   for(i=pos-1;i>=0;i--) {
      diff = abs(Ar[current] - Ar[i]);
```

```
        seekTime += diff;
        printf("Move from %d to %d with seek time %d\n", Ar[current], Ar[i], diff);
        current=i;
    }
    printf("\nTotal Seek Time: %d", seekTime);
    printf("\nAverage Seek Time = %f",(float) seekTime/(n-1));
    printf("\n");
}
void sort(int Ar[20], int n) {
    int i, j, tmp;
    for(i=0;i<n-1;i++) {
        for(j=0;j<n-1-i;j++) {
            if(Ar[j]>Ar[j+1]) {
                tmp = Ar[j];
                Ar[j] = Ar[j+1];
                Ar[j+1] = tmp;
            }
        }
    }
}
```

RESULT

Program to implement SCAN disk scheduling algorithm has been implemented successfully .

DATE :

EXPERIMENT NO – 1.4(iii)
DISK SCHEDULING ALGORITHM – C SCAN

AIM

To simulate C - SCAN disk scheduling algorithm .

ALGORITHM

1. Start of program
2. **Include Libraries**: Import necessary libraries like **<stdio.h>** and **<stdlib.h>** for standard input/output and other essential functions.
3. **Function Declarations**:
    - **void cscan(int Ar[20], int n, int start)**: Function to implement the C-SCAN algorithm.
    - **void sort(int Ar[20], int n)**: Function to sort the disk queue in ascending order.
4. **Main Function** (**main()**):
    - Declare variables: **diskQueue[20]** for the disk queue, **n** for the size of the queue, **start** for the initial head position, **i** as an iterator, and **max** for the maximum size.
    - Ask the user to input the size of the queue and the queue elements.
    - Input the initial head position and adjust the queue accordingly.
    - Increment the size of the queue by one (**++n**) and call the **sort()** function to sort the disk queue.
    - Finally, call the **cscan()** function to perform the C-SCAN algorithm on the disk queue.
5. **C-SCAN Function** (**cscan()**):
    - Initialize variables: **i** as an iterator, **pos** to store the position of the initial head, **diff** for the difference between disk positions, **seekTime** to calculate total seek time, and **current** to keep track of the current head position.
    - Find the position of the initial head in the sorted disk queue.
    - Implement the movement of the disk head to the right and then to the left using a loop.
    - Calculate the seek time for each movement and print the details of the movement.
    - Display the total seek time and average seek time.
6. **Sorting Function** (**sort()**):
    - Sort the disk queue in ascending order using a bubble sort algorithm.
7. End of program

OUTPUT

```
Enter the size of Queue: 8
Enter the Queue: 21
90
54
37
29
51
19
31
Enter the initial head position: 20

Movement of Cylinders
Move from 20 to 21 with seek time 1
Move from 21 to 29 with seek time 8
Move from 29 to 31 with seek time 2
Move from 31 to 37 with seek time 6
Move from 37 to 51 with seek time 14
Move from 51 to 54 with seek time 3
Move from 54 to 90 with seek time 36
Move from 90 to 0 with seek time 0
Move from 0 to 19 with seek time 19

Total Seek Time: 89
Average Seek Time = 11.125000
```

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
void cscan(int Ar[20], int n, int start);
void sort(int Ar[20], int n);
int main() {
   int diskQueue[20], n, start, i, max;
   printf("Enter the size of Queue: ");
   scanf("%d", &n);
   printf("Enter the Queue: ");
   for(i=1;i<=n;i++) {
          scanf("%d",&diskQueue[i]);
   }
   printf("Enter the initial head position: ");
   scanf("%d", &start);
   diskQueue[0] = start;
   ++n;
   sort(diskQueue, n);
   cscan(diskQueue, n, start);
   return 0;
}
void cscan(int Ar[20], int n, int start) {
   int i, pos, diff, seekTime=0, current;
   for(i=0;i<n;i++) {
      if(Ar[i]==start) {
         pos=i;
         break;
      }
   }
   printf("\nMovement of Cylinders\n");
   for(i=pos;i<n-1;i++) {
      diff = abs(Ar[i+1] - Ar[i]);
      seekTime+= diff;
      printf("Move from %d to %d with seek time %d\n", Ar[i], Ar[i+1], diff);
   }
   current=0;
   printf("Move from %d to %d with seek time %d\n", Ar[i], current, 0);
   for(i=0;i<pos;i++) {
```

```
        diff = abs(Ar[i] - current);
        seekTime+= diff;
        printf("Move from %d to %d with seek time %d\n", current, Ar[i], diff);
        current = Ar[i];
    }
    printf("\nTotal Seek Time: %d", seekTime);
    printf("\nAverage Seek Time = %f",(float) seekTime/(n-1));
    printf("\n");
}

void sort(int Ar[20], int n) {
    int i, j, tmp;
    for(i=0;i<n-1;i++) {
        for(j=0;j<n-1-i;j++) {
            if(Ar[j]>Ar[j+1]) {
                tmp = Ar[j];
                Ar[j] = Ar[j+1];
                Ar[j+1] = tmp;
            }
        }
    }
}
```

RESULT

Program to implement C – SCAN disk scheduling algorithm has been implemented successfully .

DATE :

EXPERIMENT NO – 1.5(i)
PAGE REPLACEMENT ALGORITHM - FIFO

AIM

To simulate the FIFO page replacement algorithm .

ALGORITHM

1. Start of program
2. **Include Library**: Import the **<stdio.h>** library for standard input/output functions.
3. **Main Function** (**main()**):
    - Declare variables: **i**, **j**, **n**, **a[50]**, **frame[10]**, **no**, **k**, **avail**, and **count**.
    - Ask the user to input the number of pages (**n**) and the page numbers.
    - Input the number of frames (**no**).
    - Initialize the **frame** array with -1 (indicating an empty frame).
    - Set **j** to 0 for frame indexing and print headers for reference string and page frames.
    - Loop through each page number and implement the FIFO page replacement algorithm:
        - Check if the page is already in a frame (**avail == 1**).
        - If the page is not in any frame (**avail == 0**), perform page replacement:
            - Place the page in the current frame (**frame[j] = a[i]**).
            - Increment **j** (frame pointer) using modulo operation to simulate a circular queue.
            - Increment the **count** variable to track page faults.
            - Display the current state of the frames after the page replacement.
4. Display the total number of page faults (**Page Fault Is %d**).
5. Finally, return 0 to indicate successful completion of the program
6. End of program

PROGRAM

```
#include<stdio.h>
int main()
{
int i,j,n,a[50],frame[10],no,k,avail,count=0;
        printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
```

OUTPUT

```
 ENTER THE NUMBER OF PAGES:
8

 ENTER THE PAGE NUMBER :
7
3
1
5
7
7
1
0

 ENTER THE NUMBER OF FRAMES :3
ref string          page frames
7                   7       -1      -1
3                   7       3       -1
1                   7       3       1
5                   5       3       1
7                   5       7       1
7
1
0                   5       7       0
Page Fault Is 6
```

```
        printf("\n ENTER THE PAGE NUMBER :\n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\n ENTER THE NUMBER OF FRAMES :");
        scanf("%d",&no);
for(i=0;i<no;i++)
        frame[i]= -1;
                j=0;
                printf("\tref string\t page frames\n");
for(i=1;i<=n;i++)
                {
                        printf("%d\t\t",a[i]);
                        avail=0;
                        for(k=0;k<no;k++)
if(frame[k]==a[i])
                                avail=1;
                        if (avail==0)
                        {
                                frame[j]=a[i];
                                j=(j+1)%no;
                                count++;
                                for(k=0;k<no;k++)
                                printf("%d\t",frame[k]);
}
                        printf("\n");
}
                printf("Page Fault Is %d",count);
                return 0;
}
```

RESULT

C program to simulate the FIFO page replacement algorithm has been implemented successfully .

DATE :

EXPERIMENT NO – 1.5(ii)
PAGE REPLACEMENT ALGORITHM - LRU

AIM

To simulate the LRU page replacement algorithm .

ALGORITHM

1. Start of program
2. **Include Library**: The program includes the **<stdio.h>** library for standard input/output functions.
3. **Function Declarations**:
    - **void LRU(int[], int[], int[], int, int)**: Function implementing the LRU algorithm.
    - **int findLRU(int[], int)**: Function to find the least recently used frame.
4. **Main Function** (**main()**):
    - Declare variables: **i**, **pCount**, **fCount**, **pages[30]**, **frames[20]**, and **time[20]**.
    - Ask the user to input the number of frames (**fCount**) and the number of pages (**pCount**).
    - Initialize the **frames** array with null values (-1).
    - Input the reference string (pages to be referenced).
5. **LRU Function** (**LRU()**):
    - Initialize variables: **i**, **j**, **k**, **pos**, **flag**, **faultCount**, **counter**, and **queue** to manage the algorithm logic.
    - Loop through each page in the reference string:
        - Check if the page is already in the frames. If found, update the time of occurrence and print "Hit".
        - If the frame is empty and there's space available, insert the page and update the occurrence time.
        - If all frames are occupied, find the least recently used frame using the **findLRU()** function and replace it with the current page.
        - Print the frames after each page reference.
        - Keep track of the number of page faults.
6. **LRU Helper Function** (**findLRU()**):
    - Find the least recently used frame by iterating through the time array and returning the position of the frame with the minimum time of occurrence.
7. Display the total number of page faults (**Total Page Faults = %d**).
8. End of program

OUTPUT

```
Number of Frames : 3
Number of Pages : 9
Enter the reference string
2
7
9
3
2
0
9
2
7
Ref.String   |        Frames
-------------------------------
 2       |        2        -1        -1

 7       |        2        7         -1

 9       |        2        7         9

 3       |        3        7         9

 2       |        3        2         9

 0       |        3        2         0

 9       |        9        2         0

 2       |          Hit

 7       |        9        2         7

Total Page Faults = 8
```

