RAJIV GANDHI INSTITUTE OF TECHNOLOGY

GOVERNMENT ENGINEERING COLLEGE

KOTTAYAM - 686 501

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CSL 331 –SYSTEM SOFTWARE AND MICROPROCESSORS LAB

2024-2025

Submitted by

JESSIN SUNNY (KTE22CS036)

KERALA TECHNOLOGICAL UNIVERSITY THIRUVANANTHAPURAM

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

GOVERNMENT ENGINEERING COLLEGE

KOTTAYAM -686051

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LABORATORY RECORD

This is to certify that this is a bonafide record of the work done by ***JESSIN SUNNY (KTE22CS036****) of the 5th Semester in* ***CSL 331 -*** **SYSTEM SOFTWARE AND MICROPROCESSORS LAB** *during the academic year* ***2024-2025.***

Staff in Charge

Internal Examiner

External Examiner

DEPARTMENT VISION

TO BE A CENTER OF EXCELLENCE FOR NURTURING THE YOUNG MINDS TO BECOME INNOVATIVE COMPUTING PROFESSIONALS FOR THE EMPOWERMENT OF SOCIETY.

DEPARTMENT MISSION

1. TO OFFER A SOLID FOUNDATION IN COMPUTING AND TECHNOLOGY FOR CRAFTING COMPETENT PROFESSIONALS.

2. TO PROMOTE INNOVATIVE AND ENTREPRENEURIAL SKILLS OF STUDENTS BY EXPOSING THEM TO THE FOREFRONT OF DEVELOPMENTS IN THE FIELD OF COMPUTING.

3. SKILLS OF STUDENTS BY EXPOSING THEM TO THE FOREFRONT OF DEVELOPMENTS IN THE FIELD OF COMPUTING.

COURSE OUTCOMES

At the end of the course, the student should be able to

CO1	Develop 8086 programs and execute it using a microprocessor kit. (Cognitive Knowledge Level: Apply) .
CO2	Develop 8086 programs and, debug and execute it using MASM assemblers (Cognitive Knowledge Level: Apply)
CO3	Develop and execute programs to interface stepper motor, 8255, 8279 and digital to analog converters with 8086 trainer kit (Cognitive Knowledge Level: Apply)
CO4	Implement and execute different scheduling and paging algorithms in OS. (Cognitive Knowledge Level: Apply)
CO5	Design and implement assemblers, Loaders and macroprocessors. (Cognitive Knowledge Level: Apply)

INDEX

SYSTEM SOFTWARE LAB

I. Exercises/Experiments from operating system
II. Exercises/Experiments from assemblers

Exp. NO: 1 **JESSIN SUNNY**

Date: 07/08/2024 **ROLL NO: 37**

CPU SCHEDULING

Aim: Simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time.
a)FCFS b)SJF c)Round Robin(preemptive) d)Priority

Algorithm:

FCFS (First-Come, First-Served)
Step 1: Start
Step 2: Input the number of processes n and an array p[] containing arrival and burst times
Step 3: Initialize current_time <- 0, total_waiting_time <- 0, total_turnaround_time <- 0
Step 4: Sort processes by arrival time in ascending order
Step 5: For each process:
- a. If the process arrives after current_time, set current_time <- arrival_time
- b. Calculate waiting_time <- current_time - arrival_time
- c. Calculate turnaround_time <- waiting_time + burst_time
- d. Update current_time <- current_time + burst_time
- e. Accumulate total_waiting_time and total_turnaround_time

Step 6: Calculate avg_waiting_time <- total_waiting_time / n
Step 7: Calculate avg_turnaround_time <- total_turnaround_time / n
Step 8: Output the waiting time, turnaround time, and average times
Step 9: Stop

SJF (Shortest Job First)
Step 1: Start
Step 2: Input the number of processes n and an array p[] containing arrival and burst times
Step 3: Initialize current_time <- 0, completed_processes <- 0, total_waiting_time <- 0, total_turnaround_time <- 0
Step 4: While there are incomplete processes:
- a. Select the process with the shortest burst time that has arrived by current_time
- b. If no such process exists, increment current_time
- c. Otherwise:
  - i. Calculate waiting_time <- current_time - arrival_time
  - ii. Calculate turnaround_time <- waiting_time + burst_time
  - iii. Update current_time <- current_time + burst_time
  - iv. Increment completed_processes
  - v. Accumulate total_waiting_time and total_turnaround_time

Step 5: Calculate avg_waiting_time <- total_waiting_time / n
Step 6: Calculate avg_turnaround_time <- total_turnaround_time / n

Step 7: Output the waiting time, turnaround time, and average times
Step 8: Stop

Priority Scheduling
Step 1: Start
Step 2: Input the number of processes n and an array p[] containing arrival times, burst times, and priorities
Step 3: Initialize current_time <- 0, completed_processes <- 0, total_waiting_time <- 0, total_turnaround_time <- 0
Step 4: While there are incomplete processes:
a. Select the process with the highest priority (lowest priority number) that has arrived by current_time
b. If no such process exists, increment current_time
c. Otherwise:
i. Calculate waiting_time <- current_time - arrival_time
ii. Calculate turnaround_time <- waiting_time + burst_time
iii. Update current_time <- current_time + burst_time
iv. Increment completed_processes
v. Accumulate total_waiting_time and total_turnaround_time
Step 5: Calculate avg_waiting_time <- total_waiting_time / n
Step 6: Calculate avg_turnaround_time <- total_turnaround_time / n
Step 7: Output the waiting time, turnaround time, and average times
Step 8: Stop

Round Robin Scheduling
Step 1: Start
Step 2: Input the number of processes n, an array p[] containing arrival and burst times, and time slice time_slice
Step 3: Initialize current_time <- 0, total_waiting_time <- 0, total_turnaround_time <- 0, and a queue for processes
Step 4: While there are incomplete processes:
a. For each process in the queue:
i. If remaining_burst_time <= time_slice, execute process fully:
Update waiting_time <- current_time - arrival_time - (burst_time - remaining_burst_time)
Update turnaround_time <- current_time - arrival_time
Set remaining_burst_time <- 0 and completed <- true
Increment current_time by remaining_burst_time
ii. Else, execute time_slice and update remaining_burst_time <- remaining_burst_time - time_slice
Increment current_time by time_slice
iii. If process is not completed, re-add to the queue
Step 5: Calculate avg_waiting_time <- total_waiting_time / n
Step 6: Calculate avg_turnaround_time <- total_turnaround_time / n
Step 7: Output the waiting time, turnaround time, and average times
Step 8: Stop

Program

```
#include <stdio.h>
#include <stdlib.h>
struct process
{
   int pid;
   int priority;
   int arrival;
   int burst;
   int completion;
   int turnaround;
   int response;
   int waiting;
   int completed;
   int remaining;
}p[10],temp[10],temp_p;

int n,total_burst,slice;
float avg_tt,avg_rt,avg_wt;

void initialize()
{
   int i;
   avg_tt=avg_rt=avg_wt=0;
   total_burst=0;
   for(i=0;i<n;i++)
   {
      p[i].completion=0;
      p[i].turnaround=0;
      p[i].response=0;
      p[i].waiting=0;
      p[i].completed=0;
      temp[i]=p[i];
      total_burst+=p[i].burst;
      p[i].remaining=p[i].burst;
   }
}

void display(struct process* p)
{
   int i;
   printf("PID\t|PRIORITY\t|BURST TIME\t|ARRIVAL TIME\t|COMPLETION
TIME\t|TURNAROUND TIME\t|RESPONSE TIME\t|WAITING TIME\t|\n");
   for(i=0;i<n;i++)
   {
      avg_tt+=p[i].turnaround;
```

```
        avg_rt+=p[i].response;
        avg_wt+=p[i].waiting;

printf("%d\t|%d\t\t|%d\t\t|%d\t\t|%d\t\t\t|%d\t\t\t|%d\t\t|%d\t\t|\n",i+1,p[i].priority,p[
i].burst,p[i].arrival,p[i].completion,p[i].turnaround,p[i].response,p[i].waiting);
    }
    avg_tt=avg_tt/n;
    avg_wt=avg_wt/n;
    avg_rt=avg_rt/n;
    printf("Average Turnaround Time : %.2f\n",avg_tt);
   // printf("Average Response Time : %.2f\n",avg_rt);
    printf("Average Waiting Time : %.2f\n",avg_wt);
}

void fcfs()
{
    int i,j,total=0;
    initialize();
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(temp[j].arrival > temp[j+1].arrival)
            {
                temp_p=temp[j];
                temp[j]=temp[j+1];
                temp[j+1]=temp_p;
            }
        }
    }
    for(i=0;i<n;i++)
    {
        while(temp[i].arrival > total)
        {
            total++;
        }
        temp[i].completion=total+temp[i].burst;
        temp[i].turnaround=temp[i].completion-temp[i].arrival;
        temp[i].waiting=total-temp[i].arrival;
        temp[i].response=total-temp[i].arrival;
        total=total+temp[i].burst;
    }
    display(temp);
}

void sjf()
{
```

```
int i,j;
int completed=0,currenttime=0;
int minindex,minimum;
initialize();
for(i=0;i<n-1;i++)
{
   for(j=0;j<n-i-1;j++)
   {
      if(temp[j].arrival > temp[j+1].arrival)
      {
         temp_p=temp[j];
         temp[j]=temp[j+1];
         temp[j+1]=temp_p;
      }
   }
}
for(i=0;i<n-1;i++)
{
   for(j=0;j<n-i-1;j++)
   {
      if(temp[j].arrival == temp[j+1].arrival && temp[j].burst > temp[j+1].burst)
      {
         temp_p=temp[j];
         temp[j]=temp[j+1];
         temp[j+1]=temp_p;
      }
   }
}
while(completed!=n)
{
   minindex=-1;
   minimum=100000;
   for(i=0;i<n;i++)
   {
      if(temp[i].arrival <= currenttime && temp[i].completed == 0)
      {
         if(temp[i].burst < minimum)
         {
            minindex=i;
            minimum=temp[i].burst;
         }
         if(temp[i].burst == minimum)
         {
            if(temp[i].arrival < temp[minindex].arrival)
            {
               minindex=i;
               minimum=temp[i].burst;
```

```
                }
            }
        }
    }
    if(minindex == -1)
    {
        currenttime++;
    }
    else
    {
        temp[minindex].completion=currenttime+temp[minindex].burst;
        temp[minindex].turnaround=temp[minindex].completion-temp[minindex].arrival;
        temp[minindex].waiting=currenttime-temp[minindex].arrival;
        temp[minindex].response=currenttime-temp[minindex].arrival;
        temp[minindex].completed=1;
        currenttime=currenttime+temp[minindex].burst;
        completed++;
    }
  }
  display(temp);
}

void priority()
{
  int i,j;
  int completed=0,currenttime=0;
  int minindex,minimum;
  initialize();
  for(i=0;i<n-1;i++)
  {
    for(j=0;j<n-i-1;j++)
    {
      if(temp[j].arrival > temp[j+1].arrival)
      {
        temp_p=temp[j];
        temp[j]=temp[j+1];
        temp[j+1]=temp_p;
      }
    }
  }
  for(i=0;i<n-1;i++)
  {
    for(j=0;j<n-i-1;j++)
    {
      if(temp[j].arrival == temp[j+1].arrival && temp[j].priority > temp[j+1].priority)
      {
        temp_p=temp[j];
```

```
            temp[j]=temp[j+1];
            temp[j+1]=temp_p;
          }
        }
      }
      while(completed!=n)
      {
        minindex=-1;
        minimum=100000;
        for(i=0;i<n;i++)
        {
          if(temp[i].arrival <= currenttime && temp[i].completed == 0)
          {
            if(temp[i].priority < minimum)
            {
              minindex=i;
              minimum=temp[i].priority;
            }
            if(temp[i].priority == minimum)
            {
              if(temp[i].arrival < temp[minindex].arrival)
              {
                minindex=i;
                minimum=temp[i].priority;
              }
            }
          }
        }
        if(minindex == -1)
        {
          currenttime++;
        }
        else
        {
          temp[minindex].completion=currenttime+temp[minindex].burst;
          temp[minindex].turnaround=temp[minindex].completion-temp[minindex].arrival;
          temp[minindex].waiting=currenttime-temp[minindex].arrival;
          temp[minindex].response=currenttime-temp[minindex].arrival;
          temp[minindex].completed=1;
          currenttime=currenttime+temp[minindex].burst;
          completed++;
        }
      }
      display(temp);
}

void roundrobin()
```

```
{
   int i,j,flag,remain=n,sys_time;
   initialize();
   for(i=0;i<n;i++)
   {
      for(j=0;j<n-i-1;j++)
      {
         if(temp[j].arrival > temp[j+1].arrival)
         {
            temp_p=temp[j];
            temp[j]=temp[j+1];
            temp[j+1]=temp_p;
         }
      }
   }
   int ready[30],count=0;
   for(sys_time=0,i=0;remain!=0;)
   {
      if(temp[i].remaining <= slice && temp[i].remaining > 0)
      {
         sys_time+=temp[i].remaining;
         temp[i].remaining=0;
         flag=1;
      }
      else if(temp[i].remaining > 0)
      {
         sys_time+=slice;
         temp[i].remaining=temp[i].remaining-slice;
      }
      if(temp[i].remaining == 0 && flag == 1)
      {
         remain--;
         temp[i].completion=sys_time;
         temp[i].turnaround=temp[i].completion-temp[i].arrival;
         temp[i].waiting=temp[i].turnaround-temp[i].burst;
         temp[i].completed=1;
         flag=0;
      }
      int m=0;
      for(j=0;j<n;j++)
      {
         if(temp[j].arrival <= sys_time && temp[j].remaining != 0)
         {
            ready[m++]=j;
         }
      }
      if(m > 0)
```

```
        {
            count=(count+1)%m;
            i=ready[count];
        }
        else
        {
            sys_time++;
        }
    }
    display(temp);
}

void main()
{
    int i,value;
    char ch='y';
    while(ch == 'y' || ch == 'Y')
    {
        printf("1 : FCFS\n2 : SJF\n3 : Priority\n4 : Round Robin\n5 : Exit\n");
        printf("Enter your choice : ");
        scanf(" %d",&value);
        if(value == 5)
        {
            break;
        }
        printf("How many process ? : ");
        scanf("%d",&n);
        if (value == 4)
        {
            printf("Enter the time slice : ");
            scanf("%d",&slice);
        }
        for(i=0;i<n;i++)
        {
            printf("Process[%d]\n",i+1);
            if(value == 3)
            {
                printf("Priority : ");
                scanf("%d",&p[i].priority);
            }
            printf("Arrival Time : ");
            scanf("%d",&p[i].arrival);
            printf("Burst Time : ");
            scanf("%d",&p[i].burst);
        }
        if(value == 1)
        {
```

```
            printf("\nFIRST COME FIRST SERVE (FCFS)\n");
            fcfs();
        }
        else if(value == 2)
        {
            printf("\nSHORTEST JOB FIRST (SJF)\n");
            sjf();
        }
        else if(value == 3)
        {
            printf("\nPRIORITY SCHEDULING\n");
            priority();
        }
        else if(value == 4)
        {
            printf("\nROUND ROBIN SCHEDULING\n");
            roundrobin();
        }
        else
        {
            printf("Invalid Choice\n");
        }
    }
}
```