PROGRAM

```
#include <stdio.h>
void LRU(int[], int[], int[], int, int);
int findLRU(int[], int);
int main() {
   int i, pCount, fCount, pages[30], frames[20], time[20];
   printf("Number of Frames : ");
   scanf("%d", &fCount);
   for (i = 0; i < fCount; ++i) {
      frames[i] = -1;
   }
   printf("Number of Pages : ");
   scanf("%d", &pCount);
   printf("Enter the reference string\n");
   for (i = 0; i < pCount; ++i) {
      scanf("%d", &pages[i]);
   }
   LRU(pages, frames, time, fCount, pCount);
   return 0;
}
void LRU(int pages[], int frames[], int time[], int fCount, int pCount) {
   printf("\nRef.String   |\tFrames\n");
   printf("-------------------------------\n");
   int i, j, k, pos, flag, faultCount, counter, queue;
   counter = 0, queue = 0, faultCount = 0;
   for (i = 0; i < pCount; ++i) {
      flag = 0;
      printf("  %d\t|\t", pages[i]);
      for (j = 0; j < fCount; ++j) {
         if (frames[j] == pages[i]) {
            flag = 1;
            counter++;
            time[j] = counter;
            printf("   Hit\n\n");
            break;
         }
      }
      if ((flag == 0) && (queue < fCount)) {
```

```
            faultCount++;
            counter++;
            frames[queue] = pages[i];
            time[queue] = counter;
            queue++;
        }
        else if ((flag == 0) && (queue == fCount)) {
            faultCount++;
            counter++;
            pos = findLRU(time, fCount);
            frames[pos] = pages[i];
            time[pos] = counter;
        }
        if (flag == 0) {
            for (k = 0; k < fCount; ++k) {
                printf("%d  ", frames[k]);
            }
            printf("\n\n");
        }
    }
    printf("Total Page Faults = %d\n\n", faultCount);
}
int findLRU(int time[], int fCount) {
    int k, min, pos;
    pos = 0;
    min = time[0];
    for (k = 1; k < fCount; ++k) {
        if (time[k] < min) {
            min = time[k];
            pos = k;
        }
    }
    return pos;
}
```

RESULT

C program for the simulation of the LRU page replacement algorithm has been implemented successfully .

DATE :

EXPERIMENT NO – 1.5(iii)
PAGE REPLACEMENT ALGORITHM - LFU

AIM

To simulate the LFU page replacement algorithm .

ALGORITHM

1. Start of program
2. Initialize necessary variables: arrays **q**, **p**, **b**, and **c2**, integers **c**, **c1**, **d**, **f**, **i**, **j**, **k**, **n**, **r**, and **t**.
3. Take user input for the number of pages (**n**), the reference string (**p[]**), and the number of frames (**f**).
4. Initialize the first element of **q[]** with the first page number (**p[0]**), print it, increment the count of page faults (**c**), and increment the index **k**.
5. Loop through the reference string (**p[]**) starting from the second element (**i = 1**):
    - ➢ Initialize **c1** as 0.
    - ➢ Check if the current page in **p[]** is not present in **q[]**:
        - Increment **c1** for each page in **q[]** that does not match **p[i]**.

c. If **c1** equals the number of frames (**f**), indicating a page fault:

- ➢ Increment the count of page faults (**c**).
- ➢ If there are available empty frames (**k < f**):
    - Place the page (**p[i]**) into an empty frame in **q[]**, print the frames, and increment **k**.
- ➢ If all frames are occupied (**k >= f**):
    - Calculate the distance till the next occurrence of each page in **q[]** in the remaining reference string.
    - Determine the page in **q[]** that will not be used for the longest duration (**b[]** and **c2[]**).
    - Replace the identified page in **q[]** with the current page (**p[i]**), print the frames.

6. After looping through all pages, print the total number of page faults (**c**).
7. End of program

PROGRAM

```
#include <stdio.h>
void main()
{
  int q[20], p[50], c = 0, c1, d, f, i, j, k = 0, n, r, t, b[20], c2[20];
```

OUTPUT

```
Enter no of pages: 15
Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 0 3 3 7
Enter no of frames: 3

        7
        7       1
        7       1       2
        0       1       2
        0       3       2
        0       3       4
        0       2       4
        3       2       4
        3       2       0
        3       7       0
```

```
printf("Enter no of pages: ");
scanf("%d", &n);
printf("Enter the reference string: ");
for (i = 0; i < n; i++)
   scanf("%d", &p[i]);
printf("Enter no of frames: ");
scanf("%d", &f);
q[k] = p[k];
printf("\n\t%d\n", q[k]);
c++;
k++;
for (i = 1; i < n; i++)
{
   c1 = 0;
   for (j = 0; j < f; j++)
   {
      if (p[i] != q[j])
         c1++;
   }
   if (c1 == f)
   {
      c++;
      if (k < f)
      {
         q[k] = p[i];
         k++;
         for (j = 0; j < k; j++)
            printf("\t%d", q[j]);
         printf("\n");
      }
      else
      {
         for (r = 0; r < f; r++)
         {
            c2[r] = 0;
            for (j = i - 1; j < n; j--)
            {
               if (q[r] != p[j])
```

```
                    c2[r]++;
                  else
                    break;
                }
              }
              for (r = 0; r < f; r++)
                b[r] = c2[r];
              for (r = 0; r < f; r++)
              {
                for (j = r; j < f; j++)
                {
                  if (b[r] < b[j])
                  {
                    t = b[r];
                    b[r] = b[j];
                    b[j] = t;
                  }
                }
              }
              for (r = 0; r < f; r++)
              {
                if (c2[r] == b[0])
                  q[r] = p[i];
                printf("\t%d", q[r]);
              }
              printf("\n");
            }
          }
        }
        printf("\nThe no of page faults is %d", c);
      }
```

RESULT

C program to simulate the LFU page replacement algorithm has been implemented successfully .

EXPERIMENTS FROM ASSEMBLERS , LOADERS AND MACROPROCESSOR

DATE :

EXPERIMENT NO – 2.1
IMPLEMENT PASS 1 OF A TWO PASS ASSEMBLER

AIM

To implement pass 1 of a two pass assembler .

ALGORITHM

1. Start
2. Main Function (main()):
    - Declares variables for label, opcode, and operand.
    - Calls the passOne() function.
3. Pass One (passOne() Function):
    - Declares variables for locctr (location counter), start, and length.
    - Opens files for input, output, symbol table, intermediate code, and length.
    - Reads the first instruction from "input.txt".
    - If the opcode is "START", sets the start address and initializes locctr. Writes the initial line to the intermediate file.
    - Enters a loop until the opcode is "END":
        - Writes intermediate code and updates the symbol table if a label is present.
        - Reads opcode and mnemonic from "optab.txt" and checks for opcode match.
        - Increments locctr based on different opcode types: WORD, RESW, BYTE, RESB.
        - Reads the next instruction from "input.txt".
    - Writes the final instruction to the intermediate file.
    - Closes files and calls the display() function.
    - Calculates the program length and writes it to the length file.
4. Display (display() Function):
    - Opens files for input, intermediate, and symbol table.
    - Displays the contents of these files character by character.
    - Closes the files after display.
5. The passOne() function performs the first pass of the assembler, generating intermediate code and symbol table information . The display() function simply displays the contents of input, intermediate, and symbol table files.
6. Stop

INPUT

input.txt

```
**      START   2000
**      LDA     FIVE
**      STA     ALPHA
**      LDCH    CHARZ
**      STCH    C1
ALPHA   RESW    2
FIVE    WORD    5
CHARZ   BYTE    C'Z'
C1      RESB    1
**      END     **
```

optab.txt

```
LDA    03
STA   0f
LDCH  53
STCH  57
END   *
```

OUTPUT

symtab.txt

```
ALPHA       2012
FIVE        2018
CHARZ       2021
C1          2022
```

intermediate.txt

```
         **          START   2000
2000     **          LDA     FIVE
2003     **          STA     ALPHA
2006     **          LDCH    CHARZ
2009     **          STCH    C1
```

PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void passOne(char label[10], char opcode[10], char operand[10], char code[10], char
mnemonic[3]);
void display();
int main()
{
   char label[10], opcode[10], operand[10];
   char code[10], mnemonic[3];
   passOne(label, opcode, operand, code, mnemonic);
   return 0;
}
void passOne(char label[10], char opcode[10], char operand[10], char code[10], char
mnemonic[3])
{
   int locctr, start, length;
   FILE *fp1, *fp2, *fp3, *fp4, *fp5;
   fp1 = fopen("input.txt", "r");
   fp2 = fopen("optab.txt", "r");
   fp3 = fopen("symtab.txt", "w");
   fp4 = fopen("intermediate.txt", "w");
   fp5 = fopen("length.txt", "w");
   fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);
   if (strcmp(opcode, "START") == 0) {
      start = atoi(operand);
      locctr = start;
      fprintf(fp4, "\t%s\t%s\t%s\n", label, opcode, operand);
      fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);
   }
   else {
     locctr = 0;
   }
   while (strcmp(opcode, "END") != 0) {
      fprintf(fp4, "%d\t%s\t%s\t%s\n", locctr, label, opcode, operand);
      if (strcmp(label, "**") != 0) {
         fprintf(fp3, "%s\t%d\n", label, locctr);
```

```
2012     ALPHA           RESW     2
2018     FIVE            WORD     5
2021     CHARZ           BYTE     C'Z'
2022     C1              RESB     1
2023      **             END      **
```

length.txt