Sample Code

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

struct process
{
    int pid;
    int priority;
    int arrival;
    int burst;
    int completion;
    int turnaround;
    int response;
    int waiting;
    int completed;
    int remaining;
}p[10],temp[10],temp_p;

int n,total_burst,slice;
float avg_tt,avg_rt,avg_wt;
 void initialize()
 {
     int i;
     avg_tt=avg_rt=avg_wt=0;
     total_burst=0;
     for(i=0;i<n;i++)
     {
         p[i].completion=0;
         p[i].turnaround=0;
         p[i].response=0;
         p[i].waiting=0;
         p[i].completed=0;
         temp[i]=p[i];
         total_burst+=p[i].burst;
         p[i].remaining=p[i].burst;
     }
 }
void display(struct process* p)
{
   int i;
   printf("PID\t|PRIORITY\t|BURST TIME\t|ARRIVAL TIME\t|COMPLETION TIME\t|TURNAROUND TIME\t|RESPONSE TIME\t|WAITING TIME\t|\n");
   for(i=0;i<n;i++)
   {
       avg_tt+=p[i].turnaround;
       avg_rt+=p[i].response;
       avg_wt+=p[i].waiting;
       printf("%d\t|%d\t\t|%d\t\t|%d\t\t|%d\t\t|%d\t\t|%d\t\t|%d\t\t|\n",i+1,p[i].priority,p[i].burst,p[i].arrival,p[i].completion,p[i].turnaround,p[i].response,p[i].waiting);
   }
   avg_tt=avg_tt/n;
   avg_wt=avg_wt/n;
   avg_rt=avg_rt/n;
   printf("Average Turnaround Time : %.2f\n",avg_tt);
  // printf("Average Response Time : %.2f\n",avg_rt);
   printf("Average Waiting Time : %.2f\n",avg_wt);
}
```

```
void fcfs()
{
    int i,j,total=0;
    initialize();
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(temp[j].arrival > temp[j+1].arrival)
            {
                temp_p=temp[j];
                temp[j]=temp[j+1];
                temp[j+1]=temp_p;
            }
        }
    }
    for(i=0;i<n;i++)
    {
        while(temp[i].arrival > total)
        {
            total++;
        }
        temp[i].completion=total+temp[i].burst;
        temp[i].turnaround=temp[i].completion-temp[i].arrival;
        temp[i].waiting=total-temp[i].arrival;
        temp[i].response=total-temp[i].arrival;
        total=total+temp[i].burst;
    }
    display(temp);
}
void sjf()
{
    int i,j;
    int completed=0,currenttime=0;
    int minindex,minimum;
    initialize();
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(temp[j].arrival > temp[j+1].arrival)
            {
                temp_p=temp[j];
                temp[j]=temp[j+1];
                temp[j+1]=temp_p;
            }
        }
    }
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(temp[j].arrival == temp[j+1].arrival && temp[j].burst > temp[j+1].burst)
            {
                temp_p=temp[j];
                temp[j]=temp[j+1];
                temp[j+1]=temp_p;
            }
        }
    }
    while(completed!=n)
    {
        minindex=-1;
        minimum=100000;
        for(i=0;i<n;i++)
        {
```

```
            if(temp[i].arrival <= currenttime && temp[i].completed == 0)
            {
                if(temp[i].burst < minimum)
                {
                    minindex=i;
                    minimum=temp[i].burst;
                }
                if(temp[i].burst == minimum)
                {
                    if(temp[i].arrival < temp[minindex].arrival)
                    {
                        minindex=i;
                        minimum=temp[i].burst;
                    }
                }
            }
        }
        if(minindex == -1)
        {
            currenttime++;
        }
        else
        {
            temp[minindex].completion=currenttime+temp[minindex].burst;
            temp[minindex].turnaround=temp[minindex].completion-temp[minindex].arrival;
            temp[minindex].waiting=currenttime-temp[minindex].arrival;
            temp[minindex].response=currenttime-temp[minindex].arrival;
            temp[minindex].completed=1;
            currenttime=currenttime+temp[minindex].burst;
            completed++;
        }
    }
    display(temp);
}
void priority()
{
    int i,j;
    int completed=0,currenttime=0;
    int minindex,minimum;
    initialize();
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(temp[j].arrival > temp[j+1].arrival)
            {
                temp_p=temp[j];
                temp[j]=temp[j+1];
                temp[j+1]=temp_p;
            }
        }
    }
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(temp[j].arrival == temp[j+1].arrival && temp[j].priority > temp[j+1].priority)
            {
                temp_p=temp[j];
                temp[j]=temp[j+1];
                temp[j+1]=temp_p;
            }
        }
    }
    while(completed!=n)
    {
        minindex=-1;
        minimum=100000;
        for(i=0;i<n;i++)
        {
```

```
            if(temp[i].arrival <= currenttime && temp[i].completed == 0)
            {
                if(temp[i].priority < minimum)
                {
                    minindex=i;
                    minimum=temp[i].priority;
                }
                if(temp[i].priority == minimum)
                {
                    if(temp[i].arrival < temp[minindex].arrival)
                    {
                        minindex=i;
                        minimum=temp[i].priority;
                    }
                }
            }
        }
        if(minindex == -1)
        {
            currenttime++;
        }
        else
        {
            temp[minindex].completion=currenttime+temp[minindex].burst;
            temp[minindex].turnaround=temp[minindex].completion-temp[minindex].arrival;
            temp[minindex].waiting=currenttime-temp[minindex].arrival;
            temp[minindex].response=currenttime-temp[minindex].arrival;
            temp[minindex].completed=1;
            currenttime=currenttime+temp[minindex].burst;
            completed++;
        }
    }
    display(temp);
}
void roundrobin()
{
    int i,j,flag,remain=n,sys_time;
    initialize();
    for(i=0;i<n;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(temp[j].arrival > temp[j+1].arrival)
            {
                temp_p=temp[j];
                temp[j]=temp[j+1];
                temp[j+1]=temp_p;
            }
        }
    }
    int ready[30],count=0;
    for(sys_time=0,i=0;remain!=0;)
    {
        if(temp[i].remaining <= slice && temp[i].remaining > 0)
        {
            sys_time+=temp[i].remaining;
            temp[i].remaining=0;
            flag=1;
        }
        else if(temp[i].remaining > 0)
        {
            sys_time+=slice;
            temp[i].remaining=temp[i].remaining-slice;
        }
        if(temp[i].remaining == 0 && flag == 1)
        {
            remain--;
            temp[i].completion=sys_time;
            temp[i].turnaround=temp[i].completion-temp[i].arrival;
            temp[i].waiting=temp[i].turnaround-temp[i].burst;
            temp[i].completed=1;
```

```
            flag=0;
        }
        int m=0;
        for(j=0;j<n;j++)
        {
            if(temp[j].arrival <= sys_time && temp[j].remaining != 0)
            {
                ready[m++]=j;
            }
        }
        if(m > 0)
        {
            count=(count+1)%m;
            i=ready[count];
        }
        else
        {
            sys_time++;
        }
    }
    display(temp);
}
void main()
{
    int i,value;
    char ch='y';
    while(ch == 'y' || ch == 'Y')
    {
        printf("1 : FCFS\n2 : SJF\n3 : Priority\n4 : Round Robin\n5 : Exit\n");
        printf("Enter your choice : ");
        scanf(" %d",&value);
        if(value == 5)
        {
            break;
        }
        printf("How many process ? : ");
        scanf("%d",&n);
        if (value == 4)
        {
            printf("Enter the time slice : ");
            scanf("%d",&slice);
        }
        for(i=0;i<n;i++)
        {
            printf("Process[%d]\n",i+1);
            if(value == 3)
            {
                printf("Priority : ");
                scanf("%d",&p[i].priority);
            }
            printf("Arrival Time : ");
            scanf("%d",&p[i].arrival);
            printf("Burst Time : ");
            scanf("%d",&p[i].burst);
        }
        if(value == 1)
        {
            printf("\nFIRST COME FIRST SERVE (FCFS)\n");
            fcfs();
        }
        else if(value == 2)
        {
            printf("\nSHORTEST JOB FIRST (SJF)\n");
            sjf();
        }
        else if(value == 3)
        {
            printf("\nPRIORITY SCHEDULING\n");
            priority();
        }
```

```
        else if(value == 4)
        {
            printf("\nROUND ROBIN SCHEDULING\n");
            roundrobin();
        }
        else
        {
            printf("Invalid Choice\n");
        }
    }
}
```

Output

```
1 : FCFS
2 : SJF
3 : Priority
4 : Round Robin
5 : Exit
Enter your choice : 1
How many process ? : 5
Process[1]
Arrival Time : 0
Burst Time : 8
Process[2]
Arrival Time : 2
Burst Time : 6
Process[3]
Arrival Time : 2
Burst Time : 1
Process[4]
Arrival Time : 1
Burst Time : 9
Process[5]
Arrival Time : 3
Burst Time : 3

FIRST COME FIRST SERVE (FCFS)
PID     |PRIORITY       |BURST TIME     |ARRIVAL TIME   |COMPLETION TIME        |TURNAROUND TIME        |RESPONSE TIME  |WAITING TIME   |
1       |0              |8              |0              |8                      |8                      |0              |0              |
2       |0              |9              |1              |17                     |16                     |7              |7              |
3       |0              |6              |2              |23                     |21                     |15             |15             |
4       |0              |1              |2              |24                     |22                     |21             |21             |
5       |0              |3              |3              |27                     |24                     |21             |21             |
Average Turnaround Time : 18.20
Average Waiting Time : 12.80
1 : FCFS
2 : SJF
3 : Priority
4 : Round Robin
5 : Exit
Enter your choice : 2
How many process ? : 4
Process[1]
Arrival Time : 0
Burst Time : 8
Process[2]
Arrival Time : 2
Burst Time : 4
Process[3]
Arrival Time : 4
Burst Time : 9
Process[4]
Arrival Time : 5
Burst Time : 5

SHORTEST JOB FIRST (SJF)
PID     |PRIORITY       |BURST TIME     |ARRIVAL TIME   |COMPLETION TIME        |TURNAROUND TIME        |RESPONSE TIME  |WAITING TIME   |
1       |0              |8              |0              |8                      |8                      |0              |0              |
2       |0              |4              |2              |12                     |10                     |6              |6              |
3       |0              |9              |4              |26                     |22                     |13             |13             |
4       |0              |5              |5              |17                     |12                     |7              |7              |
Average Turnaround Time : 13.00
Average Waiting Time : 6.50
```

```
1 : FCFS
2 : SJF
3 : Priority
4 : Round Robin
5 : Exit
Enter your choice : 3
How many process ? : 5
Process[1]
Priority : 4
Arrival Time : 0
Burst Time : 8
Process[2]
Priority : 1
Arrival Time : 2
Burst Time : 6
Process[3]
Priority : 2
Arrival Time : 2
Burst Time : 1
Process[4]
Priority : 2
Arrival Time : 1
Burst Time : 9
Process[5]
Priority : 3
Arrival Time : 3
Burst Time : 3

PRIORITY SCHEDULING
PID     |PRIORITY       |BURST TIME     |ARRIVAL TIME   |COMPLETION TIME        |TURNAROUND TIME        |RESPONSE TIME  |WAITING TIME   |
1       |4              |8              |0              |8                      |8                      |0              |0              |
2       |2              |9              |1              |23                     |22                     |13             |13             |
3       |1              |6              |2              |14                     |12                     |6              |6              |
4       |2              |1              |2              |24                     |22                     |21             |21             |
5       |3              |3              |3              |27                     |24                     |21             |21             |
Average Turnaround Time : 17.60
Average Waiting Time : 12.20
1 : FCFS
2 : SJF
3 : Priority
4 : Round Robin
5 : Exit
Enter your choice : 4
How many process ? : 5
Enter the time slice : 4
Process[1]
Arrival Time : 0
Burst Time : 11
Process[2]
Arrival Time : 5
Burst Time : 13
Process[3]
Arrival Time : 9
Burst Time : 6
Process[4]
Arrival Time : 13
Burst Time : 9
Process[5]
Arrival Time : 17
Burst Time : 12

ROUND ROBIN SCHEDULING
PID     |PRIORITY       |BURST TIME     |ARRIVAL TIME   |COMPLETION TIME        |TURNAROUND TIME        |RESPONSE TIME  |WAITING TIME   |
1       |4              |11             |0              |27                     |27                     |0              |16             |
2       |1              |13             |5              |51                     |46                     |0              |33             |
3       |2              |6              |9              |29                     |20                     |0              |14             |
4       |2              |9              |13             |46                     |33                     |0              |24             |
5       |3              |12             |17             |45                     |28                     |0              |16             |
Average Turnaround Time : 30.80
Average Waiting Time : 20.60
1 : FCFS
2 : SJF
3 : Priority
4 : Round Robin
5 : Exit
Enter your choice : 5
```

Result

The algorithm was developed and the program was coded. The program was tested successfully.

Exp. NO: 2 **JESSIN SUNNY**

Date: 07/08/2024 **ROLL NO: 37**

BANKER'S ALGORITHM

Aim: Implement the banker's algorithm for deadlock avoidance.

Algorithm:

Step 1: Start

Step 2: Input the number of processes and resource types as n and m respectively

Step 3: Input the available matrix of order m, allocation matrix of order n and m, max matrix of order n and m

Step 4: Calculate the need matrix of order n and m by
need[i][j] <- max[i][j] – allocation[i][j], where i <- 0 to n-1 and j<- 0 to m-1

Step 5: Let work[i] <- available[i] where i <- 0 to m-1

Step 6: Initialize finish[i] <- 0 where i<- 0 to n-1

Step 7: Let y <-0

Step 8: Find a process i such that finish[i] <- 0 and need[i][j] < available[j],
where i <- 0 to n-1 and j<- 0 to m-1
Let work[j] <- work[j] + allocation[i][j], finish[i] <- 1, seq[y] <- I and y <- y+1

Step 9: Check if all processes has been allocated and finished its execution, If so then let safeflag <- 1

Step 10: If safeflag <- 1 print seq[i] where i <- 0 to n-1

Step 11: Else print "System is in unsafe state"

Step 11: Stop

Program

```
#include <stdio.h>
#include <stdlib.h>
#define size 25

void main()
{
   int i,j,n,m,avail[size],max[size][size],alloc[size][size];
   int s=0,flag,safeflag=1,k,y,work[size],need[size][size],finish[size],sequence[size];
   printf("How many Processes ? : ");
   scanf("%d",&n);
   printf("How many Resourc Types ? : ");
```