```
23
```

TERMINAL

```
The contents of Input Table :

**            START         2000
**            LDA           FIVE
**            STA           ALPHA
**            LDCH          CHARZ
**            STCH          C1
ALPHA         RESW          2
FIVE          WORD          5
CHARZ         BYTE          C'Z'
C1            RESB          1
**            END           **


The contents of Output Table :

        **            START         2000
2000    **            LDA           FIVE
2003    **            STA           ALPHA
2006    **            LDCH          CHARZ
2009    **            STCH          C1
2012    ALPHA         RESW          2
2018    FIVE          WORD          5
2021    CHARZ         BYTE          C'Z'
2022    C1            RESB          1
2023    **            END           **


The contents of Symbol Table :
        }
        fscanf(fp2, "%s\t%s", code, mnemonic);
        while (strcmp(code, "END") != 0) {
            if (strcmp(opcode, code) == 0) {
                locctr += 3;
                break;
            }
            fscanf(fp2, "%s\t%s", code, mnemonic);
        }
        if (strcmp(opcode, "WORD") == 0) {
            locctr += 3;
        }
        else if (strcmp(opcode, "RESW") == 0) {
            locctr += (3 * (atoi(operand)));
        }
        else if (strcmp(opcode, "BYTE") == 0) {
            ++locctr;
        }
        else if (strcmp(opcode, "RESB") == 0) {
            locctr += atoi(operand);
        }
        fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);
    }
    fprintf(fp4, "%d\t%s\t%s\t%s\n", locctr, label, opcode, operand);
    fclose(fp4);
    fclose(fp3);
    fclose(fp2);
    fclose(fp1);
    display();
    length = locctr - start;
    fprintf(fp5, "%d", length);
    fclose(fp5);
    printf("\nThe length of the code : %d\n", length);
}
void display() {
    char str;
    FILE *fp1, *fp2, *fp3;
    printf("\nThe contents of Input Table :\n\n");
```

```
ALPHA       2012
FIVE        2018
CHARZ       2021
C1          2022

The length of the code : 23
```

```
    fp1 = fopen("input.txt", "r");
    str = fgetc(fp1);
    while (str != EOF) {
        printf("%c", str);
        str = fgetc(fp1);
    }
    fclose(fp1);
    printf("\n\nThe contents of Output Table :\n\n");
    fp2 = fopen("intermediate.txt", "r");
    str = fgetc(fp2);
    while (str != EOF) {
        printf("%c", str);
        str = fgetc(fp2);
    }
    fclose(fp2);
    printf("\n\nThe contents of Symbol Table :\n\n");
    fp3 = fopen("symtab.txt", "r");
    str = fgetc(fp3);
    while (str != EOF) {
        printf("%c", str);
        str = fgetc(fp3);
    }
    fclose(fp3);
}
```

RESULT

C program for the implementation of pass 1 of a two pass assembler is executed successfully .

DATE :

EXPERIMENT NO – 2.2
IMPLEMENT PASS 2 OF A TWO PASS ASSEMBLER

AIM

To implement pass 2 of a two pass assembler .

ALGORITHM

1. Function Declarations:
    - display() function is declared to display the contents of various files.
2. Swap Function:
    - swap() function swaps the values of two characters using pointers.
3. Reverse Function:
    - reverse() function reverses a character buffer from index i to index j.
4. Integer to ASCII Conversion (itoa() Function):
    - itoa() converts an integer value to a string in a specified base.
    - Checks if the base is within the valid range (2 to 32).
    - Converts the absolute value of the number to a string representation in the specified base.
    - Handles negative numbers for base 10 by adding a negative sign.
    - Returns the reversed string representation of the number.
5. Main Function:
    - Declares variables for file pointers, addresses, lengths, symbols, opcodes, and mnemonics.
    - Opens input and output files.
    - Reads the "intermediate.txt" file to determine the final address and sets it as finaddr.
    - Processes the "intermediate.txt" file line by line until the opcode is "END":
        - Handles BYTE, WORD, RESB, RESW, and other operations.
        - Generates object code based on different opcodes and operands.
    - Writes formatted output to "output.txt" and object code to "objcode.txt".
    - Closes files and calls the display() function.
6. Display Function:
    - Opens various files to display their contents:
        - "intermediate.txt", "symtab.txt", "output.txt", and "objcode.txt".
    - Prints the contents of each file character by character.
    - Closes the files after display.

INPUT

intermediate.txt

```
             **              START   2000
2000       **              LDA       FIVE
2003        **              STA      ALPHA
2006     **               LDCH       CHARZ
2009     **                STCH      C1
2012    ALPHA            RESW      2
2018    FIVE               WORD     5
2021    CHARZ           BYTE       C'Z'
2022    C1                  RESB       1
2023     **               END         **
```

symtab.txt

```
ALPHA          2012
FIVE             2018
CHARZ          2021
C1                2022
```

OUTPUT

objcode.txt

```
H^**^002000^0023
T^002000^22^332018^442012^532021^572022^000005^5A
E^002000
```

output.txt

```
                  **      START          2000
2000              **      LDA            FIVE            332018
2003              **      STA            ALPHA           442012
2006              **      LDCH           CHARZ           532021
2009              **      STCH           C1              572022
2012         ALPHA      RESW           2
2018         FIVE       WORD           5               000005
```

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void display();
void swap(char *x, char *y) {
   char t = *x; *x = *y; *y = t;
}
char* reverse(char *buffer, int i, int j)
{
   while (i < j) {
      swap(&buffer[i++], &buffer[j--]);
   }
   return buffer;
}
char* itoa(int value, char* buffer, int base)
{
   if (base < 2 || base > 32) {
      return buffer;
   }
   int n = abs(value);
   int i = 0;
   while (n)
   {
      int r = n % base;
      if (r >= 10) {
         buffer[i++] = 65 + (r - 10);
      }
      else {
         buffer[i++] = 48 + r;
      }
      n = n / base;
   }
   if (i == 0) {
      buffer[i++] = '0';
   }
   if (value < 0 && base == 10) {
      buffer[i++] = '-';
```

```
2021      CHARZ    BYTE        C'Z'        5A
2022       C1      RESB        1
2023       **      END         **
```

TERMINAL

Intermediate file is converted into object code

The contents of Intermediate file:

```
        **          START       2000
2000    **          LDA         FIVE
2003    **          STA         ALPHA
2006    **          LDCH        CHARZ
2009    **          STCH        C1
2012    ALPHA       RESW        2
2018    FIVE        WORD        5
2021    CHARZ       BYTE        C'Z'
2022    C1          RESB        1
2023    **          END         **
```

The contents of Symbol Table :

ALPHA	2012
FIVE	2018
CHARZ	2021
C1	2022

The contents of Output file :

```
        **          START       2000
2000    **          LDA         FIVE        332018
2003    **          STA         ALPHA       442012
2006    **          LDCH        CHARZ       532021
2009    **          STCH        C1          572022
2012    ALPHA       RESW        2
2018    FIVE        WORD        5           000005
```

```
    }
    buffer[i] = '\0';
    return reverse(buffer, 0, i - 1);
}
int main()
{
    char a[10], ad[10], label[10], opcode[10], operand[10], symbol[10];
    int start, diff, 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;
    fp1 = fopen("output.txt", "w");
    fp2 = fopen("symtab.txt", "r");
    fp3 = fopen("intermediate.txt", "r");
    fp4 = fopen("objcode.txt", "w");
    fscanf(fp3, "%s\t%s\t%s", label, opcode, operand);
    while (strcmp(opcode, "END") != 0)
    {
        prevaddr = address;
        fscanf(fp3, "%d%s%s%s", &address, label, opcode, operand);
    }
    finaddr = address;
    fclose(fp3);
    fp3 = fopen("intermediate.txt", "r");
    fscanf(fp3, "\t%s\t%s\t%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^00%s^00%d\n", label, operand, finaddr-atoi(operand));
        fscanf(fp3, "%d%s%s%s", &address, label, opcode, operand);
        start = address;
        diff = prevaddr - start;
        fprintf(fp4, "T^00%d^%d", address, diff);
    }

    while (strcmp(opcode, "END") != 0)
    {
        if (strcmp(opcode, "BYTE") == 0)
```

```
2021    CHARZ       BYTE        C'Z'        5A
2022    C1          RESB        1
2023    **          END         **
```

The contents of Object code file :

```
H^**^002000^0023
T^002000^22^332018^442012^532021^572022^000005^5A
E^002000
```

```
{
   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++)
   {
      itoa(operand[i], ad, 16);
      fprintf(fp1, "%s", ad);
      fprintf(fp4, "%s", ad);
   }
   fprintf(fp1, "\n");
}
else if (strcmp(opcode, "WORD") == 0)
{
   len = strlen(operand);
   itoa(atoi(operand), a, 10);
   fprintf(fp1, "%d\t%s\t%s\t%s\t00000%s\n", address, label, opcode, operand, a);
   fprintf(fp4, "^00000%s", a);
}
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++;
   if (strcmp(operand, "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);
      fprintf(fp1, "%d\t%s\t%s\t%s\t%s%d\n", address, label, opcode, operand, code[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", start);
    fclose(fp4);
    fclose(fp3);
    fclose(fp2);
    fclose(fp1);
    display();
    return 0;
}
void display() {
    char ch;
    FILE *fp1, *fp2, *fp3, *fp4;
    printf("\nIntermediate file is converted into object code");
    printf("\n\nThe contents of Intermediate file:\n\n");
    fp3 = fopen("intermediate.txt", "r");
    ch = fgetc(fp3);
    while (ch != EOF)
    {
        printf("%c", ch);
        ch = fgetc(fp3);
    }
    fclose(fp3);
    printf("\n\nThe contents of Symbol Table :\n\n");
    fp2 = fopen("symtab.txt", "r");
    ch = fgetc(fp2);
    while (ch != EOF)
    {
        printf("%c", ch);
        ch = fgetc(fp2);
    }
    fclose(fp2);
    printf("\n\nThe contents of Output file :\n\n");
    fp1 = fopen("output.txt", "r");
    ch = fgetc(fp1);
```

```
  while (ch != EOF)
  {
    printf("%c", ch);
    ch = fgetc(fp1);
  }
  fclose(fp1);
  printf("\n\nThe contents of Object code file :\n\n");
  fp4 = fopen("objcode.txt", "r");
  ch = fgetc(fp4);
  while (ch != EOF)
  {
    printf("%c", ch);
    ch = fgetc(fp4);
  }
  fclose(fp4);
}
```

RESULT

C program for the implementation of pass 2 of a two pass assembler is executed successfully .

DATE :

EXPERIMENT NO – 2.3
IMPLEMENT A SINGLE PASS MACROPROCESSOR

AIM

To implement a single pass macroprocessor .

ALGORITHM

1. Start the program
2. Open files: **input.txt**, **namtab.txt**, **deftab.txt**, **argtab.txt**, and **op.txt** for reading and writing.
3. Read the input file **input.txt** and process lines until encountering **END**.
4. If the line contains **MACRO**, extract the macro name **la** and its operands **opnd**.
    - ➢ Write the macro name to **namtab.txt**.
    - ➢ Write the macro name and operands to **deftab.txt**.
    - ➢ Process lines until **MEND** is encountered:
        - If the operand starts with **&**, replace it with **?** followed by a positional number (**pos1**).
        - Write the macro instructions and modified operands to **deftab.txt**.

5. If not a macro definition:
    - ➢ Compare the line's mnemonic **mne** with names in **namtab.txt**.
    - ➢ If a match is found, extract and format operands from the line to **argtab.txt**.
    - ➢ Seek to the beginning of **deftab.txt** and **argtab.txt**.
    - ➢ Read the macro instructions and operands:
        - Write the macro name and operands to **op.txt**.
        - Process lines until encountering **MEND**:
            - o If the operand starts with **?**, substitute it with arguments from **argtab.txt**.
            - o Write substituted instructions and operands to **op.txt**.
6. Close all opened files and print a success message.
7. End the program .

INPUT

input.txt

```
PGM    START     0
ABC    MACRO     &A,&B
-      STA       &A
-      STB       &B
-      MEND       -
-      ABC       P,Q
-      ABC       R,S
P      RESW      1
Q      RESW      1
R      RESW      1
S      RESW      1
-      END        -
```

OUTPUT

namtab.txt

```
ABC
```

argtab.txt

```
R
S
```

deftab.txt

```
ABC    &A,&B
STA    ?
STB    ?
MEND
```

op.txt

```
PGM    START     0
.      ABC       P,Q
-      STA       ?
-      STB       ?
.      ABC       R,S
```

PROGRAM

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
FILE *f1,*f2,*f3,*f4,*f5;
int len,i,pos=1;
char
arg[20],mne[20],opnd[20],la[20],name[20],
mne1[20],opnd1[20],pos1[10],pos2[10];
f1=fopen("input.txt","r");
f2=fopen("namtab.txt","w+");
f3=fopen("deftab.txt","w+");
f4=fopen("argtab.txt","w+");
f5=fopen("op.txt","w+");
fscanf(f1,"%s%s%s",la,mne,opnd);
while(strcmp(mne,"END")!=0)
{
if(strcmp(mne,"MACRO")==0)
{
fprintf(f2,"%s\n",la);
fseek(f2,SEEK_SET,0);
fprintf(f3,"%s\t%s\n",la,opnd);
fscanf(f1,"%s%s%s",la,mne,opnd);
while(strcmp(mne,"MEND")!=0)
{
if(opnd[0]=='&')
{
(pos,pos1,5);
strcpy(pos2,"?");
strcpy(opnd,strcat(pos2,pos1));
pos=pos+1;
}
fprintf(f3,"%s\t%s\n",mne,opnd);
fscanf(f1,"%s%s%s",la,mne,opnd);
}
fprintf(f3,"%s",mne);
```

```
-       STA     ?
-       STB     ?
P       RESW    1
Q       RESW    1
R       RESW    1
S       RESW    1
-       END     -
```

TERMINAL

```
Successfull !!!
```

```
}
else
{
fscanf(f2,"%s",name);
if(strcmp(mne,name)==0)
{
len=strlen(opnd);
for(i=0;i<len;i++)
{
if(opnd[i]!=',')
fprintf(f4,"%c",opnd[i]);
else
fprintf(f4,"\n");
}
fseek(f3,SEEK_SET,0);
fseek(f4,SEEK_SET,0);
fscanf(f3,"%s%s",mne1,opnd1);
fprintf(f5,".\t%s\t%s\n",mne1,opnd);
fscanf(f3,"%s%s",mne1,opnd1);
while(strcmp(mne1,"MEND")!=0)
{
if((opnd[0]=='?'))
{
fscanf(f4,"%s",arg);
fprintf(f5,"-\t%s\t%s\n",mne1,arg);
}
else
fprintf(f5,"-\t%s\t%s\n",mne1,opnd1);
fscanf(f3,"%s%s",mne1,opnd1);
}
}
else
fprintf(f5,"%s\t%s\t%s\n",la,mne,opnd);
}
fscanf(f1,"%s%s%s",la,mne,opnd);
}
fprintf(f5,"%s\t%s\t%s",la,mne,opnd);
fclose(f1);
```

```
fclose(f2);
fclose(f3);
fclose(f4);
fclose(f5);
printf("Successfull !!! \n");
}
```

RESULT

C program for the implementation of one pass macroprocessor has been implemented successfully .

DATE :

EXPERIMENT NO – 2.4
IMPLEMENT AN ABSOLUTE LOADER

AIM

To implement an absolute loader .

ALGORITHM

1. **Include Necessary Libraries and Declare Variables**
    - Include the standard libraries **<stdio.h>** and **<string.h>**.
    - Declare variables:
        - **input[10]**, **label[10]**: Arrays to store input and label.
        - **addr**, **start**, **ptaddr**, **l**, **length**, **end**, **count**, **k**, **taddr**, **address**, **i**: Integer variables used for memory address manipulation and counting.
        - **ch1**, **ch2**: Characters used for file processing.
        - **fp1**, **fp2**: File pointers for input and output files.
2. **Function Declaration**
    - **check()**: A function to perform checks and manipulate file pointers and counters.
3. **Main Function (main()):**
    - Open the input file ("input.txt") in read mode (**"r"**) and the output file ("output.txt") in write mode (**"w"**).
    - Read the first string from the input file and display a header for the loader.
    - Write the header for the output file ("MEMORY ADDRESS", "CONTENTS").
    - Enter a loop that continues until the input is not "E":
        - Check if the input is "H":
            - Read label, start address, end address, and the next input string from the file.
            - Set the memory address (**address**) to the start address.
        - Otherwise, if the input is "T":
            - Store the current length and program text address (**l** and **ptaddr**).
            - Read target address, length, and input from the file.
            - Update the memory address (**address**) and check for gaps in memory.
            - If **w** is 0, set **ptaddr** to **address**.
            - Write data to the output file in a specified format, update counters, and perform checks.
        - If the input is neither "H" nor "T":

INPUT

input.txt

```
H COPY 001000 00107A
T 001000 1E 141033 482039 001036 281030 301015 482061 3C1003 00102A 0C1039 00102D
T 00101E 15 0C1036 482061 081033 4C0000 454F46 000003 000000
T 001047 1E 041030 001030 E0205D 30203F D8205D 281030 302057 549039 2C205E 38203F
T 001077 1C 101036 4C0000 000000 001000 041030 E02079 302064 509039 DC2079 2C1036
E 001000
```

OUTPUT

```
ABSOLUTE LOADER
 The contents of output file:
MEMORY ADDRESS                    CONTENTS

1000                    14103348  20390010  36281030  30101548

1010                    20613C10  0300102A  0C103900  102D0C10

1020                    36482061  0810334C  0000454F  46000003

1030                    000000xx  xxxxxxxx  xxxxxxxx  xxxxxxxx

1040                    xxxxxxxx  xxxxxx04  10300010  30E0205D

1050                    30203FD8  205D2810  30302057  5490392C

1060                    205E3820  3Fxxxxxx  xxxxxxxx  xxxxxxxx

1070                    xxxxxxxx  xxxxxx10  10364C00  00000000

1080                    00100004  1030E020  79302064  509039DC

1090                    20792C10  36
```

        - Write data to the output file in a specified format, update counters, and perform checks.
    - Read the next input from the file.
  - Close both input and output files.
  - Display the contents of the output file on the console by opening "output.txt" in read mode and printing its contents.
4. **check() Function:**
  - Increment various counters and memory addresses (**count**, **address**, **taddr**) based on certain conditions.
  - Write appropriate formatting to the output file (**fp2**).
5. End of program

PROGRAM

```
#include <stdio.h>
#include <string.h>
char input[10], label[10], ch1, ch2;
int addr, w = 0, start, ptaddr, l, length = 0, end, count = 0, k, taddr, address, i = 0;
FILE *fp1, *fp2;
void check();
void main()
{
        fp1 = fopen("input.txt", "r");
        fp2 = fopen("output.txt", "w");
        fscanf(fp1, "%s", input);
        printf("\n\nABSOLUTE LOADER\n");
        fprintf(fp2, "MEMORY ADDRESS\t\t\tCONTENTS");
        while (strcmp(input, "E") != 0)
        {
                if (strcmp(input, "H") == 0)
                {
                        fscanf(fp1, "%s %x %x %s", label, &start, &end, input);
                        address = start;
                }
                else if (strcmp(input, "T") == 0)
                {
                        l = length;
                        ptaddr = addr;
                        fscanf(fp1, "%x %x %s", &taddr, &length, input);
```