```
scanf("%d",&m);
printf("Enter the Available Matrix\n");
for(i=0;i<m;i++)
{
   scanf("%d",&avail[i]);
}
printf("Enter the Max Matrix\n");
for(i=0;i<n;i++)
{
   printf("P[%d] : ",i+1);
   for(j=0;j<m;j++)
   {
      scanf("%d",&max[i][j]);
   }
}
printf("Enter the Allocation Matrix\n");
for(i=0;i<n;i++)
{
   printf("P[%d] : ",i+1);
   for(j=0;j<m;j++)
   {
      scanf("%d",&alloc[i][j]);
   }
}
for(i=0;i<n;i++)
{
   for(j=0;j<m;j++)
   {
      need[i][j]=max[i][j]-alloc[i][j];
   }
}
for(i=0;i<m;i++)
{
   work[i]=avail[i];
}
for(i=0;i<n;i++)
{
   finish[i]=0;
}
for(k=0;k<n;k++)
{
   for(i=0;i<n;i++)
   {
      if(finish[i] == 0)
      {
         flag=1;
         for(j=0;j<m;j++)
```

```
            {
               if(need[i][j] > work[j])
               {
                  flag=0;
                  break;
               }
            }
            if(flag)
            {
               sequence[s++]=i+1;
               for(y=0;y<m;y++)
               {
                  work[y]+=alloc[i][y];
               }
               finish[i]=1;
            }
         }
      }
   }
   printf("PROCESS\t\tALLOCATION\tMAX\t\tNEED\t\n");
   for(i=0;i<n;i++)
   {
      printf("P[%d]\t\t",i+1);
      for(j=0;j<m;j++)
      {
         printf("%d ",alloc[i][j]);
      }
      printf("\t\t");
      for(j=0;j<m;j++)
      {
         printf("%d ",max[i][j]);
      }
      printf("\t\t");
      for(j=0;j<m;j++)
      {
         printf("%d ",need[i][j]);
      }
      printf("\t\t\n");
   }
   for(i=0;i<n;i++)
   {
      if(finish[i] == 0)
      {
         safeflag=0;
         break;
      }
   }
```

```
    if(safeflag)
    {
        printf("SAFE SEQUENCE : ");
        for(i=0;i<n;i++)
        {
            printf("P%d ",sequence[i]);
        }
        printf("\n");
    }
    else
    {
        printf("SYSTEM IS IN UNSAFE STATE\n");
    }
}
```

Sample Code

```c
#include <stdio.h>
#include <stdlib.h>
#define size 25

void main()
{
    int i,j,n,m,avail[size],max[size][size],alloc[size][size];
    int s=0,flag,safeflag=1,k,y,work[size],need[size][size],finish[size],sequence[size];
    printf("How many Processes ? : ");
    scanf("%d",&n);
    printf("How many Resourc Types ? : ");
    scanf("%d",&m);
    printf("Enter the Available Matrix\n");
    for(i=0;i<m;i++)
    {
        scanf("%d",&avail[i]);
    }
    printf("Enter the Max Matrix\n");
    for(i=0;i<n;i++)
    {
        printf("P[%d] : ",i+1);
        for(j=0;j<m;j++)
        {
            scanf("%d",&max[i][j]);
        }
    }
    printf("Enter the Allocation Matrix\n");
    for(i=0;i<n;i++)
    {
        printf("P[%d] : ",i+1);
        for(j=0;j<m;j++)
        {
            scanf("%d",&alloc[i][j]);
        }
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
            need[i][j]=max[i][j]-alloc[i][j];
        }
```

```
        }
    }
    for(i=0;i<m;i++)
    {
        work[i]=avail[i];
    }
    for(i=0;i<n;i++)
    {
        finish[i]=0;
    }
    for(k=0;k<n;k++)
    {
        for(i=0;i<n;i++)
        {
            if(finish[i] == 0)
            {
                flag=1;
                for(j=0;j<m;j++)
                {
                    if(need[i][j] > work[j])
                    {
                        flag=0;
                        break;
                    }
                }
                if(flag)
                {
                    sequence[s++]=i+1;
                    for(y=0;y<m;y++)
                    {
                        work[y]+=alloc[i][y];
                    }
                    finish[i]=1;
                }
            }
        }
    }
    printf("PROCESS\t\tALLOCATION\tMAX\t\tNEED\t\n");
    for(i=0;i<n;i++)
    {
        printf("P[%d]\t\t",i+1);
         for(j=0;j<m;j++)
         {
             printf("%d ",need[i][j]);
         }
         printf("\t\t\n");
     }
     for(i=0;i<n;i++)
     {
         if(finish[i] == 0)
         {
             safeflag=0;
             break;
         }
     }
     if(safeflag)
     {
         printf("SAFE SEQUENCE : ");
         for(i=0;i<n;i++)
         {
             printf("P%d ",sequence[i]);
         }
         printf("\n");
     }
     else
     {
         printf("SYSTEM IS IN UNSAFE STATE\n");
     }
}
```

Output

```
How many Processes ? : 5
How many Resourc Types ? : 3
Enter the Available Matrix
3 3 2
Enter the Max Matrix
P[1] : 7 5 3
P[2] : 3 2 2
P[3] : 9 0 2
P[4] : 2 2 2
P[5] : 4 3 3
Enter the Allocation Matrix
P[1] : 0 1 0
P[2] : 2 0 0
P[3] : 3 0 2
P[4] : 2 1 1
P[5] : 0 0 2
PROCESS         ALLOCATION      MAX             NEED
P[1]            0 1 0           7 5 3           7 4 3
P[2]            2 0 0           3 2 2           1 2 2
P[3]            3 0 2           9 0 2           6 0 0
P[4]            2 1 1           2 2 2           0 1 1
P[5]            0 0 2           4 3 3           4 3 1
SAFE SEQUENCE : P2 P4 P5 P1 P3
```

Result

The algorithm was developed and the program was coded. The program was tested successfully.

Exp. NO: 3 **JESSIN SUNNY**

Date: 13/08/2024 **ROLL NO: 37**

DISK SCHEDULING

Aim: Simulate the following disk scheduling algorithms.
a)FCFS b)SCAN c)C-SCAN

Algorithm:

Input: Number of requests as n, requests as an array r of size n, disk size as size, specify whether increasing order or decreasing order traversal and initial head position as initial

Output: Seek time and Disk traversal order

FCFS

Step 1: Start

Step 2: Let seek <- 0, current <- initial

Step 3: Print “Disk Traversal Order”

Step 4: Let seek <- |r[i] - current| + seek, current <- r[i]
Print r[i], where i <- 0 to n-1

Step 5: Print “Seek Time : ”,seek

Step 6: Stop

SCAN

Step 1: Start

Step 2: Let seek <- 0, current <- initial

Step 3: Print “Disk Traversal Order”

Step 4: If Increasing order traversal then sort the array r in ascending order

Step 4.1: Find the index to start the traversal from the sorted array r

Step 4.2: Let seek <- |r[i] - current| + seek, current <- r[i]
Print r[i], where i <- index to n-1

Step 4.3: Let seek <- |size - 1 - current| + seek, current <- size - 1

Step 4.4: Let seek <- |r[i] - current| + seek, current <- r[i]
Print r[i], where i <- index-1 to 0

Step 5: Else if decreasing order traversal then sort the array r in descending order

Step 5.1: Find the index to start the traversal from the sorted array r

Step 5.2: Let seek <- |r[i] - current| + seek, current <- r[i]
Print r[i], where i <- index to n-1

Step 5.3: Let seek <- |current - 0| + seek, current <- 0

Step 5.4: Let seek <- |r[i] - current| + seek, current <- r[i]
Print r[i], where i <- index-1 to 0

Step 6: Print “Seek Time : ”,seek

Step 7: Stop

CSCAN

Step 1: Start

Step 2: Let seek <- 0, current <- initial

Step 3: Print “Disk Traversal Order”

Step 4: If Increasing order traversal then sort the array r in ascending order

Step 4.1: Find the index to start the traversal from the sorted array r

Step 4.2: Let seek <- |r[i] - current| + seek, current <- r[i]
Print r[i], where i <- index to n-1

Step 4.3: Let seek <- |size - 1 - current| + seek, current <- size – 1

Step 4.4: Let seek <- |current - 0| + seek, current <- 0

Step 4.5: Let seek <- |r[i] - current| + seek, current <- r[i]
Print r[i], where i <- 0 to index-1

Step 5: Else if decreasing order traversal then sort the array r in descending order

Step 5.1: Find the index to start the traversal from the sorted array r

Step 5.2: Let seek <- |r[i] - current| + seek, current <- r[i]
Print r[i], where i <- index to n-1

Step 5.3: Let seek <- |current - 0| + seek, current <- 0

Step 5.4: Let seek <- |size -1 - current| + seek, current <- size-1

Step 5.5: Let seek <- |r[i] - current| + seek, current <- r[i]
Print r[i], where i <- 0 to index-1

Step 6: Print “Seek Time : ”,seek

Step 7: Stop

Program

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

int n,r[25],size,initial,traversal;

void fcfs()
{
   int i,seek=0,current=initial;
   printf("Disk Traversal Order : ");
   for(i=0;i<n;i++)
   {
      seek+=abs(current-r[i]);
      current=r[i];
      printf("%d ",r[i]);
   }
   printf("\nSeek Time : %d\n",seek);
}

void scan()
{
   int i,j,temp,current=initial,seek=0,index;
   printf("Disk Traversal Order : ");
   if(traversal == 1)
   {
      for(i=0;i<n;i++)
      {
         for(j=0;j<n-i-1;j++)
         {
            if(r[j] > r[j+1])
            {
               temp=r[j];
               r[j]=r[j+1];
               r[j+1]=temp;
            }
         }
      }
      for(i=0;i<n;i++)
      {
         if(current < r[i])
         {
            index=i;
            break;
         }
      }
      for(i=index;i<n;i++)
      {
```

```
            seek+=abs(r[i]-current);
            current=r[i];
            printf("%d ",r[i]);
        }
        seek+=abs(size-1-current);
        current=size-1;
        for(i=index-1;i>=0;i--)
        {
            seek+=abs(current-r[i]);
            current=r[i];
            printf("%d ",r[i]);
        }
        printf("\nSeek Time : %d\n",seek);
    }
    else if(traversal == 2)
    {
        for(i=0;i<n;i++)
        {
            for(j=0;j<n-i-1;j++)
            {
                if(r[j] < r[j+1])
                {
                    temp=r[j];
                    r[j]=r[j+1];
                    r[j+1]=temp;
                }
            }
        }
        for(i=0;i<n;i++)
        {
            if(current > r[i])
            {
                index=i;
                break;
            }
        }
        for(i=index;i<n;i++)
        {
            seek+=abs(current-r[i]);
            current=r[i];
            printf("%d ",r[i]);
        }
        seek+=abs(current-0);
        current=0;
        for(i=index-1;i>=0;i--)
        {
            seek+=abs(r[i]-current);
```

```
            current=r[i];
            printf("%d ",r[i]);
        }
        printf("\nSeek Time : %d\n",seek);
    }
}

void cscan()
{
    int i,j,temp,current=initial,seek=0,index;
    printf("Disk Traversal Order : ");
    if(traversal == 1)
    {
        for(i=0;i<n;i++)
        {
            for(j=0;j<n-i-1;j++)
            {
                if(r[j] > r[j+1])
                {
                    temp=r[j];
                    r[j]=r[j+1];
                    r[j+1]=temp;
                }
            }
        }
        for(i=0;i<n;i++)
        {
            if(current < r[i])
            {
                index=i;
                break;
            }
        }
        for(i=index;i<n;i++)
        {
            seek+=abs(r[i]-current);
            current=r[i];
            printf("%d ",r[i]);
        }
        seek+=abs(size-1-current);
        current=size-1;
        seek+=abs(current-0);
        current=0;
        for(i=0;i<index;i++)
        {
            seek+=abs(current-r[i]);
            current=r[i];
```

```
            printf("%d ",r[i]);
        }
        printf("\nSeek Time : %d\n",seek);
    }
    else if(traversal == 2)
    {
        for(i=0;i<n;i++)
        {
            for(j=0;j<n-i-1;j++)
            {
                if(r[j] < r[j+1])
                {
                    temp=r[j];
                    r[j]=r[j+1];
                    r[j+1]=temp;
                }
            }
        }
        for(i=0;i<n;i++)
        {
            if(current > r[i])
            {
                index=i;
                break;
            }
        }
        for(i=index;i<n;i++)
        {
            seek+=abs(current-r[i]);
            current=r[i];
            printf("%d ",r[i]);
        }
        seek+=abs(current-0);
        current=0;
        seek+=abs(size-1-current);
        current=size-1;
        for(i=0;i<index;i++)
        {
            seek+=abs(r[i]-current);
            current=r[i];
            printf("%d ",r[i]);
        }
        printf("\nSeek Time : %d\n",seek);
    }
}

void main()
```

```
{
   int i,value;
   char ch='y';
   printf("Enter the size of disk : ");
   scanf("%d",&size);
   printf("How many requests ? : ");
   scanf("%d",&n);
   printf("Enter the request sequence : ");
   for(i=0;i<n;i++)
   {
      scanf("%d",&r[i]);
   }
   printf("Enter the initial position of header : ");
   scanf("%d",&initial);
   while(ch=='y' || ch=='Y')
   {
      printf("1 : FCFS\n2 : SCAN\n3 : C-SCAN\n4 : EXIT\n");
      printf("Enter your value : ");
      scanf(" %d",&value);
      if(value==4)
      {
         break;
      }
      if(value==2 || value==3)
      {
         printf("1 : Increasing order traversal\n2 : Decreasing order traversal\n");
         printf("Enter your choice  : ");
         scanf(" %d",&traversal);
      }
      switch(value)
      {
         case 1 : fcfs();
                  break;
         case 2 : scan();
                  break;
         case 3 : cscan();
                  break;
      }
   }
}
```

Sample Code

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

int n,r[25],size,initial,traversal;

void fcfs()
{
    int i,seek=0,current=initial;
    printf("Disk Traversal Order : ");
    for(i=0;i<n;i++)
    {
        seek+=abs(current-r[i]);
        current=r[i];
        printf("%d ",r[i]);
    }
    printf("\nSeek Time : %d\n",seek);
}
void scan()
{
    int i,j,temp,current=initial,seek=0,index;
    printf("Disk Traversal Order : ");
    if(traversal == 1)
    {
        for(i=0;i<n;i++)
        {
            for(j=0;j<n-i-1;j++)
            {
                if(r[j] > r[j+1])
                {
                    temp=r[j];
                    r[j]=r[j+1];
                    r[j+1]=temp;
                }
            }
        }
        for(i=0;i<n;i++)
        {
            if(current < r[i])
            {
                index=i;
                break;
            }
        }
        for(i=index;i<n;i++)
        {
            seek+=abs(r[i]-current);
            current=r[i];
            printf("%d ",r[i]);
        }
        seek+=abs(size-1-current);
        current=size-1;
        for(i=index-1;i>=0;i--)
        {
            seek+=abs(current-r[i]);
            current=r[i];
            printf("%d ",r[i]);
        }
        printf("\nSeek Time : %d\n",seek);
    }
    else if(traversal == 2)
    {
        for(i=0;i<n;i++)
        {
            for(j=0;j<n-i-1;j++)
            {
                if(r[j] < r[j+1])
                {
                    temp=r[j];
                    r[j]=r[j+1];
                    r[j+1]=temp;
                }
            }
        }
```