```
		addr = taddr;
		if (w == 0)
		{
			ptaddr = address;
			w = 1;
		}
		for (k = 0; k < (taddr - (ptaddr + l)); k++)
		{
			address = address + 1;
			fprintf(fp2, "xx");
			count++;
			if (count == 4)
			{
				fprintf(fp2, "  ");
				i++;
				if (i == 4)
				{
					fprintf(fp2, "\n\n%x\t\t", address);
					i = 0;
				}
				count = 0;
			}
		}
		if (taddr == start)
			fprintf(fp2, "\n\n%x\t\t", taddr);
		fprintf(fp2, "%c%c", input[0], input[1]);
		check();
		fprintf(fp2, "%c%c", input[2], input[3]);
		check();
		fprintf(fp2, "%c%c", input[4], input[5]);
		check();
		fscanf(fp1, "%s", input);
	}
	else
	{
		fprintf(fp2, "%c%c", input[0], input[1]);
		check();
		fprintf(fp2, "%c%c", input[2], input[3]);
```

```
                check();
                fprintf(fp2, "%c%c", input[4], input[5]);
                check();
                fscanf(fp1, "%s", input);
            }
        }
        fclose(fp1);
        fclose(fp2);
        printf("\n\n The contents of output file:\n");
        fp2 = fopen("output.txt", "r");
        ch2 = fgetc(fp2);
        while (ch2 != EOF)
        {
            printf("%c", ch2);
            ch2 = fgetc(fp2);
        }
        fclose(fp2);
}
void check()
{
        count++;
        address++;
        taddr = taddr + 1;
        if (count == 4)
        {
            fprintf(fp2, "  ");
            i++;
            if (i == 4)
            {
                fprintf(fp2, "\n\n%x\t\t", taddr);
                i = 0;
            }
            count = 0;
        }
}
```

RESULT

C program for the implementation of an absolute loader has been implemented successfully .

DATE :

EXPERIMENT NO – 2.5
IMPLEMENT A RELOCATING LOADER

AIM

To implement a relocating loader .

ALGORITHM

1. Start the program
2. **Include Necessary Libraries and Declare Variables**
   - Include standard libraries: **<stdio.h>**, **<string.h>**, **<stdlib.h>**.
   - Declare variables:
     - **bit[30]**: Array to store bits converted from the bitmask.
     - **bitmask[20]**: Array to store the bitmask.
     - **objptr**: File pointer for reading input from "relinput.txt".
     - **start**, **addr**: Integer variables to store the starting address and current address.
     - **rec[20]**: Array to store a record from the input file.
     - **name[20]**: Array to store the program name.
     - **modif_obj_code**: Integer variable to hold modified object code.
     - **first[3]**, **second[5]**: Arrays to parse the object code into parts.
     - **bitmask_index**, **i**, **add**, **len**: Integer variables used for indexing and looping.
3. **Function bitmask_convert(char mask[]):**
   - Converts the given bitmask into bits and stores the result in the **bit[]** array.
4. **Main Function (main()):**
   - Prompt the user to enter the starting address of the program.
   - Read the starting address from the user input.
   - Open the input file ("relinput.txt") for reading.
   - Read the first record from the file.
   - Check if the record is a header record ("H"):
     - Read the program name, starting address, and length.
     - Display a header for the output ("ADDRESS OBJECT CODE").
   - If the record is not a header record, display an error message and exit the program.
   - Read the next record from the file.
   - Enter a loop until the record is not "E":
     - If the record is "T":

INPUT

relinput.txt

```
H COPY 000000 00107A
T 000000 1E FFC 140033 481039 100036 280030 300015 481061 3C0003 20002A 1C0039
30002D
T 002500 15 E00 1D0036 481061 180033 4C1000 801000 601003
E 000000
```

OUTPUT

```
ENTER THE STARTING ADDRESS OF THE PROGRAM
2000
 ADDRESS   OBJECT CODE
______________________________
2000    142033
2003    483039
2006    102036
2009    282030
200C    302015
200F    483061
2012    3C2003
2015    20202A
2018    1C2039
201B    30202D
4500    1D2036
4503    483061
4506    182033
4509    4C1000
450C    801000
450F    601003
```

        - Read the address, length, and bitmask.
        - Convert the bitmask to bits using the **bitmask_convert()** function.
        - Read the next record.
    - If the bit at **bitmask_index** is '1':
        - Extract parts of the object code and convert to modified object code.
        - Print the modified address and object code.
    - If the bit is '0':
        - Print the address and object code without modification.
    - Increment the address and bitmask_index.
    - Read the next record.

5. **Close File and Terminate**
    - Close the input file.
    - Terminate the program.

PROGRAM

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
char bit[30];
char bitmask[20];
void bitmask_convert(char mask[])
{
  int len;
  len = strlen(mask);
  strcpy(bit, "");
  int i;
  for (i = 0; i < len; ++i){
    switch (mask[i])
    {
    case '0':
      strcat(bit, "0000");
      break;
    case '1':
      strcat(bit, "0001");
      break;
    case '2':
      strcat(bit, "0010");
```

```
    break;
  case '3':
    strcat(bit, "0011");
    break;
  case '4':
    strcat(bit, "0100");
    break;
  case '5':
    strcat(bit, "0101");
    break;
  case '6':
    strcat(bit, "0110");
    break;
  case '7':
    strcat(bit, "0111");
    break;
  case '8':
    strcat(bit, "1000");
    break;
  case '9':
    strcat(bit, "1001");
    break;
  case 'A':
    strcat(bit, "1010");
    break;
  case 'B':
    strcat(bit, "1011");
    break;
  case 'C':
    strcat(bit, "1100");
    break;
  case 'D':
    strcat(bit, "1101");
    break;
  case 'E':
    strcat(bit, "1110");
    break;
  case 'F':
```

```
            strcat(bit, "1111");
            break;
        default:
            break;
        }
    }
}
void main(){
    FILE *objptr;
    int start, addr;
    char rec[20];
    char name[20];
    int modif_obj_code;
    char first[3];
    char second[5];
    int bitmask_index = 0;
    int i;
    int add, len;
    printf("ENTER THE STARTING ADDRESS OF THE PROGRAM\n");
    scanf("%X", &start);
    addr = start;
    objptr = fopen("relinput.txt", "r");
    fscanf(objptr, "%s", rec);
    if (strcmp(rec, "H") == 0)
    {
        fscanf(objptr, "%s", name);
        fscanf(objptr, "%X", &add);
        fscanf(objptr, "%X", &len);
        printf(" ADDRESS   OBJECT CODE \n");
        printf("______________________________\n");
    }
    else
    {
        printf("INAVLID OBJECT CODE FORMAT\n");
        fclose(objptr);
        exit(1);
    }
```

```
strcpy(rec, "");
fscanf(objptr, "%s", rec);
while (strcmp(rec, "E") != 0)
{
   if (strcmp(rec, "T") == 0)
   {
      fscanf(objptr, "%X", &add);
      fscanf(objptr, "%X", &len);
      fscanf(objptr, "%s", bitmask);
      add += start;
      bitmask_index = 0;
      bitmask_convert(bitmask);
      fscanf(objptr, "%s", rec);
   }
   if (bit[bitmask_index] == '1')
   {
      for (i = 0; i < 6; ++i)
      {
         if (i < 2)
         {
            first[i] = rec[i];
         }
         else
         {
            second[i - 2] = rec[i];
         }
      }
      first[2] = '\0';
      second[4] = '\0';
      modif_obj_code = strtol(second, NULL, 16);
      modif_obj_code += start;
      printf("%X\t%s%X\n", add, first, modif_obj_code);
   }
   else
   {
      printf("%X\t%s\n", add, rec);
   }
   add += 3;
```

```
        bitmask_index++;
        fscanf(objptr, "%s", rec);
    }
    fclose(objptr);
}
```

RESULT

C program for the implementation of a relocating loader has been implemented successfully .

EXPERIMENTS USING 8086 MICROPROCESSOR KIT/EMULATOR

DATE :

EXPERIMENT NO – 3.1
STUDY OF ASSEMBLER AND DEBUGGING COMMANDS

AIM

To study about the assembler and different debugging commands .

THEORY

A program called 'assembler' is used to convert assembly input file (source file) to an object file , that is further converted to machine codes using linker . The chances of error being committed are less (than handcoding) as mneumonics are used instead of opcodes .
DEBUG.COM is a DOS utility that facilitates the debugging and trouble-shooting of assembly language programs. The DEBUG enables you to use the personal computer as a low level microprocessor kit.
The DEBUG command at DOS prompt invokes this facility . A'_' (dash) display signals the successful invoke operation of DEBUG, that is further used as DEBUG prompt for debugging commands.
A valid command is accepted using the enter key. The program DEBUG may be used either to debug a source program or to observe the results of execution of an EXE file with the help of the .LST file . DEBUG is able to troubleshoot only .EXE files .