```
void cscan()
{
    int i,j,temp,current=initial,seek=0,index;
    printf("Disk Traversal Order : ");
    if(traversal == 1)
    {
        for(i=0;i<n;i++)
        {
            for(j=0;j<n-i-1;j++)
            {
                if(r[j] > r[j+1])
                {
                    temp=r[j];
                    r[j]=r[j+1];
                    r[j+1]=temp;
                }
            }
        }
        for(i=0;i<n;i++)
        {
            if(current < r[i])
            {
                index=i;
                break;
            }
        }
         for(i=index;i<n;i++)
         {
             seek+=abs(r[i]-current);
             current=r[i];
             printf("%d ",r[i]);
         }
         seek+=abs(size-1-current);
         current=size-1;
         seek+=abs(current-0);
         current=0;
         for(i=0;i<index;i++)
         {
             seek+=abs(current-r[i]);
             current=r[i];
             printf("%d ",r[i]);
         }
         printf("\nSeek Time : %d\n",seek);
     }
```

```
    else if(traversal == 2)
    {
        for(i=0;i<n;i++)
        {
            for(j=0;j<n-i-1;j++)
            {
                if(r[j] < r[j+1])
                {
                    temp=r[j];
                    r[j]=r[j+1];
                    r[j+1]=temp;
                }
            }
        }
        for(i=0;i<n;i++)
        {
            if(current > r[i])
            {
                index=i;
                break;
            }
        }
        for(i=index;i<n;i++)
        {
            seek+=abs(current-r[i]);
            current=r[i];
            printf("%d ",r[i]);
        }
        seek+=abs(current-0);
        current=0;
        seek+=abs(size-1-current);
        current=size-1;
        for(i=0;i<index;i++)
        {
            seek+=abs(r[i]-current);
            current=r[i];
            printf("%d ",r[i]);
        }
         printf("\nSeek Time : %d\n",seek);
    }
}

void main()
{
    int i,value;
    char ch='y';
    printf("Enter the size of disk : ");
    scanf("%d",&size);
    printf("How many requests ? : ");
    scanf("%d",&n);
    printf("Enter the request sequence : ");
    for(i=0;i<n;i++)
    {
        scanf("%d",&r[i]);
    }
    printf("Enter the initial position of header : ");
    scanf("%d",&initial);
    while(ch=='y' || ch=='Y')
    {
        printf("1 : FCFS\n2 : SCAN\n3 : C-SCAN\n4 : EXIT\n");
        printf("Enter your value : ");
        scanf(" %d",&value);
        if(value==4)
        {
            break;
        }
        if(value==2 || value==3)
        {
            printf("1 : Increasing order traversal\n2 : Decreasing order traversal\n");
            printf("Enter your choice  : ");
            scanf(" %d",&traversal);
        }
        switch(value)
        {
            case 1 : fcfs();
                     break;
            case 2 : scan();
                     break;
            case 3 : cscan();
                     break;
        }
    }
}
```

Output

```
Enter the size of disk : 200
How many requests ? : 8
Enter the request sequence : 98 183 37 122 14 124 65 67
Enter the initial position of header : 53
1 : FCFS
2 : SCAN
3 : C-SCAN
4 : EXIT
Enter your value : 1
Disk Traversal Order : 98 183 37 122 14 124 65 67
Seek Time : 640
1 : FCFS
2 : SCAN
3 : C-SCAN
4 : EXIT
Enter your value : 3
1 : Increasing order traversal
2 : Decreasing order traversal
Enter your choice  : 1
Disk Traversal Order : 65 67 98 122 124 183 14 37
Seek Time : 382
1 : FCFS
2 : SCAN
3 : C-SCAN
4 : EXIT
Enter your value : 3
1 : Increasing order traversal
2 : Decreasing order traversal
Enter your choice  : 2
Disk Traversal Order : 37 14 183 124 122 98 67 65
Seek Time : 386
1 : FCFS
2 : SCAN
3 : C-SCAN
4 : EXIT
Enter your value : 4
1 : FCFS
2 : SCAN
3 : C-SCAN
4 : EXIT
Enter your value : 2
1 : Increasing order traversal
2 : Decreasing order traversal
Enter your choice  : 1
Disk Traversal Order : 65 67 98 122 124 183 37 14
Seek Time : 331
1 : FCFS
2 : SCAN
3 : C-SCAN
4 : EXIT
Enter your value : 2
1 : Increasing order traversal
2 : Decreasing order traversal
Enter your choice  : 2
Disk Traversal Order : 37 14 65 67 98 122 124 183
Seek Time : 236
```

Result

The algorithm was developed and the program was coded. The program was tested successfully.

Exp. NO: 4 **JESSIN SUNNY**

Date: 03/09/2024 **ROLL NO: 37**

PASS ONE OF A TWO PASS ASSEMBLER

Aim: Implement pass one of a two pass assembler

Algorithm:

```
begin
   read first input line
   if OPCODE = 'START' then
      begin
         save # [ OPERAND ] as starting address
         initialize LOCCTR to starting address
         write line to intermediate file
         read next input line
      end { if START }
   else
      initialize LOCCTR to 0

   write last line to intermediate file
   save ( LOCCTR – starting address ) as program length
end { PASS 1 }
while OPCODE != 'END' do
   begin
      if this is not a comment line then
         begin
            if there is a symbol in the LABEL field then
               begin
                  search SYMTAB for LABEL
                  if found then
                     set error flag (duplicate symbol)
                  else
                     insert {LABEL,LOCCTR} into SYMTAB
               end { if symbol }
               search OPTAB for OPCODE
               if found then
                   add 3 { instruction length } to LOCCTR
               else if OPCODE = 'WORD' then
                   add 3 to LOCCTR
               else if OPCODE = 'RESW' then
                   add 3 * # [ OPERAND ] to LOCCTR
               else if OPCODE = 'RESB' then
                   add # [ OPERAND ] to LOCCTR
               else if OPCODE = 'BYTE' then
```

```
                begin
                  find length of constant in bytes
                  add length to LOCCTR
                end { if BYTE }
              else
                set error flag (invalid operation code)
          end { if not a comment}
        write line to intermediate file
        read next input line
    end {while not END}
```

Program

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

void main()
{
   char label[10],opcode[10],operand[10],symbol[10],code[10],mnemonic[3];
   int locctr,start,length,*sym_loc,sym_count=0;
   int error=0,op_found;
   char error_desc[100];
   char sym_tab[50][50];
   FILE *fp1,*fp2,*fp3,*fp4;

   fp1=fopen("input.txt","r");
   fp2=fopen("output.txt","w");
   fp3=fopen("optab.txt","r");
   fp4=fopen("symtab.txt","w");
   fscanf(fp1,"%s\t%s\t%s",label,opcode,operand);

   if(strcmp(opcode,"START")==0)
   {
      sscanf(operand,"%X",&start);
      locctr=start;
      fprintf(fp2,"\t%s\t%s\t%X\n",label,opcode,locctr);
      fscanf(fp1,"%s\t%s\t%s",label,opcode,operand);
   }
   else
   {
      locctr=0;
   }

   while(strcmp(opcode,"END")!=0 && error==0)
   {
      if(strcmp(label,"#")!=0)
      {
```

```
fprintf(fp2,"%4X\t",locctr);
if(strcmp(label,"*")!=0)
{
   for(int i=0;i<sym_count;i++)
   {
      if (strcmp(sym_tab[i],label)==0)
      {
         error=1;
         strcat(error_desc,"ERROR: Duplicate Symbol Found: ");
         strcat(error_desc,label);
         break;
      }
   }
   if(!error)
   {
      strcpy(sym_tab[sym_count],label);
      sym_count++;
      fprintf(fp4,"%s\t%X\n",label,locctr);
   }
}
fscanf(fp3,"%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,"RESB")==0)
{
   locctr+=(atoi(operand));
}
else if(strcmp(opcode,"BYTE")==0)
{
   locctr+=strlen(operand)-3;//C''
}
else
{
   op_found=0;
   while(fscanf(fp3,"%s\t%s",code,mnemonic)!=EOF)
   {
      if(strcmp(opcode,code)==0)
      {
         op_found=1;
         locctr+=3;
         break;
```

```
            }
          }
          rewind(fp3);
          if(op_found == 0)
          {
            error=1;
            strcat(error_desc,"ERROR Opcode cannot be found : ");
            strcat(error_desc,opcode);
            break;
          }
        }

        fprintf(fp2,"%s\t%s\t%s\n",label,opcode,operand);
        fscanf(fp1,"%s\t%s\t%s",label,opcode,operand);
      }
      else
      {
        fscanf(fp1,"%s\t%s\t%s",label,opcode,operand);
      }
    }
    if(error)
    {
      printf("%s\n",error_desc);
    }
    else
    {
      fprintf(fp2,"\t\t%s\n",opcode);
      length=locctr-start;
      printf("The length of the code : %d\n",length);
    }
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);
}
```

Sample Code

input.txt

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

optab.txt

```
ADD     18
AND     40
COMP    28
DIV     24
J       3C
JEQ     30
JGT     34
JLT     38
JSUB    48
LDA     00
LDCH    50
LDL     08
LDX     04
MUL     20
OR      44
RD      D8
RSUB    4C
STA     0C
STCH    54
STL     14
STSW    E8
STX     10
SUB     1C
TD      E0
TIX     2C
WD      DC
END
```

symtab.txt

```
ALPHA   1009
BETA    100C
GAMMA   100F
```

PassOne.c

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

void main()
{
    char label[10],opcode[10],operand[10],symbol[10],code[10],mnemonic[3];
    int locctr,start,length,*sym_loc,sym_count=0,size,prevlocctr=0;
    int error=0,op_found;
    char error_desc[100];
    char sym_tab[50][50];
    FILE *fp1,*fp2,*fp3,*fp4,*fp5;
```

```
fp1=fopen("input.txt","r");
fp2=fopen("intermediate.txt","w");
fp3=fopen("optab.txt","r");
fp4=fopen("symtab.txt","w");
fp5=fopen("length.txt","w");
fscanf(fp1,"%s\t%s\t%s",label,opcode,operand);

if(strcmp(opcode,"START")==0)
{
    sscanf(operand,"%X",&start);
    locctr=start;
    fprintf(fp2,"\t%s\t%s\t%X\n",label,opcode,locctr);
    fscanf(fp1,"%s\t%s\t%s",label,opcode,operand);
}
else
{
    locctr=0;
}

while(strcmp(opcode,"END")!=0 && error==0)
{
    if(strcmp(label,"#")!=0)
    {
        fprintf(fp2,"%4X\t",locctr);
        if(strcmp(label,"*")!=0)
        {
            for(int i=0;i<sym_count;i++)
            {
                if (strcmp(sym_tab[i],label)==0)
                {
                    error=1;
                    strcat(error_desc,"ERROR: Duplicate Symbol Found: ");
                    strcat(error_desc,label);
                    break;
                }
            }
            if(!error)
            {
                strcpy(sym_tab[sym_count],label);
                sym_count++;
                fprintf(fp4,"%s\t%X\n",label,locctr);
             }
         }
         fscanf(fp3,"%s\t%s",code,mnemonic);
         if(strcmp(opcode,"WORD")==0)
         {
             prevlocctr=locctr;
             locctr+=3;
         }
         else if(strcmp(opcode,"RESW")==0)
         {
             prevlocctr=locctr;
             locctr+=(3*(atoi(operand)));
         }
         else if(strcmp(opcode,"RESB")==0)
         {
             prevlocctr=locctr;
             locctr+=(atoi(operand));
         }
         else if(strcmp(opcode,"BYTE")==0)
         {
             prevlocctr=locctr;
             locctr+=strlen(operand)-3;//C''
         }
```

```
            else
            {
                op_found=0;
                while(fscanf(fp3,"%s\t%s",code,mnemonic)!=EOF)
                {
                    if(strcmp(opcode,code)==0)
                    {
                        op_found=1;
                        prevlocctr=locctr;
                        locctr+=3;
                        break;
                    }
                }
                rewind(fp3);
                if(op_found == 0)
                {
                    error=1;
                    strcat(error_desc,"ERROR Opcode cannot be found : ");
                    strcat(error_desc,opcode);
                    break;
                }
            }

            fprintf(fp2,"%s\t%s\t%s\n",label,opcode,operand);
            fscanf(fp1,"%s\t%s\t%s",label,opcode,operand);
        }
        else
        {
            fscanf(fp1,"%s\t%s\t%s",label,opcode,operand);
        }
    }
    if(error)
    {
        printf("%s\n",error_desc);
    }
    else
    {
        fprintf(fp2,"*\t*\t%s\t*\n",opcode);
        length=locctr-start;
        size=prevlocctr-start;
        fprintf(fp5,"%X\t%X\n",length,size);
         printf("The length of the code : %d\n",length);
    }
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);
    fclose(fp5);
}
```

Output

intermediate.txt

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

length.txt

```
12      F
```

Result

The algorithm was developed and the program was coded. The program was tested successfully.

Exp. NO: 5 **JESSIN SUNNY**

Date: 24/09/2024 **ROLL NO: 37**

PASS TWO OF A TWO PASS ASSEMBLER

Aim: Implement pass two of a two pass assembler

Algorithm:

```
begin
   read first input line {from intermediate file}
   if OPCODE = 'START' then
      begin
         write listing line
         read next input line
      end { if START }
   write Header record to object program
   initialize first Text record

   write last Text record to object program
   write End record to object program
   write last listing line
end { PASS 2 }
while OPCODE != 'END' do
   begin
      if this is not a comment line then
         begin
            search OPTAB for OPCODE
             if found then
               begin
                  if there is a SYMBOL in LABEL field then
                     begin
                        search SYMTAB for OPERAND
                        if found then
                           store symbol value as operand address
                        else
                           begin
                              store 0 as operand address
                              set error flag (undefined symbol)
                           end
                     end {if symbol}
                  else
                     store 0 as operand address
                     assemble the object code instruction
               end {if opcode found}
            else if OPCODE = 'BYTE' or 'WORD' then
```

```
            convert constant to object code
        if object code will not fit into the current Text record then
          begin
            write Text record to object program
            initialize new Text record
          end
        add object code to Text record
      end { if not a comment}
    write listing line
    read next input line
  end {while not END}
```

Program

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

char
t1[20],t2[20],t3[20],t4[10],address[10],label[10],opcode[10],operand[10],length[10],size[10]
,a[10],ad[10],st_addr[10];
int s=-1,o=-1,i,j;

FILE *fp1,*fp2,*fp3,*fp4,*fp5,*fp6;

struct SYNTAB
{
  char label[10];
  char addr[10];
} ST[30];

struct OPTAB
{
  char opcode[10];
  char hexcode[10];
} OT[30];

void read_optab()
{
  while(1)
  {
    o++;
    fscanf(fp3,"%s %s",OT[o].opcode,OT[o].hexcode);
    if (getc(fp3) == EOF)
      {
      break;
    }
```

```
    }
}

void read_symtab()
{
    while (1)
    {
        s++;
        fscanf(fp2, "%s %s", ST[s].label, ST[s].addr);
        if (getc(fp2) == EOF)
            {
            break;
        }
    }
}

void read_line()
{
    strcpy(t1,"");
    strcpy(t2,"");
    strcpy(t3,"");
    strcpy(t4,"");

    fscanf(fp1,"%s",t1);
    if (getc(fp1)!='\n')
    {
        fscanf(fp1,"%s",t2);
        if (getc(fp1)!='\n')
            {
            fscanf(fp1, "%s", t3);
            if (getc(fp1) != '\n')
                {
                fscanf(fp1, "%s", t4);
            }
        }
    }

    if (strcmp(t1,"")!=0 && strcmp(t2,"")!=0 && strcmp(t3,"")!=0 && strcmp(t4,"") != 0)
    {
        strcpy(address, t1);
        strcpy(label, t2);
        strcpy(opcode, t3);
        strcpy(operand, t4);
    }
    else if (strcmp(t1,"")!=0 && strcmp(t2,"")!=0 && strcmp(t3,"")!=0 && strcmp(t4,"")==0)
    {
        strcpy(address, t1);
```

```
        strcpy(label, "");
        strcpy(opcode, t2);
        strcpy(operand, t3);
    }
    else if (strcmp(t1,"")!=0 && strcmp(t2,"")!=0 && strcmp(t3,"")==0 && strcmp(t4,"")==0)
    {
        if (strcmp(t1,"END")==0)
            {
            strcpy(address,"");
            strcpy(label,"");
            strcpy(opcode,t1);
            strcpy(operand,t2);
        }
            else
            {
            strcpy(address,t1);
            strcpy(label,"");
            strcpy(opcode,t2);
            strcpy(operand, "");
        }
    }
}

int main()
{
    fp1=fopen("intermediate.txt", "r");
    fp2=fopen("symtab.txt", "r");
    fp3=fopen("optab.txt", "r");
    fp4=fopen("length.txt", "r");
    fp5=fopen("output.txt", "w");
    fp6=fopen("object_program.txt", "w");

    if (fp1==NULL||fp2==NULL||fp3==NULL||fp4==NULL||fp5==NULL||fp6==NULL)
    {
        printf("Error opening files\n");
        return 1;
    }

    fscanf(fp4,"%s\t%s\t",length,size);

    read_optab();
    read_symtab();

    fscanf(fp1,"%s\t%s\t%s\t",label,opcode,operand);
    strcpy(st_addr,operand);

    if (strcmp(opcode,"START")==0)
```

```
  {
    fprintf(fp5,"%s\t%s\t%s\t%s\n",address,label,opcode,operand); // Print header line in
assembly listing
    fprintf(fp6,"H^%s^00%s^0000%s\n",label,operand,length);
    fprintf(fp6,"T^00%s^%s",operand,size);
    read_line();
  }

  while (strcmp(opcode,"END")!=0)
  {
    if (strcmp(opcode,"BYTE")==0)
        {
       // Handle BYTE directive
       if (operand[0]=='C')
           {
         fprintf(fp5,"%s\t%s\t%s\t%s\t",address,label,opcode,operand); // Print assembly
         for (i=2;operand[i]!='\'';i++)
                 {
           sprintf(ad,"%x",operand[i]);  // Convert character to hex
           fprintf(fp5,"%s",ad);        // Print object code
           fprintf(fp6,"^%s",ad);       // Write object code to object file
         }
         fprintf(fp5, "\n");
       }
           else if(operand[0] == 'X')
           {
         strncpy(ad,operand + 2,strlen(operand) - 3);  // Extract hex value
         fprintf(fp5,"%s\t%s\t%s\t%s\t%s\n",address,label,opcode,operand,ad);
         fprintf(fp6,"^%s",ad);
       }
    }
        else if(strcmp(opcode, "WORD")==0)
        {
       // Handle WORD directive
       sprintf(a, "%06x", atoi(operand));  // Convert to 6-digit hex
       fprintf(fp5, "%s\t%s\t%s\t%s\t%s\n", address, label, opcode, operand, a);
       fprintf(fp6, "^%s", a);
    }
        else if (strcmp(opcode, "RESW") == 0 || strcmp(opcode, "RESB") == 0)
        {
       // For RESW and RESB, print the line but no object code is generated
       fprintf(fp5, "%s\t%s\t%s\t%s\n", address, label, opcode, operand);
    }
        else
        {
       // Look for opcode in OPTAB
       i = 0;
```

```
        while (i <= o && strcmp(opcode, OT[i].opcode) != 0)
            {
           i++;
        }

        if (i > o)
            {
           printf("Error: Opcode %s not found in OPTAB\n", opcode);
        }
            else
            {
           // Look for operand in SYMTAB
           j = 0;
           while (j <= s && strcmp(operand, ST[j].label) != 0)
                   {
              j++;
           }

           if (j > s)
                   {
              printf("Error: Operand %s not found in SYMTAB\n",operand);
           }
                   else
                   {

fprintf(fp5,"%s\t%s\t%s\t%s\t%s%s\n",address,label,opcode,operand,OT[i].hexcode,ST[j].ad
dr);
              fprintf(fp6,"^%s%s",OT[i].hexcode,ST[j].addr);
           }
        }
     }
     read_line();  // Move to the next line in the intermediate file.
  }

  fprintf(fp5,"%s\t%s\t%s\t%s\t%s\n","*",label,opcode,operand,"*");
  fprintf(fp6,"\nE^00%s\n",st_addr);  // Write end record to object code

  fclose(fp1);
  fclose(fp2);
  fclose(fp3);
  fclose(fp4);
  fclose(fp5);
  fclose(fp6);

  return 0;
}
```

Sample Code

optab.txt

```
ADD     18
AND     40
COMP    28
DIV     24
J       3C
JEQ     30
JGT     34
JLT     38
JSUB    48
LDA     00
LDCH    50
LDL     08
LDX     04
MUL     20
OR      44
RD      D8
RSUB    4C
STA     0C
STCH    54
STL     14
STSW    E8
STX     10
SUB     1C
TD      E0
TIX     2C
WD      DC
END
```

symtab.txt

```
ALPHA   1009
BETA    100C
GAMMA   100F
```

intermediate.txt

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

length.txt

```
12      F
```

PassTwo.c

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

char t1[20],t2[20],t3[20],t4[10],address[10],label[10],opcode[10],operand[10],length[10],size[10],a[10],ad[10],st_addr[10];
int s=-1,o=-1,i,j;

FILE *fp1,*fp2,*fp3,*fp4,*fp5,*fp6;

struct SYNTAB
{
    char label[10];
    char addr[10];
} ST[30];

struct OPTAB
{
    char opcode[10];
    char hexcode[10];
} OT[30];

void read_optab()
{
    while(1)
    {
        o++;
        fscanf(fp3,"%s %s",OT[o].opcode,OT[o].hexcode);
        if (getc(fp3) == EOF)
        {
            break;
        }
    }
}

void read_symtab()
{
    while (1)
    {
        s++;
        fscanf(fp2, "%s %s", ST[s].label, ST[s].addr);
```


```c
void read_symtab()
{
    while (1)
    {
        s++;
        fscanf(fp2, "%s %s", ST[s].label, ST[s].addr);
        if (getc(fp2) == EOF)
        {
            break;
        }
    }
}
```


```c
void read_line()
{
    strcpy(t1,"");
    strcpy(t2,"");
    strcpy(t3,"");
    strcpy(t4,"");

    fscanf(fp1,"%s",t1);
    if (getc(fp1)!='\n')
    {
        fscanf(fp1,"%s",t2);
        if (getc(fp1)!='\n')
        {
            fscanf(fp1, "%s", t3);
            if (getc(fp1) != '\n')
            {
                fscanf(fp1, "%s", t4);
            }
        }
    }

    if (strcmp(t1,"")!=0 && strcmp(t2,"")!=0 && strcmp(t3,"")!=0 && strcmp(t4,"") != 0)
    {
        strcpy(address, t1);
        strcpy(label, t2);
        strcpy(opcode, t3);
        strcpy(operand, t4);
    }
```

```
    else if (strcmp(t1,"")!=0 && strcmp(t2,"")!=0 && strcmp(t3,"")!=0 && strcmp(t4,"")==0)
    {
        strcpy(address, t1);
        strcpy(label, "");
        strcpy(opcode, t2);
        strcpy(operand, t3);
    }
    else if (strcmp(t1,"")!=0 && strcmp(t2,"")!=0 && strcmp(t3,"")==0 && strcmp(t4,"")==0)
    {
        if (strcmp(t1,"END")==0)
        {
            strcpy(address,"");
            strcpy(label,"");
            strcpy(opcode,t1);
            strcpy(operand,t2);
        }
        else
        {
            strcpy(address,t1);
            strcpy(label,"");
            strcpy(opcode,t2);
            strcpy(operand, "");
        }
    }
}

int main()
{
    fp1=fopen("intermediate.txt", "r");
    fp2=fopen("symtab.txt", "r");
    fp3=fopen("optab.txt", "r");
    fp4=fopen("length.txt", "r");
    fp5=fopen("output.txt", "w");
    fp6=fopen("object_program.txt", "w");

    if (fp1==NULL||fp2==NULL||fp3==NULL||fp4==NULL||fp5==NULL||fp6==NULL)
    {
        printf("Error opening files\n");
        return 1;
    }
    fscanf(fp4,"%s\t%s\t",length,size);

    read_optab();
    read_symtab();

    fscanf(fp1,"%s\t%s\t%s\t",label,opcode,operand);
    strcpy(st_addr,operand);

    if (strcmp(opcode,"START")==0)
    {
        fprintf(fp5,"%s\t%s\t%s\t%s\n",address,label,opcode,operand); // Print header line in assembly listing
        fprintf(fp6,"H^%s^00%s^0000%s\n",label,operand,length);
        fprintf(fp6,"T^00%s^%s",operand,size);
        read_line();
    }

    while (strcmp(opcode,"END")!=0)
    {
        if (strcmp(opcode,"BYTE")==0)
        {
            // Handle BYTE directive
            if (operand[0]=='C')
            {
                fprintf(fp5,"%s\t%s\t%s\t%s\t",address,label,opcode,operand); // Print assembly
                for (i=2;operand[i]!='\'';i++)
                {
                    sprintf(ad,"%x",operand[i]);  // Convert character to hex
                    fprintf(fp5,"%s",ad);         // Print object code
                    fprintf(fp6,"^%s",ad);        // Write object code to object file
                }
                fprintf(fp5, "\n");
            }
            else if(operand[0] == 'X')
            {
                strncpy(ad,operand + 2,strlen(operand) - 3);  // Extract hex value
                fprintf(fp5,"%s\t%s\t%s\t%s\t%s\n",address,label,opcode,operand,ad);
                fprintf(fp6,"^%s",ad);
            }
        }
```

```
        else if(strcmp(opcode, "WORD")==0)
        {
            // Handle WORD directive
            sprintf(a, "%06x", atoi(operand));  // Convert to 6-digit hex
            fprintf(fp5, "%s\t%s\t%s\t%s\n", address, label, opcode, operand, a);
            fprintf(fp6, "^%s", a);
        }
        else if (strcmp(opcode, "RESW") == 0 || strcmp(opcode, "RESB") == 0)
        {
            // For RESW and RESB, print the line but no object code is generated
            fprintf(fp5, "%s\t%s\t%s\t%s\n", address, label, opcode, operand);
        }
        else
        {
            // Look for opcode in OPTAB
            i = 0;
            while (i <= o && strcmp(opcode, OT[i].opcode) != 0)
            {
                i++;
            }

            if (i > o)
            {
                printf("Error: Opcode %s not found in OPTAB\n", opcode);
            }
            else
            {
                // Look for operand in SYMTAB
                j = 0;
                while (j <= s && strcmp(operand, ST[j].label) != 0)
                {
                    j++;
                }

                if (j > s)
                {
                    printf("Error: Operand %s not found in SYMTAB\n",operand);
                }
                else
                {
                    fprintf(fp5,"%s\t%s\t%s\t%s\t%s%s\n",address,label,opcode,operand,OT[i].hexcode,ST[j].addr);
                     fprintf(fp6,"^%s%s",OT[i].hexcode,ST[j].addr);
                 }
             }
         }
         read_line();  // Move to the next line in the intermediate file.
    }

    fprintf(fp5,"%s\t%s\t%s\t%s\t%s\n","*",label,opcode,operand,"*");
    fprintf(fp6,"\nE^00%s\n",st_addr);  // Write end record to object code

    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
    fclose(fp4);
    fclose(fp5);
    fclose(fp6);

    return 0;
}
```

Output

output.txt

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

object_program.txt

```
H^PGM1^001000^000012
T^001000^F^001009^20100C^0C100F^000002^000004
E^001000
```

Result

The algorithm was developed and the program was coded. The program was tested successfully.

Exp. NO: 6 **JESSIN SUNNY**

Date: 16/10/2024 **ROLL NO: 37**

SINGLE PASS ASSEMBLER

Aim: Implement a single pass assembler

Algorithm:

```
begin
  read first input line
  if OPCODE = 'START' then
    begin
      save # [ OPERAND ] as starting address
      initialize LOCCTR to starting address
      read next input line
    end { if START }
  else
    initialize LOCCTR to 0
while OPCODE != 'END' do
  begin
    if there is not a comment line then
      begin
        if there is a symbol in the LABEL field then
          begin
            search SYMTAB for LABEL
            if found then
              begin
                if symbol value as null
                set symbol value as LOCCTR and search the linked list with the
                corresponding operand
                PTR addresses and generate operand addresses as corresponding symbol
                values
                Set symbol value as LOCCTR in symbol table and delete the linked list
              end
            else
              insert {LABEL, LOCCTR} into SYMTAB
          end { if symbol }
          search OPTAB for OPCODE
          if found then
            begin
              search SYMTAB for OPERAND address
              if found then
                if symbol value not equal to null then
                  store symbol value as OPERAND address
                else
```