DEBUG COMMANDS

COMMAND CHARACTER	FORMAT/FORMATS	FUNCTIONS
-R	<ENTER>	Display all Registers and flags
-R	reg<ENTER> old contents:New contents	Display specified register contents and modify with the entered new contents.
-D	<ENTER>	Display 128 memory locations of RAM starting from the current display pointer.
-D	SEG:OFFSET1 OFFSET2<ENTER>	Display memory contents in SEG from OFFSET1 to OFFSET2.
-E	<ENTER>	Enter Hex data at current display pointer SEG:OFFSET.
-E	SEG:OFFSET1 <ENTER>	Enter data at SEG:OFFSET1 byte by byte. The memory pointer is to be incremented by space key, data entry is to be completed by pressing the <ENTER> key.
-f	SEG:OFFSET1 OFFSET2 BYTE<ENTER>	Fill the memory area starting from SEG:OFFSET1 to OFFSET2 by the byte BYTE.
-f	SEG:OFFSET1 OFFSET2 BYTE1,BYTE2,BYTE3<ENTER>	Fill the memory area as above with the sequence BYTE1, BYTE2, BYTE3, etc.

-a	<ENTER>	Assemble from the current CS:IP.
-a	SEG:OFFSET <ENTER>	Assemble the entered instruction from SEG:OFFSET address.
-u	<ENTER>	Unassemble from the current CS:IP.
-u	SEG:OFFSET <ENTER>	Unassemble from the address SEG:OFFSET.
-g	<ENTER>	Execute from current CS:IP. By modifying CS and IP using R command this can be used for any address
-g	=OFFSET <ENTER>	Execute from OFFSET in the current CS.
-s	SEG:OFFSET1 TO OFFSET2 BYTE/BYTES<ENTER>	Searches a BYTE or string of BYTES separated by ' , ' in the memory block SEG:OFFSET1 to OFFSET2, and displays all the offsets at which the byte or string of bytes is found .
-q	<ENTER>	16. Quit the DEBUG and return to DOS

<u>RESULT</u>

Successfully studied about assembler and different debugging commands .

DATE :

EXPERIMENT NO – 3.2(i)
IMPLEMENT ADDITION OF 16 BIT NUMBERS (KIT)

AIM

To perform addition of 16 – bit numbers using kit .

ALGORITHM

1. Start the program
2. Set the Source Index (SI) register to point to memory address 3000.
3. Set the Destination Index (DI) register to point to memory address 4000.
4. Load the value stored at the memory address pointed to by SI into the AX register.
5. Increment the SI register by 2 bytes (assuming SI is pointing to a word or 2 bytes).
6. Load the value stored at the updated memory address pointed to by SI into the BX register.
7. Add the values stored in AX and BX registers and store the result in the AX register.
8. Check if there was no carry generated during the addition.
    - If no carry occurred, store the value 0001 at the memory address pointed to by DI.
    - If a carry occurred, store the value 0000 at the memory address pointed to by DI.
9. Increment the DI register to point to the next memory address.
10. Store the contents of the AX register at the memory address pointed to by DI.
11. Halt the execution of the program.

<u>TEST CASE – 1</u>

<u>INPUT</u>

3000 : B0
3001 : FA
3002 : B0
3003: 12

<u>OUTPUT</u>

4000 : 01
4001 : 60
4002 : 0D

<u>TEST CASE – 2</u>

<u>INPUT</u>

3000 : 10
3001 : EA
3002 : 31
3003: 04

<u>OUTPUT</u>

4000 : 00
4001 : 41
4002 : EE

PROGRAM

```
2000    MOV SI,3000
2003    MOV DI,4000
2006    MOV AX,[SI]
2008    INC SI
2009    INC SI
200A    MOV BX,[SI]
200C    ADD AX,BX
200E    JNC 2015
2010    MOV [DI],0001
2013    JMP 2018
2015    MOV [DI],0000
2018    INC DI
2019    MOV [DI],AX
201B    HLT
```

RESULT

Assembly program to perform addition of 16 – bit numbers have been implemented successfully .

DATE :

EXPERIMENT NO – 3.2(ii)
IMPLEMENT SUBTRACTION OF 16 BIT NUMBERS (KIT)

AIM

To perform subtraction of 16 – bit numbers using kit .

ALGORITHM

1. Start the program
2. Set SI register to the memory address 3000.
3. Set DI register to the memory address 4000.
4. Load the value at the memory address stored in SI into the AX register.
5. Increment the SI register by 2 (assuming SI points to a word or 2 bytes).
6. Load the value at the updated memory address stored in SI into the BX register.
7. Compare the values in AX and BX.
    - If AX is less than BX, jump to memory address 2017.
    - Otherwise, continue to the next instruction.
8. Subtract BX from AX.
9. Store the value 0000 at the memory address stored in DI.
10. Jump to memory address 2022.
11. If AX was less than BX:
    - Move the value in AX to the CX register.
    - Move the value in BX to AX.
    - Move the value in CX to BX.
    - Subtract BX from AX.
    - Store the value 0001 at the memory address stored in DI.
12. Increment the DI register by 1.
13. Store the value in AX at the memory address stored in DI.
14. Halt the program.

PROGRAM

```
2000    MOV SI,3000
2003    MOV DI,4000
2006    MOV AX,[SI]
2008    INC SI
2009    INC SI
200A    MOV BX,[SI]
200C    CMP AX,BX
200E    JC 2017
```

TEST CASE – 1

INPUT

3000 : 10
3001 : EA
3002 : 31
3003: 04

OUTPUT

4000 : 00
4001 : DF
4002 : E5

TEST CASE – 2

INPUT

3000 : 05
3001 : 01
3002 : 31
3003: 04

OUTPUT

4000 : 01
4001 : 2C
4002 : 03

```
2010    SUB AX,BX
2012    MOV [DI],0000
2015    JMP 2022
2017    MOV CX,AX
2019    MOV AX,BX
201B    MOV BX,CX
201D    SUB AX,BX
201F    MOV [DI],0001
2022    INC DI
2023    MOV [DI],AX
2025    HLT
```

RESULT

Assembly program to perform subtraction of 16 – bit numbers have been implemented successfully .

INPUT

3000 : 12

3001 : 12

3002 : 10

3003: 01

OUTPUT

4000 : 20

4001 : 33

4002 : 13

4003 : 00

DATE :

EXPERIMENT NO – 3.2(iii)
IMPLEMENT MULTIPLICATION OF 16 BIT NUMBERS (KIT)

AIM

To perform multiplication of 16 – bit numbers using kit .

ALGORITHM

1. Start the program
2. Set the source index register SI to point to memory address 3000.
3. Set the destination index register DI to point to memory address 4000.
4. Load the value at the memory address pointed by SI into the AX register.
5. Increment SI by 2 (assuming SI points to a word or 2 bytes in memory).
6. Load the value at the updated memory address pointed by SI into the BX register.
7. Multiply the value in AX by the value in BX, storing the result in AX (product) and DX (high-order bits of the product).
8. Store the value in AX at the memory address pointed by DI.
9. Increment DI by 2 (assuming DI points to a word or 2 bytes in memory).
10. Store the value in DX at the updated memory address pointed by DI.
11. Halt the program.

PROGRAM

```
2000    MOV SI,3000
2003    MOV DI,4000
2006    MOV AX,[SI]
2008    INC SI
2009    INC SI
200A    MOV BX,[SI]
200C    MUL BX
200E    MOV [DI],AX
2010    INC DI
2011    INC DI
2012    MOV [DI],DX
2014    HLT
```

RESULT

Assembly program to perform multiplicatio of 16 – bit numbers have been implemented successfully .

INPUT

3000 : 12
3001 : 04
3002 : 23
3003: 01

OUTPUT

4000 : 03
4001 : 00
4002 : A9
4003 : 00

DATE :

EXPERIMENT NO – 3.2(iv)
IMPLEMENT DIVISION OF 16 BIT NUMBERS (KIT)

AIM

To perform division of 16 – bit numbers using kit .

ALGORITHM

1. Start the program
2. Set SI (source index) register to point to memory address 3000.
3. Set DI (destination index) register to point to memory address 4000.
4. Load the value at the memory address pointed by SI into register AX.
5. Increment the SI register by 2, assuming it's pointing to a word (2 bytes) in memory.
6. Load the value at the updated memory address pointed by SI into register BX.
7. Divide the value in AX by the value in BX, storing the quotient in AX and the remainder in DX.
8. Store the value in AX at the memory address pointed by DI.
9. Increment the DI register by 2, assuming it's pointing to a word in memory.
10. Store the value in DX at the updated memory address pointed by DI.
11. Halt the program.

PROGRAM

```
2000    MOV SI,3000
2003    MOV DI,4000
2006    MOV AX,[SI]
2008    INC SI
2009    INC SI
200A    MOV BX,[SI]
200C    DIV BX
200E    MOV [DI],AX
2010    INC DI
2011    INC DI
2012    MOV [DI],DX
2014    HLT
```

RESULT

Assembly program to division of 16 – bit numbers have been implemented successfully .

DATE :

EXPERIMENT NO – 3.3
IMPLEMENT SORTING OF 16 – BIT NUMBERS (KIT)

AIM

To perform sorting of 16 – bit numbers using kit .

ALGORITHM

1. Start
2. Load the length of the array into BX.
3. Decrement BX by 1 to represent the number of iterations needed.
4. Load the length of the array into CX.
5. Decrement CX by 1 to represent the number of comparisons within an iteration.
6. Start a loop:
    - Load the value at memory address pointed by SI into AX.
    - Increment SI to point to the next element.
    - Compare AX with the value at the next memory address pointed by SI.
    - If AX is not greater than or equal to the value at the next memory address, repeat step 'c'.
    - If AX is greater than the next value:
        - Swap the values by using XCHG and store the greater value at the current address.
        - Move back SI to the previous position.
        - Store the swapped value at the current address.
7. Increment SI twice to point to the next element in the array.
8. Repeat the loop until CX becomes 0.
9. Decrement BX to track the remaining iterations.
10. If BX is not zero, repeat steps 3 to 7.
11. Halt the execution.