```
                    insert at the end of the linked list with a node with address as LOCCTR
                else
                    insert (symbol name, null)
                    add 3 to LOCCTR
                end
            else if OPCODE = 'WORD' then
                add 3 to LOCCTR & convert to object code
            else if OPCODE = 'RESW' then
                add 3 * # [ OPERAND ] to LOCCTR
            else if OPCODE = 'RESB' then
                add # [ OPERAND ] to LOCCTR
            else if OPCODE = 'BYTE' then
                begin
                    find length of constant in bytes
                    add length to LOCCTR
                    convert constant to object code
                end { if BYTE }
            if object code will not fit into current text record then
                begin
                    write text record to obect program
                    initialize new text record
                end
            add object code to Text record
        end { if not a comment}
    write listing line
    read next input line
  end {while not END}
  write last Text record to object program
  write End record to object program
  write last listing line
end {Pass 1}
```

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define MAX 10

int s = 0, i;
struct adr
{
  int addr;
  struct adr *next;
};

typedef struct
```

```
{
   char item[10];
   struct adr *link;
   int address;
} symbol;


struct adr *temps, *temps2;

symbol table[10];
void inittable()
{
   for (i = 0; i < 10; i++)
      strcpy(table[i].item, "");
}
int issymbolpresent(char *label)
{
   for (i = 0; i < 10; i++)
   {
   // printf("%d.",i);
      if (strcmp(table[i].item, label) == 0)
      {
         if (table[i].address == -1)
            return i; // address not present
         else
            return -1; // present
      }
   }
  // printf(";;");
   return -2; // not present
}

char *insertsymboladdress(char *label, int address)
{
   int f = 0;
   f = issymbolpresent(label);
   //printf("%d.",f);
   if (f == -2)
   {
      for (i = 0; i < 10; i++)
      {
         if (strcmp(table[i].item, "") == 0){
            f = i;
            break;
         }
      printf("%d",i);
      }
```

```
        if (f != -2)
        {
            strcpy(table[f].item, label);
            table[f].address = address;
            table[f].link = NULL;
        }
    }
    else if (f != -1)
    {
        temps = table[f].link;
        table[f].address = address;
        char *mrecord = (char *)malloc(64);
        char from_address[10], to_address[10];
        sprintf(from_address, "%04X", address);

        mrecord[0] = '\0';
        while (temps != NULL)
        {
            strcat(mrecord, "T");
            sprintf(to_address, "%06X", temps->addr);
            strcat(mrecord, to_address);
            strcat(mrecord, "02");
            strcat(mrecord, from_address);
            strcat(mrecord, "\n");
            temps2 = temps;
            temps = temps->next;
            free(temps2);
        }
        //printf("mrcrd\n");
        return mrecord;
    }
    return NULL;
}

int addsymbol_no_address(char *label, int address)
{

    for (i = 0; i < 10; i++)
    {
        if (strcmp(table[i].item, "") == 0)
        {
            strcpy(table[i].item, label);
            table[i].address = -1;
            temps = (struct adr *)malloc(sizeof(struct adr));
            table[i].link = temps;
            temps->addr = address;
            temps->next = NULL;
```

```
            break;
        }
    }
    return (i != 10);
}

char *getconstant(char str1[])
{
    int p, i, l = strlen(str1);
    char *out = malloc(20), temp[5];
    out[0] = '\0';
    for (i = 2; i < l - 1; i++)
    {
        p = str1[i];
        sprintf(temp, "%X", p);
        strcat(out, temp);
    }
    return out;
}

char *getopcode(char opcode[])
{
    FILE *optable = fopen("optable.txt", "r");
    char *value = malloc(10), code[10], line[20];
    while (fgets(line, sizeof(line), optable))
    {
        //printf("%s-- %s\n",line,opcode);
        sscanf(line, "%s %s", value, code);
        if (strcmp(code, opcode) == 0)
        {
            fclose(optable);
            return value;
        }
    }
    fclose(optable);
    return "ff";
}

int hex_int(char * num)
{
    int i, hex = 0,ln,val;
    ln = strlen(num);
    for(i=ln-1;i>=0;i--){
      if(num[i]>='0' && num[i]<='9')
        val = num[i]-'0';
      else if(num[i]>='a' && num[i]<='f')
        val = num[i]-87;
```

```
    else if(num[i]>='A' && num[i]<='F')
      val = num[i]-55;
    hex += (pow((double)16, (double)ln-i-1)) * val;
    //printf("%d,  %c\n",hex,num[i]);
  }
  return hex;
}

void onepass(FILE *input, FILE *output)
{
  inittable();
  char line[64], label[10], operand[10], opcode[10];
  char str1[10], str2[10], str3[10], locctr_str[10], start_str[10], tempstr[10];
  char text[64], header[20], end[10], object[10];
  int start, locctr, f, i, cnt = 0, flag = 0;
  char file[1024], *mrecord;
  file[0] = '\0';
  strcpy(header, "H");
  strcpy(label, "NONAME");
  fgets(line, sizeof(line), input);
  str3[0] = '\0';
  sscanf(line, " %s %s %s ", str1, str2, str3);
  if (str3[0] == '\0')
  {
    strcpy(opcode, str1);
    strcpy(operand, str2);
  }
  else
  {
    strcpy(label, str1);
    strcpy(opcode, str2);
    strcpy(operand, str3);
  }
  if (strcmp(opcode, "START") == 0)
  {
    start = hex_int(operand);
    locctr = start;
    strcpy(start_str, operand);
    sprintf(tempstr, "%06X", start);
    fgets(line, sizeof(line), input);
    strcat(header, label);
    strcat(header, tempstr);
  }
  else
    start = locctr = 0;

  strcpy(text, "T");
```

```
strcat(text, tempstr);
strcat(text, "00");
strcpy(end, "E");
strcat(end, tempstr);

//printf("pass 1 do function\n");
do
{
   str3[0] = '\0';
   sscanf(line, " %s %s %s ", str1, str2, str3);
   if (strcmp(".", str1) == 0)
      continue;
   if (str3[0] == '\0')
   {   //printf("<>");
      strcpy(opcode, str1);
      strcpy(operand, str2);
      strcpy(label, "\t");
   }
   else
   {
      strcpy(label, str1);
      strcpy(opcode, str2);
      strcpy(operand, str3);
      int f = issymbolpresent(label);
      //printf("*");
      if (f == -1)
      {
         printf("Symbol already exist : [%s]\n", label);
         break;
      }
      else
      {
         //printf("%s",label);
         mrecord = insertsymboladdress(label, locctr);
         if (mrecord != NULL)
         {
            flag = 1;
         }
      }
   }
   //printf("--\n");
   if (strcmp(opcode, "END") == 0)
   {
      //fprintf(output, "\t\tEND\t%s\n", start_str);
      break;
   }
   sprintf(locctr_str, "%X", locctr);
```

```
// fprintf(output, "%s\t%s\t%s\t%s\n", locctr_str, str1, str2, str3);
char op_num[3];
strcpy(op_num, getopcode(opcode));

if (strcmp(op_num, "ff") != 0)
{
   strcpy(object, op_num);
   f = issymbolpresent(operand);
   if (f == -2)
   {
      addsymbol_no_address(operand, locctr + 1);
      strcat(object, "0000");
   }
   else if (f == -1)
   {
      for (i = 0; i < 10; i++)
      {
         if (strcmp(table[i].item, operand) == 0)
         {
            sprintf(tempstr, "%06X", table[i].address);
            strcat(object, tempstr);
            break;
         }
      }
   }
   if (f > -1)
   {
      strcat(object, "0000");
      temps = (struct adr *)malloc(sizeof(struct adr));
      temps2 = table[f].link;
      while (temps2->next != NULL)
      {
         temps2 = temps2->next;
      }
      temps2->next = temps;
      temps->addr = locctr + 1;
      temps->next = NULL;
   }
   //strcat(text, object);
   cnt += 3;
   if (cnt >= 60)
      {
         sprintf(tempstr, "%02X", cnt);
         text[7] = tempstr[0];
         text[8] = tempstr[1];
        // printf("[%s]\n", text);
         strcat(file, text);
```

```
            sprintf(tempstr, "%06X", locctr);
            strcpy(text, "T");
            strcat(text, tempstr);
            strcat(text, "00");
            cnt = 0;
         }
      locctr += 3;
      strcat(text, object);

   }

   else
   {
      if (strcmp(opcode, "WORD") == 0)
      {
         sprintf(tempstr, "%06d", atoi(operand));
         strcpy(object, tempstr);
         line[strlen(line) - 1] = '\0';
         cnt += 3;
         if (cnt >= 60)
         {
            sprintf(tempstr, "%02X", cnt);
            text[7] = tempstr[0];
            text[8] = tempstr[1];
           // printf("[%s]\n", text);
            strcat(file, text);
            sprintf(tempstr, "%06X", locctr);
            strcpy(text, "T");
            strcat(text, tempstr);
            strcat(text, "00");
            cnt = 0;
         }
         strcat(text, object);
         locctr += 3;
      }
      else if (strcmp(opcode, "RESW") == 0)
      {
         locctr += atoi(operand) * 3;
      }
      else if (strcmp(opcode, "RESB") == 0)
      {
         locctr += atoi(operand);
      }
      else if (strcmp(opcode, "BYTE") == 0)
      {
         int len = strlen(getconstant(operand));
         if (cnt + len >= 60)
```

```
            {
               sprintf(tempstr, "%02X", cnt);
               text[7] = tempstr[0];
               text[8] = tempstr[1];
              // printf("[%s]\n", text);
               strcat(file, text);
               strcpy(text, "T");
               strcat(text, str1);
               strcat(text, "00");
               locctr += len / 2;
               cnt = 0;
            }
            else
            {
               strcat(text, getconstant(operand));
               cnt += len;
            }
         }

         else
         {
            printf("FATAL ERROR!!");
            exit(0);
         }
      }
      if (flag)
      {
         //printf("%s", mrecord);
         if (cnt > 0)
         {
            sprintf(tempstr, "%02X", cnt);
            text[7] = tempstr[0];
            text[8] = tempstr[1];
            strcat(file, text);
            strcat(file, "\n");
            cnt = 0;
         }
         strcat(file, mrecord);
         sprintf(locctr_str, "%06X", locctr);
         strcpy(text, "T");
         strcat(text, locctr_str);
         strcat(text, "00");
         flag = 0;
      }
   } while (fgets(line, sizeof(line), input));
   printf("Total length : %d\n", locctr - start);
   if (cnt > 0)
```

```
    {
        sprintf(tempstr, "%02X", cnt);
        text[7] = tempstr[0];
        text[8] = tempstr[1];
        strcat(file, text);
        strcat(file, " ab \n");
        cnt = 0;
    }
    sprintf(tempstr, "%06X\n", locctr - start);
    strcat(header, tempstr);
    printf("%s", header);
    printf("%s", file);
    printf("%s\n", end);

    fprintf(output, "%s%s%s\n", header, file, end);

}

int main()
{
    FILE *input, *output, *symtable;
    input = fopen("input.txt", "r");
    output = fopen("output.txt", "w");
    onepass(input, output);
    fclose(input);
    fclose(output);
    char str[10];
    int i;
    // for (i = 0; i < 10; i++)
    // {
    //     printf("%s  --  %d\n", table[i].item, table[i].address);
    // }
    return 0;
}
```

Sample Code

input.txt

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

optable.txt

```
18      ADD
00      LDA
20      MUL
0C      STA
END
```

OnePass.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define MAX 10

int s = 0, i;
struct adr
{
    int addr;
    struct adr *next;
};

typedef struct
{
    char item[10];
    struct adr *link;
    int address;
} symbol;


struct adr *temps, *temps2;

symbol table[10];
void inittable()
{
    for (i = 0; i < 10; i++)
        strcpy(table[i].item, "");
}
int issymbolpresent(char *label)
{
    for (i = 0; i < 10; i++)
    {
    // printf("%d.",i);
        if (strcmp(table[i].item, label) == 0)
        {
            if (table[i].address == -1)
                return i; // address not present
            else
                return -1; // present
        }
    }
   // printf(";;");
    return -2; // not present
}
char *insertsymboladdress(char *label, int address)
{
    int f = 0;
    f = issymbolpresent(label);
    //printf("%d.",f);
    if (f == -2)
    {
        for (i = 0; i < 10; i++)
        {
            if (strcmp(table[i].item, "") == 0){
                f = i;
                break;
            }
        printf("%d",i);
        }
        if (f != -2)
        {
            strcpy(table[f].item, label);
            table[f].address = address;
            table[f].link = NULL;
        }
    }
```

```
    else if (f != -1)
    {
        temps = table[f].link;
        table[f].address = address;
        char *mrecord = (char *)malloc(64);
        char from_address[10], to_address[10];
        sprintf(from_address, "%04X", address);

        mrecord[0] = '\0';
        while (temps != NULL)
        {
            strcat(mrecord, "T");
            sprintf(to_address, "%06X", temps->addr);
            strcat(mrecord, to_address);
            strcat(mrecord, "02");
            strcat(mrecord, from_address);
            strcat(mrecord, "\n");
            temps2 = temps;
            temps = temps->next;
            free(temps2);
        }
        //printf("mrcrd\n");
        return mrecord;
    }
    return NULL;
}

int addsymbol_no_address(char *label, int address)
{

    for (i = 0; i < 10; i++)
    {
        if (strcmp(table[i].item, "") == 0)
        {
            strcpy(table[i].item, label);
            table[i].address = -1;
            temps = (struct adr *)malloc(sizeof(struct adr));
            table[i].link = temps;
            temps->addr = address;
            temps->next = NULL;
            break;
        }
    }
    return (i != 10);
}

char *getconstant(char str1[])
{
    int p, i, l = strlen(str1);
    char *out = malloc(20), temp[5];
    out[0] = '\0';
    for (i = 2; i < l - 1; i++)
    {
        p = str1[i];
        sprintf(temp, "%X", p);
        strcat(out, temp);
    }
    return out;
}
char *getopcode(char opcode[])
{
    FILE *optable = fopen("optable.txt", "r");
    char *value = malloc(10), code[10], line[20];
    while (fgets(line, sizeof(line), optable))
    {
        //printf("%s-- %s\n",line,opcode);
        sscanf(line, "%s %s", value, code);
        if (strcmp(code, opcode) == 0)
        {
            fclose(optable);
            return value;
        }
    }
    fclose(optable);
    return "ff";
}

int hex_int(char * num)
{
    int i, hex = 0,ln,val;
    ln = strlen(num);
    for(i=ln-1;i>=0;i--){
      if(num[i]>='0' && num[i]<='9')
        val = num[i]-'0';
      else if(num[i]>='a' && num[i]<='f')
        val = num[i]-87;
      else if(num[i]>='A' && num[i]<='F')
        val = num[i]-55;
      hex += (pow((double)16, (double)ln-i-1)) * val;
      //printf("%d,  %c\n",hex,num[i]);
    }
    return hex;
}
```