PROGRAM

```
0400    MOV BX,[3000]
0404    DEC BX
0405    MOV CX,[3000]
0409    DEC CX
040A    MOV SI,3002
040D    MOV AX,[SI]
040F    INC SI
```

INPUT

3000 : 06
3001 : 00
3002 : 02
3003 : 11
3004 : 35
3005 : 07
3006 : 23
3007 : 10
3008 : 00
3009 : 7A
300A : 72
300B : F0
300C : 10
300D : B0

OUTPUT

3002 : 35
3003 : 07
3004 : 23
3005 : 10
3006 : 02
3007 : 11
3008 : 00
3009 : 7A
300A : 10
300B : B0
300C : 72
300D : F0

```
0410    INC SI
0411    CMP AX,[SI]
0413    JNA 0410
0415    XCHG AX,[SI]
0417    DEC SI
0418    DEC SI
0419    MOV [SI],AX
041B    INC SI
041C    INC SI
041D    LOOP 040D
041F    DEC BX
0420    JNZ 0405
0422    HLT
```

<u>RESULT</u>

Assembly programs to perform sorting of 16 – bit numbers have been implemented successfully .

DATE :

EXPERIMENT NO – 3.4
IMPLEMENT SEARCHING OF 16 – BIT NUMBERS (EMULATOR)

AIM

To perform searching of 16 – bit numbers using 8086 emulator .

ALGORITHM

1. Define the data segment:
   - STRING1 holds an array of words.
   - MSG1 and MSG2 are strings to be printed based on the search result.
   - SE holds the value to be searched in the array.
2. Define a PRINT macro that displays a message using DOS interrupts.
3. In the code segment:
   - Set up the segment registers.
   - Move the address of the data segment to DS.
   - Set AX to the value to be searched (SE).
   - Load the address of the array STRING1 to SI.
   - Set the loop counter CX to 4 (assuming each word in the array is 2 bytes).
4. Start a loop:
   - Load a word from the memory location pointed to by SI into BX.
   - Compare AX with the value in BX.
   - If they match, jump to a label FO (Found), else continue the loop.
   - Increment SI to point to the next element.
   - Decrement the loop counter CX.
   - If CX is not zero, repeat the loop.
5. If the value was not found (CX became zero), print the message MSG2 (NOT FOUND).
6. If the value was found, print the message MSG1 (FOUND).
7. Terminate the program.

PROGRAM

```
DATA SEGMENT
STRING1 DW 1111H,2222H,3333H,4444H,5555H
MSG1 DB "Element FOUND$"
MSG2 DB "Element NOT FOUND$"
SE DW 3333H
DATA ENDS
```

OUTPUT

```
Element FOUND
```

```
PRINT MACRO MSG
MOV AH, 09H
LEA DX, MSG
INT 21H
MOV AH,4CH
INT 21H
ENDM

CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START:
MOV AX, DATA
MOV DS, AX
MOV AX, SE
LEA SI, STRING1
MOV CX, 04H

UP:
MOV BX,[SI]
CMP AX, BX
JZ FO
INC SI
INC SI
DEC CX
JNZ UP
PRINT MSG2
JMP END1

FO:
PRINT MSG1

END1: MOV AH,4CH
INT 21H
CODE ENDS
END START
```

RESULT

Assembly program to perform searching has been implemented successfully using emulator .

DATE :

EXPERIMENT NO – 3.5
IMPLEMENT STRING MANIPULATION – PALINDROME (EMULATOR)

AIM

To write an assembly program to check if a string is palindrome or not using 8086 emulator .

ALGORITHM

1. Define the DATA segment:
   - MSG1: Message prompting the user to enter a string.
   - MSG2: Message indicating that the entered string is a palindrome.
   - MSG3: Message indicating that the entered string is not a palindrome.
   - STR1: Buffer to store the input string, initialized with 50 bytes of zeros.
2. Define the CODE segment and assume segment associations (CS:code, DS:data).
3. Define a label START:
   - Move the address of the DATA segment into AX.
   - Move the value in AX to the DS register.
   - Load the effective address of MSG1 into DX.
   - Set AH to 09H (print string function).
   - Trigger interrupt 21H (INT 21H) to display MSG1 to prompt the user for input.
   - Load the effective address of STR1 into SI and DI.
   - Set AH to 01H (input character function).
4. Start a loop labeled NEXT:
   - Trigger interrupt 21H to input a character from the user.
   - Compare the input character with carriage return (0DH).
   - If it's a carriage return, jump to label TERMINATE.
   - Store the input character at the memory location pointed by DI.
   - Increment DI and continue the loop NEXT.
5. When a carriage return is encountered (label TERMINATE):
   - Store '$' (string terminator) at the memory location pointed by DI.
6. Start a loop labeled DOTHIS:
   - Decrement DI to point to the end of the entered string.
   - Load a character from the beginning of the string using SI into AL.
   - Compare the character at DI with AL.
   - If they are not equal, jump to label NOTPALINDROME.
   - Increment SI, compare SI with DI, and if SI is less than DI, continue the loop.
7. If the string is a palindrome (label PALINDROME):
   - Display MSG2 indicating that the entered string is a palindrome.

<u>OUTPUT</u>

<u>TEST CASE – 1</u>

```
ENTER THE STRING:CSA
STRING IS NOT PALINDROME
```

<u>TEST CASE – 2</u>

```
ENTER THE STRING:malayalam
STRING IS PALINDROME
```

- Jump to label XX.

8. If the string is not a palindrome (label NOTPALINDROME):
    - Display MSG3 indicating that the entered string is not a palindrome.
9. Label XX:
    - Set AH to 4CH (exit program function).
    - Trigger interrupt 21H (INT 21H) to terminate the program.

PROGRAM

```
data SEGMENT
  MSG1 DB 10,13,'ENTER THE STRING:$'
  MSG2 DB 10,13,'STRING IS PALINDROME$'
  MSG3 DB 10,13,'STRING IS NOT PALINDROME$'
  STR1 DB 50 DUP(0)
data ENDS

code SEGMENT
  ASSUME CS:code,DS:data
  START:
    MOV AX,DATA
    MOV DS,AX
    LEA DX,MSG1
    MOV AH,09H
    INT 21H
    LEA SI,STR1
    LEA DI,STR1
    MOV AH,01H

  NEXT:
    INT 21H
    CMP AL,0DH
    JE TERMINATE
    MOV [DI],AL
    INC DI
    JMP NEXT

  TERMINATE:
    MOV AL,'$'
    MOV [DI],AL
```

```
  DOTHIS:
    DEC DI
    MOV AL,[SI]
    CMP [DI],AL
    JNE NOTPALINDROME
    INC SI
    CMP SI,DI
    JL DOTHIS

  PALINDROME:
    MOV AH,09H
    LEA DX,MSG2
    INT 21H
    JMP XX

  NOTPALINDROME:
    MOV AH,09H
    LEA DX,MSG3
    INT 21H

  XX:
    MOV AH,4CH
    INT 21H
code ENDS
END START
```

RESULT

Assembly program to check whether the input string is palindrome or not has been successfully implemented using 8086 emulator .

INPUT

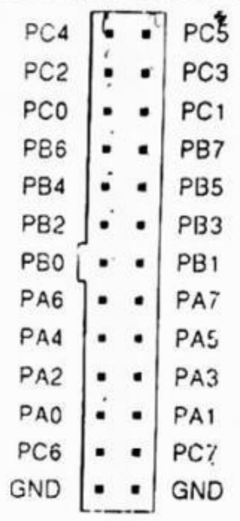
FORWARD

0500: 0A 06 05 09

REVERSE

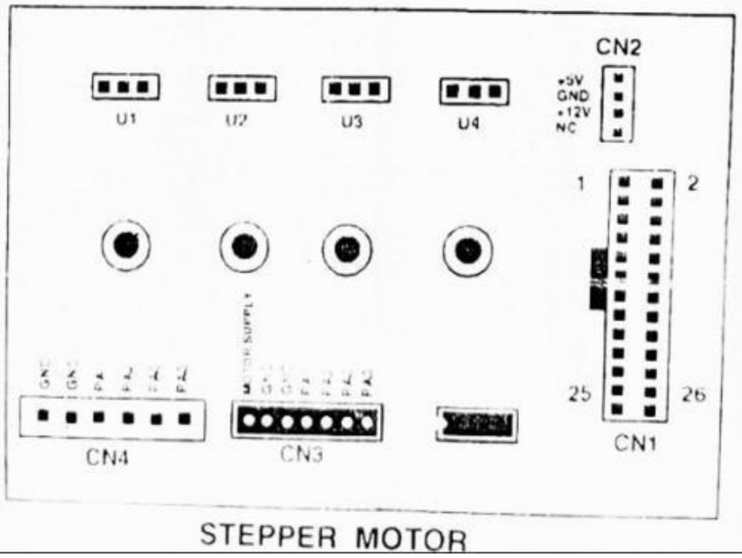
0500: 09 05 06 0A

Connector Details at CN1



SIGNAL AT CONNECTOR CN2

1	+5V	2.	GND
3	+12V	4.	NC(not connected)



STEPPER MOTOR

DATE :

EXPERIMENT NO – 3.6
STEPPER MOTOR (INTERFACING)

AIM

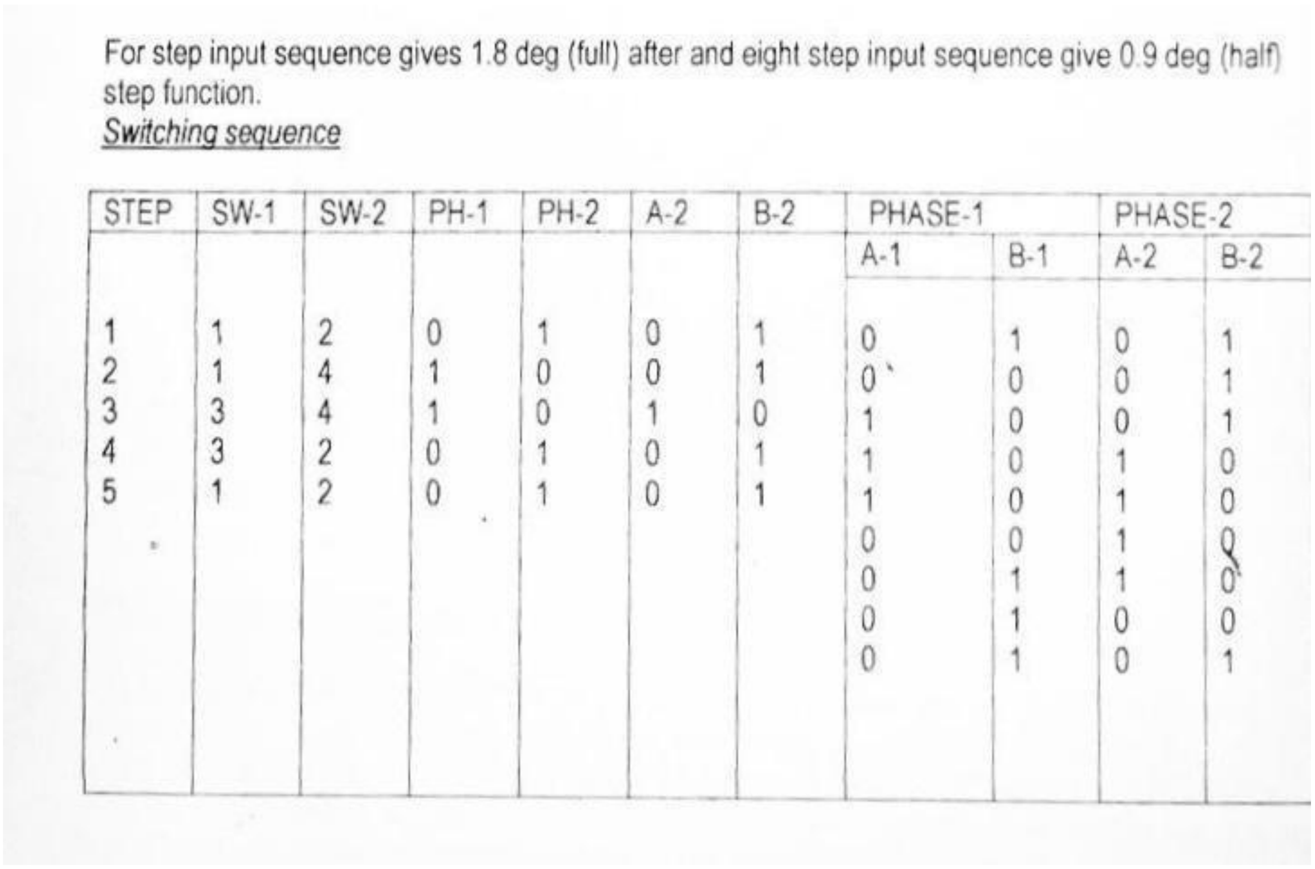
To interface stepper motor with 8086 which rotate through any given sequence .

ALGORITHM

```
1. Move the hexadecimal value 80H into the AL register.
2. Output the contents of AL to the port 46H.
3. Start a loop labeled LOOP2:
       ➢ Move the decimal value 4 into the CL register.
       ➢ Move the hexadecimal value 500H into the BX register.
       ➢ Start another loop labeled LOOP1:
             • Load the byte from the memory address pointed by BX into AL.
             • Output the contents of AL to the port 40H.
             • Call the subroutine DELAY.
             • Increment BX.
             • Continue LOOP1 until the Zero Flag (ZF) is not set.
       ➢ Jump back to LOOP2.
4. The subroutine DELAY is defined as follows:
    ➢ Push the value of CX onto the stack.
    ➢ Load the value 0FFFFH into the CX register.
    ➢ Start a loop labeled HERE:
          • Continue looping until CX becomes zero and the Zero Flag (ZF) is set.
    ➢ Pop the original value of CX from the stack.
    ➢ Return from the subroutine.
```

PROGRAM

```
0400   B0 80              MOV AL,80H
0402   E646               OUT 46H,AL
0404   B1 04    LOOP2 : MOV CL,04
0406   BB 00 05           MOV BX,500H
0409   8A 07    LOOP1 : MOV AL,[BX]
040B   E640               OUT 40H,AL
040D   E8 05 00           CALL DELAY
0410   43                 INC BX
0411   E0 F6              LOOPNZ LOOP1
```



For step input sequence gives 1.8 deg (full) after and eight step input sequence give 0.9 deg (half) step function.

Switching sequence

STEP	SW-1	SW-2	PH-1	PH-2	A-2	B-2	PHASE-1		PHASE-2	
							A-1	B-1	A-2	B-2
1	1	2	0	1	0	1	0	1	0	1
2	1	4	1	0	0	1	0	0	0	1
3	3	4	1	0	1	0	1	0	0	1
4	3	2	0	1	0	1	1	0	1	0
5	1	2	0	1	0	1	1	0	1	0
							0	0	1	0
							0	1	1	0
							0	1	0	0
							0	1	0	1

```
0413    EB EF              JMP LOOP2
0415    51        DELAY : PUSH CX
0416    B9 FF FF           MOV CX,0FFFFH
0419    E0 FE      HERE : LOOPNZ HERE
041B    59                 POP CX
041C    C3                 RET
```

RESULT

The program was executed and output was verified .

DATE :

EXPERIMENT NO – 3.7
INTERFACING WITH 8279 STATIC DISPLAY (INTERFACING)

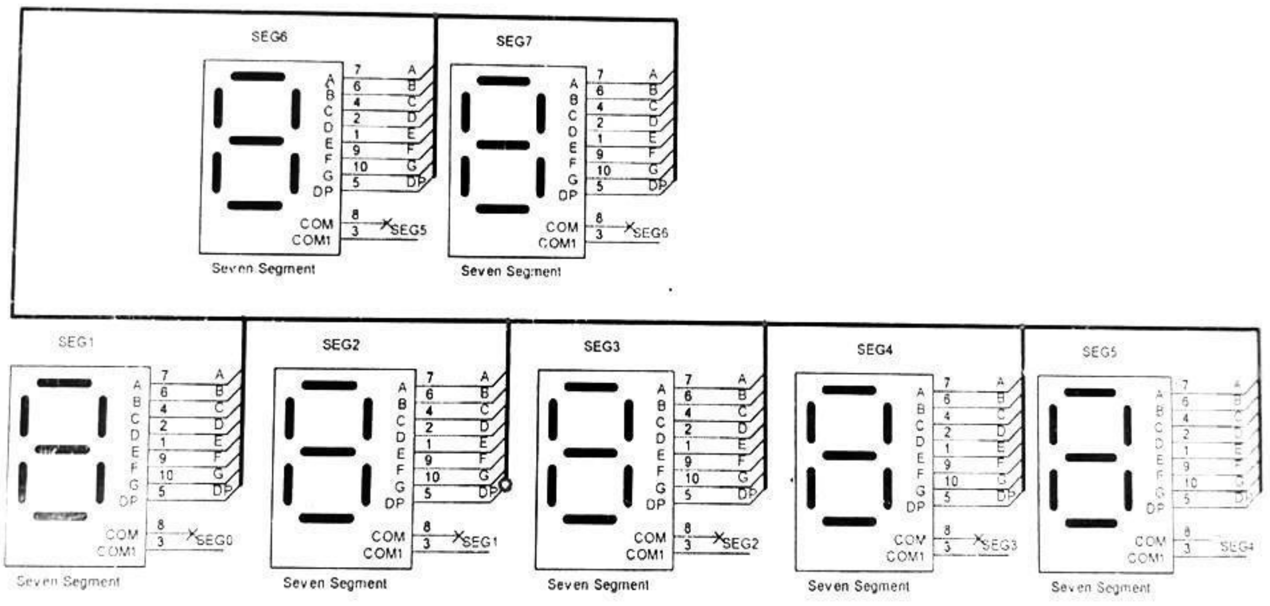
AIM

To interface 8279 keyboard and display interface with 8086 for static display .

ALGORITHM

1. Set AL register to 00H.
2. Output the content of AL (00H) to the I/O port address 22H.
3. Set AL register to 02DH.
4. Output the content of AL (02DH) to the I/O port address 22H.
5. Set AL register to 90H.
6. Output the content of AL (90H) to the I/O port address 22H.
7. Set BX register to memory address 041E.
8. Set SI register to 0.
9. Set CX register to 7.
10. Start a loop labeled REP1:
    - Load the value from the memory address specified by BX+SI into AL.
    - Output the content of AL to the I/O port address 20H.
    - Increment SI.
    - Decrease CX by 1 (LOOPNZ decrements CX and continues looping if CX is not zero).

7 SEGMENT DISPLAY



7 Segment display

INPUT

LOCATION	CONTENT
041E	77
041F	7F
0420	39
0421	3F
0422	79
0423	71
0424	7D

PROGRAM

```
0400:          MOV AL,00H
0402:          OUT 22H,AL
0404:          MOV AL,02DH
0406:          OUT 22H,AL
0408:          MOV AL,90H
040A:          OUT 22H,AL
040C:          MOV BX,041E
040F:          MOV SI,0
0412:          MOV CX,7
0415:  REP1:   MOV AL,[BX+SI]
0417:          OUT 20H,AL
0419:          INC SI
041A:          LOOPNZ REP1
```

RESULT

The program was executed and output was verified .