```c
void onepass(FILE *input, FILE *output)
{
    inittable();
    char line[64], label[10], operand[10], opcode[10];
    char str1[10], str2[10], str3[10], locctr_str[10], start_str[10], tempstr[10];
    char text[64], header[20], end[10], object[10];
    int start, locctr, f, i, cnt = 0, flag = 0;
    char file[1024], *mrecord;
    file[0] = '\0';
    strcpy(header, "H");
    strcpy(label, "NONAME");
    fgets(line, sizeof(line), input);
    str3[0] = '\0';
    sscanf(line, " %s %s %s ", str1, str2, str3);
    if (str3[0] == '\0')
    {
        strcpy(opcode, str1);
        strcpy(operand, str2);
    }
    else
    {
        strcpy(label, str1);
        strcpy(opcode, str2);
        strcpy(operand, str3);
    }
    if (strcmp(opcode, "START") == 0)
    {
        start = hex_int(operand);
        locctr = start;
        strcpy(start_str, operand);
        sprintf(tempstr, "%06X", start);
        fgets(line, sizeof(line), input);
        strcat(header, label);
        strcat(header, tempstr);
    }
    else
        start = locctr = 0;

    strcpy(text, "T");
    strcat(text, tempstr);
    strcat(text, "00");
    strcpy(end, "E");
    strcat(end, tempstr);

    //printf("pass 1 do function\n");
    do
    {
        str3[0] = '\0';
        sscanf(line, " %s %s %s ", str1, str2, str3);
        if (strcmp(".", str1) == 0)
            continue;
        if (str3[0] == '\0')
        {   //printf("<>");
            strcpy(opcode, str1);
            strcpy(operand, str2);
            strcpy(label, "\t");
        }
        else
        {
            strcpy(label, str1);
            strcpy(opcode, str2);
            strcpy(operand, str3);
            int f = issymbolpresent(label);
            //printf("*");
            if (f == -1)
            {
                printf("Symbol already exist : [%s]\n", label);
                break;
            }
            else
            {
                //printf("%s",label);
                mrecord = insertsymboladdress(label, locctr);
                if (mrecord != NULL)
                {
                    flag = 1;
                }
            }
        }
        //printf("--\n");
        if (strcmp(opcode, "END") == 0)
        {
            //fprintf(output, "\t\tEND\t%s\n", start_str);
            break;
        }
        sprintf(locctr_str, "%X", locctr);
        // fprintf(output, "%s\t%s\t%s\t%s\n", locctr_str, str1, str2, str3);
        char op_num[3];
        strcpy(op_num, getopcode(opcode));
```

```
if (strcmp(op_num, "ff") != 0)
{
    strcpy(object, op_num);
    f = issymbolpresent(operand);
    if (f == -2)
    {
        addsymbol_no_address(operand, locctr + 1);
        strcat(object, "0000");
    }
    else if (f == -1)
    {
        for (i = 0; i < 10; i++)
        {
            if (strcmp(table[i].item, operand) == 0)
            {
                sprintf(tempstr, "%06X", table[i].address);
                strcat(object, tempstr);
                break;
            }
        }
    }
    if (f > -1)
    {
        strcat(object, "0000");
        temps = (struct adr *)malloc(sizeof(struct adr));
        temps2 = table[f].link;
        while (temps2->next != NULL)
        {
            temps2 = temps2->next;
        }
        temps2->next = temps;
        temps->addr = locctr + 1;
        temps->next = NULL;
    }
    //strcat(text, object);
    cnt += 3;
    if (cnt >= 60)
        {
            sprintf(tempstr, "%02X", cnt);
            text[7] = tempstr[0];
            text[8] = tempstr[1];
           // printf("[%s]\n", text);
            strcat(file, text);
            sprintf(tempstr, "%06X", locctr);
            strcpy(text, "T");
            strcat(text, tempstr);
            strcat(text, "00");
            cnt = 0;
        }
    locctr += 3;
    strcat(text, object);

}

else
{
    if (strcmp(opcode, "WORD") == 0)
    {
        sprintf(tempstr, "%06d", atoi(operand));
        strcpy(object, tempstr);
       line[strlen(line) - 1] = '\0';
       cnt += 3;
       if (cnt >= 60)
       {
           sprintf(tempstr, "%02X", cnt);
           text[7] = tempstr[0];
           text[8] = tempstr[1];
          // printf("[%s]\n", text);
           strcat(file, text);
           sprintf(tempstr, "%06X", locctr);
           strcpy(text, "T");
           strcat(text, tempstr);
           strcat(text, "00");
           cnt = 0;
       }
       strcat(text, object);
       locctr += 3;
   }
   else if (strcmp(opcode, "RESW") == 0)
   {
       locctr += atoi(operand) * 3;
   }
   else if (strcmp(opcode, "RESB") == 0)
   {
       locctr += atoi(operand);
   }
```

```
            else if (strcmp(opcode, "BYTE") == 0)
            {
                int len = strlen(getconstant(operand));
                if (cnt + len >= 60)
                {
                    sprintf(tempstr, "%02X", cnt);
                    text[7] = tempstr[0];
                    text[8] = tempstr[1];
                   // printf("[%s]\n", text);
                    strcat(file, text);
                    strcpy(text, "T");
                    strcat(text, str1);
                    strcat(text, "00");
                    locctr += len / 2;
                    cnt = 0;
                }
                else
                {
                    strcat(text, getconstant(operand));
                    cnt += len;
                }
            }

            else
            {
                printf("FATAL ERROR!!");
                exit(0);
            }
        }
        if (flag)
        {
            //printf("%s", mrecord);
            if (cnt > 0)
            {
                sprintf(tempstr, "%02X", cnt);
                text[7] = tempstr[0];
                text[8] = tempstr[1];
                strcat(file, text);
                strcat(file, "\n");
                cnt = 0;
            }
            strcat(file, mrecord);
            sprintf(locctr_str, "%06X", locctr);
            strcpy(text, "T");
            strcat(text, locctr_str);
            strcat(text, "00");
            flag = 0;
        }
    } while (fgets(line, sizeof(line), input));
    printf("Total length : %d\n", locctr - start);
    if (cnt > 0)
    {
        sprintf(tempstr, "%02X", cnt);
        text[7] = tempstr[0];
        text[8] = tempstr[1];
        strcat(file, text);
        strcat(file, " ab \n");
        cnt = 0;
    }
    sprintf(tempstr, "%06X\n", locctr - start);
     strcat(header, tempstr);
     printf("%s", header);
     printf("%s", file);
     printf("%s\n", end);

     fprintf(output, "%s%s%s\n", header, file, end);

}

int main()
{
    FILE *input, *output, *symtable;
    input = fopen("input.txt", "r");
    output = fopen("output.txt", "w");
    onepass(input, output);
    fclose(input);
    fclose(output);
    char str[10];
    int i;
    // for (i = 0; i < 10; i++)
    // {
    //     printf("%s  --  %d\n", table[i].item, table[i].address);
    // }
    return 0;
}
```

Output

```
Total length : 18
HPGM1001000000012
T0010000C0000002000000C0000000002
T001001021009
T00100C03000004
T00100402100C
T00100702100F
E001000
```

Result

The algorithm was developed and the program was coded. The program was tested successfully.

MICROPROCESSOR LAB

I. **Assembly Language Programming Exercises/Experiments using 8086 Trainer kit**
II. **Exercises/Experiments using MASM (PC Required)**

Exp. NO: 1 **JESSIN SUNNY**

Date: 16/10/2024 **ROLL NO: 37**

ADDITION OF TWO 16 BIT NUMBERS

USING 8086 TRAINER KIT

Aim: To add two 16-bit numbers using 8086 trainer kit.

Algorithm:

Step 1: Clear the AX by performing AND operation with 0000
Step 2: Move the location where result is to be stored to BX
Step 3: Move the location of operand I to SI
Step 4: Move the location of operand 2 to DI
Step 5: Move the contents of SI to AX
Step 6: Add the contents of DI to AX
Step 7: Move the result to the location stored in BX
Step 8: Move OOOOH to AX
Step 9: Add the carry flag to AX
Step 10: Move the result to the location stored in [BX + 2]
Step 11: Halt

Program

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

Input

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

Output

0600 - DF
0601 - 5F
0602 - 01

Result

The algorithm was developed and the program was coded. The program was tested successfully.

Exp. NO: 2 **JESSIN SUNNY**

Date: 16/10/2024 **ROLL NO: 37**

SUBTRACTION OF TWO 16 BIT NUMBERS

USING 8086 TRAINER KIT

Aim: To subtract two 16-bit numbers using 8086 trainer kit.

Algorithm:

Step 1: Clear the carry flag
Step 2: Move the location where result is to be stored to BX
Step 3: Move the location of operand I to SI
Step 4: Move the location of operand 2 to DI
Step 5: Move the contents of SI to AX
Step 6: Subtract the contents of DI from AX including the borrow value
Step 7: Move the result to the location stored in BX
Step 8: Halt

Program

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

Input

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

Output

0900 - D8
0901 -F7

Result

The algorithm was developed and the program was coded. The program was tested successfully.

Exp. NO: 3 **JESSIN SUNNY**

Date: 16/10/2024 **ROLL NO: 37**

MULTIPLICATION OF TWO 16 BIT NUMBERS

USING 8086 TRAINER KIT

Aim: To multiply two 16-bit numbers using 8086 trainer kit.

Algorithm:

Step 1: Clear the carry flag
Step 2: Move the location where result is to be stored to BX
Step 3: Move the location of operand I to SI
Step 4: Move the location of operand 2 to DI
Step 5: Move the contents of SI to AX
Step 6: Move the contents of DI to CX
Step 7: Multiply CX to AX
Step 8: Move the result from AX to the location stored in BX
Step 9: Move the higher bits of result from DX to location stored in [BX+2]
Step 10: Halt

Program

ADDRESS	MNEMONICS
0400	CLC
0401	MOV BX,0900H
0404	MOV SI,0700H
0407	MOV DI,0800H
040A	MOV AX,[SI]
040C	MOV CX,[DI]
040E	MUL CX
0410	MOV [BX],AX
0412	MOV [BX+2],AX
0415	HLT

Input

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

Output

0700-9E
0701-74

0702-14
0703-03

Result

The algorithm was developed and the program was coded. The program was tested successfully.

Exp. NO: 4 **JESSIN SUNNY**

Date: 16/10/2024 **ROLL NO: 37**

DIVISION OF TWO 16 BIT NUMBERS

USING 8086 TRAINER KIT

Aim: To divide two 16-bit numbers using 8086 trainer kit.

Algorithm:

Step 1: Clear the carry flag.
Step 2: Move the location where result is to be stored to BX.
Step 3: Move the location of operand 1 to SI.
Step 4: Move the location of operand 2 to DI.
Step 5: Move the contents of SI to AX.
Step 6: Move the contents of DI to CX.
Step 7: Move 00 to CH.
Step 8: Divide CL from AX.
Step 9: Move the result from AX to the location stored in BX.
Step 10: Halt.

Program

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

Input

0750 - 43
0751 - 12
0800 - 21

Output

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

Result

The algorithm was developed and the program was coded. The program was tested successfully.

Exp. NO: 5 **JESSIN SUNNY**

Date: 16/10/2024 **ROLL NO: 37**

MAXIMUM OF N NUMBERS

USING 8086 TRAINER KIT

Aim: To find the maximum of n numbers using the 8086 trainer kit.

Algorithm:

Step 1: Clear the carry flag.
Step 2: Move the location where the result has to be stored to BX.
Step 3: Move the starting location of array to SI.
Step 4: Move the total number of elements in the array to CX.
Step 5: Move 00 to AL.
Step 6: Compare the contents of SI with AL.
Step 7: Jump to step 9 if above instruction satisfies.
Step 8: Else move the contents of SI to AL.
Step 9: Move 00 to CH
Step 10: Increment SI
Step 11: Continue the loop of comparing the contents of SI and AL till the counter reaches zero (LOOPNZ only loops when the zero flag is not set).
Step 12: Move the result, i.e. maximum number from AL to the location stored in BX.
Step 13: Halt

Program

ADDRESS	MNEMONICS
0400	CLC
0401	MOV BX,0700H
0404	MOV SI,0800H
0407	MOV CX,0005H
040A	MOV AL,00H
040C	CMP AL,[SI]
040E	JA 0414H
0410	MOV AL,[SI]
0412	MOV CH,00H
0414	INC SI
0415	LOOPNZ 040CH
0417	MOV [BX],AL
0419	HLT

Input

0800-77

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

Output

0700 – F1

Result

The algorithm was developed and the program was coded. The program was tested successfully.

Exp. NO: 6 **JESSIN SUNNY**

Date: 16/10/2024 **ROLL NO: 37**

SORTING NUMBERS IN ASCENDING ORDER

USING 8086 TRAINER KIT

Aim: To sort the numbers in ascending order using 8086 trainer kit.

Algorithm:

Step 1: Set the value of SI to 500.
Step 2: Load data from offset SI to register CL.
Step 3: Decrease value of register CL by 1.
Step 4: Set the value of SI to 500.
Step 5: Load data from offset SI to register CH. Decrease value of register CH by 1.
Step 6: Increase the value of SI by 1.
Step 7: Load value from offset SI to register AL.
Step 8: Increase the value of SI by 1.
Step 9: Compare the value of register AL and [SI],ie,(AL-[SI]).
Step 10: Jump to address 41C if carry is generated.
Step 11: Exchange the contents of register AL and SI.
Step 12: Decrease the value of SI by 1.
Step 13: Exchange the contents of register AL and SI
Step 14: Increase the value of SI by 1.
Step 15: Decrease the value of register CH by 1.
Step 16: Jump to address 40F if zero flat reset.
Step 17: Decrease the value of register CL by 1.
Step 18: Jump to address 407 if zero flat reset.
Step 19: Stop.

Program

ADDRESS	MNEMONICS
0400	MOV SI,500
0403	MOV CL,[SI]
0405	DEC CL
0407	MOV SI,500
0409	MOV CH,[SI]
040C	DEC CH
040E	INC SI
040F	MOV AL,[SI]
0411	INC SI
0412	CMP AL,[SI]
0414	JC 041C

0416	XCHG AL,[SI]
0418	DEC SI
0419	XCHG AL,[SI]
041B	INC SI
041C	DEC CH
041E	JNZ 40F
0420	DEC CL
0422	JNZ 407
0424	HLT

Input

0500 - 5
0501 - 6
0502 - 8
0503 - 3
0504 - 5
0505 - 4

Output

0500 - 5
0501 - 3
0502 - 4
0503 - 5
0504 - 6
0505 - 8

Result

The algorithm was developed and the program was coded. The program was tested successfully.

Exp. NO: 7 **JESSIN SUNNY**

Date: 30/10/2024 **ROLL NO: 37**

8-BIT ADDITION AND MULTIPLICATION USING MASM

Aim: To implement 8-bit addition and multiplication.

Input

Numbers to add

Output

Sum and Product

Addition

Algorithm

Step 1: Load data segment starting address to DS
Step 2: Get the first number and store it in AL
Step 3: Move the value to BL
Step 4: Get the second number and store it in AL
Step 5: ADD BL and AL and store it in AL
Step 6: Correct the Value and convert it into BCD using AAA
Step 7: Move value in AX to BX
Step 8: Print the value
Step 9: Stop

Program

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
    M1 DB 10,13,"Enter first number:$"
    M2 DB 10,13,"Enter second number:$"
    M3 DB 10,13,"Sum:$"
DATA ENDS
PRTMSG MACRO MESSAGE
    LEA DX,MESSAGE
    MOV AH,09
    INT 21H
    ENDM
GETDCM MACRO
    MOV AH,01
    INT 21H
    SUB AL,30H
```

```
    ENDM
CODE SEGMENT
START:MOV AX,DATA
    MOV DS,AX
    PRTMSG M1
    GETDCM
    MOV BL,AL
    PRTMSG M2
    GETDCM
    ADD AL,BL
    MOV AH,00H
    AAA
    MOV BX,AX
    PRTMSG M3
    MOV DL,BH
    ADD DL,30H
    MOV AH,02
    INT 21H
    MOV DL,BL
    ADD DL,30H
    INT 21H
    MOV AH,4CH
    INT 21H
CODE ENDS
END START
```

Multiplication

Algorithm:

Step 1: Load data segment starting address to DS
Step 2: Get the first number and store it in AL
Step 3: Move the value to BL
Step 4: Get the second number and store it in AL
Step 5: MUL BL and AL and store it in AX
Step 6: Correct the Value and convert it into BCD using AAM
Step 7: Move value in AX to BX
Step 8: Print the value
Step 9: Stop

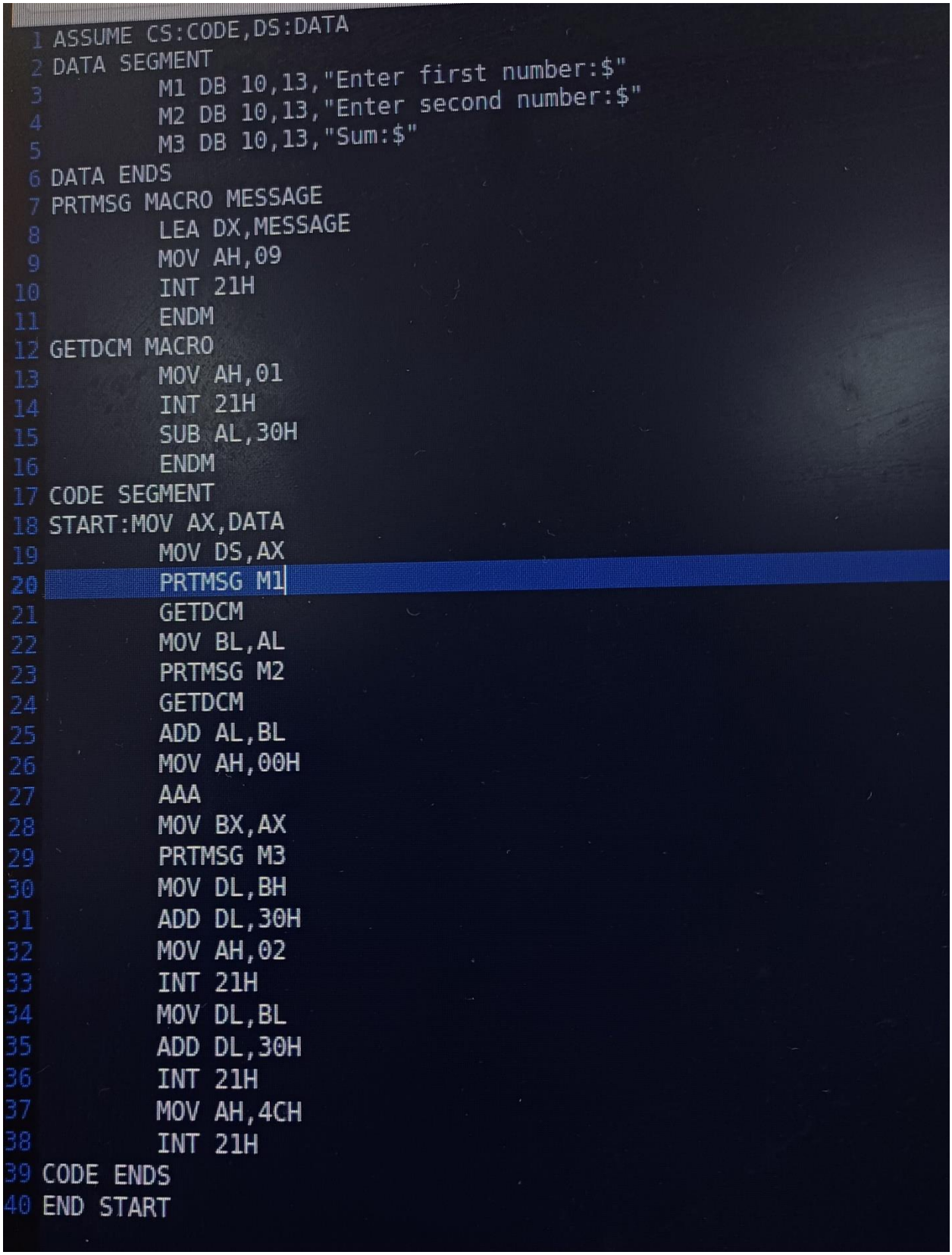
Program

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

PRTMSG MACRO MESSAGE
  LEA DX,MESSAGE
```

```
   MOV AH,09
   INT 21H
   ENDM

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

CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
   MOV DS,AX
   PRTMSG M1
   GETDCM
   MOV BL,AL
   GETDCM
   MOV AH,00H
   MUL BL
   AAM
   MOV BX,AX
   PRTMSG M2
   MOV DL,BH
   OR DL,30H
   MOV AH,02H
   INT 21H
   MOV DL,BL
   OR DL,30H
   INT 21H
   MOV AH,4CH
   INT 21H
CODE ENDS
END START
```

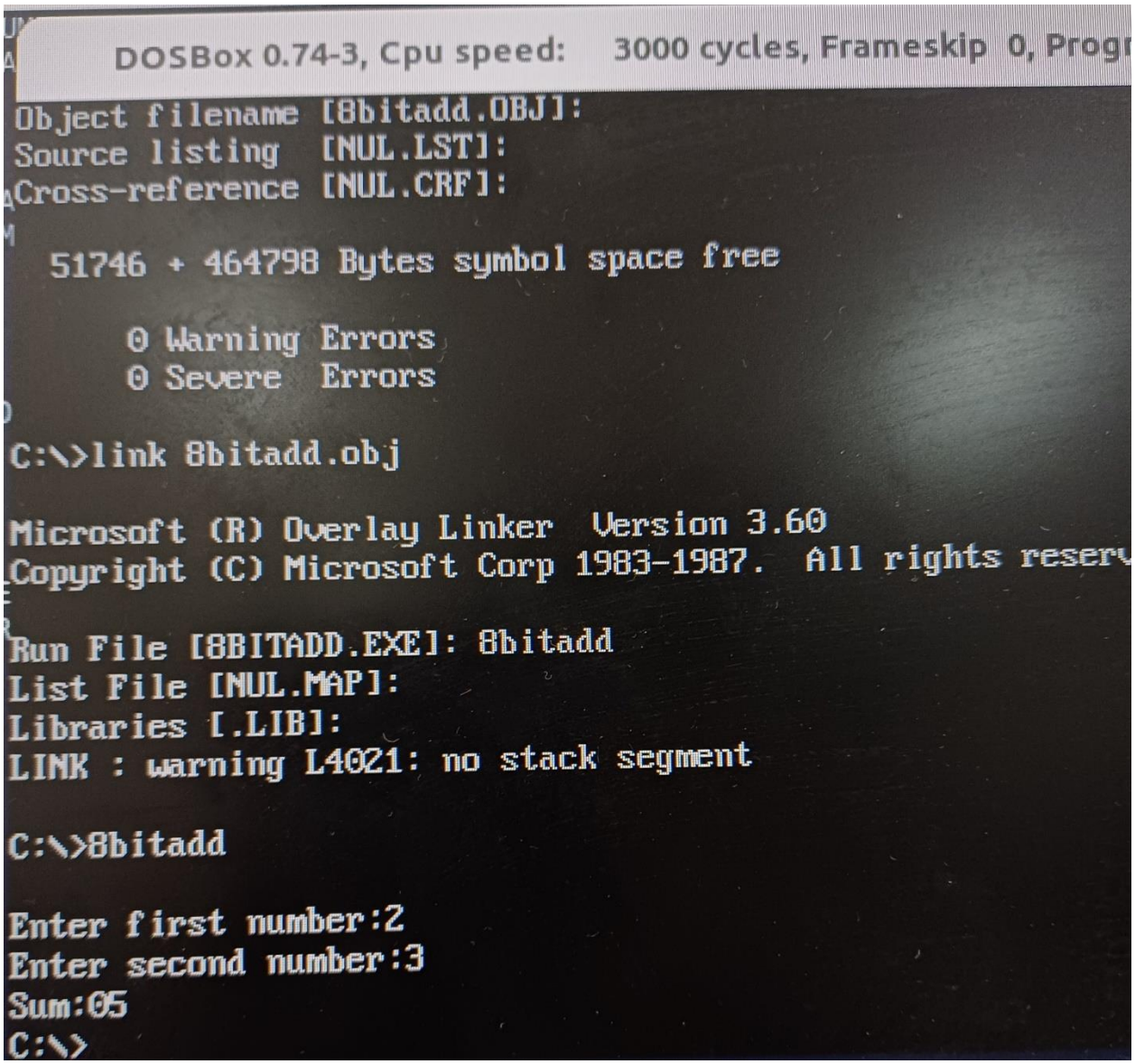
Sample Code and Output

Addition



```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
        M1 DB 10,13,"Enter first number:$"
        M2 DB 10,13,"Enter second number:$"
        M3 DB 10,13,"Sum:$"
DATA ENDS
PRTMSG MACRO MESSAGE
        LEA DX,MESSAGE
        MOV AH,09
        INT 21H
        ENDM
GETDCM MACRO
        MOV AH,01
        INT 21H
        SUB AL,30H
        ENDM
CODE SEGMENT
START:MOV AX,DATA
        MOV DS,AX
        PRTMSG M1
        GETDCM
        MOV BL,AL
        PRTMSG M2
        GETDCM
        ADD AL,BL
        MOV AH,00H
        AAA
        MOV BX,AX
        PRTMSG M3
        MOV DL,BH
        ADD DL,30H
        MOV AH,02
        INT 21H
        MOV DL,BL
        ADD DL,30H
        INT 21H
        MOV AH,4CH
        INT 21H
CODE ENDS
END START
```

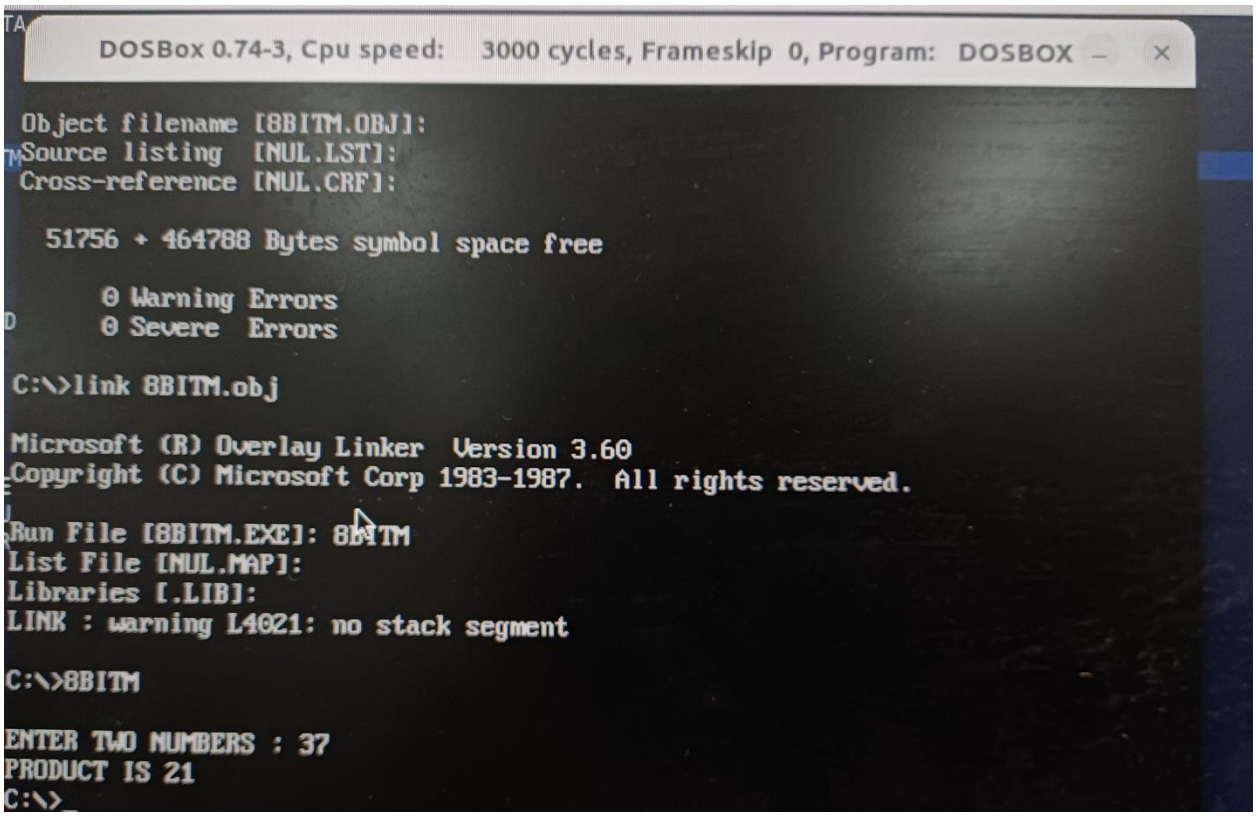
Output


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progr
Object filename [8bitadd.OBJ]:
Source listing [NUL.LST]:
Cross-reference [NUL.CRF]:
51746 + 464798 Bytes symbol space free
0 Warning Errors
0 Severe Errors
C:\>link 8bitadd.obj
Microsoft (R) Overlay Linker Version 3.60
Copyright (C) Microsoft Corp 1983-1987. All rights reserv
Run File [8BITADD.EXE]: 8bitadd
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
C:\>8bitadd
Enter first number:2
Enter second number:3
Sum:05
C:\>

Multiplication

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

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

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

CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
    MOV DS,AX
    PRTMSG M1
    GETDCM
    MOV BL,AL
    GETDCM
    MOV AH,00H
    MUL BL
    AAM
    MOV BX,AX
    PRTMSG M2
    MOV DL,BH
    OR DL,30H
    MOV AH,02H
    INT 21H
    MOV DL,BL
    OR DL,30H
    INT 21H
    MOV AH,4CH
    INT 21H
CODE ENDS
END START
```

Output



Result

Program verified with output

Exp. NO: 8 **JESSIN SUNNY**

Date: 30/10/2024 **ROLL NO: 37**

CHECK NUMBER IS ODD/EVEN USING MASM

Aim: Write a program to check whether the given number is ODD/EVEN Using Masm

Input

A Number

Output

Odd or Even

Algorithm

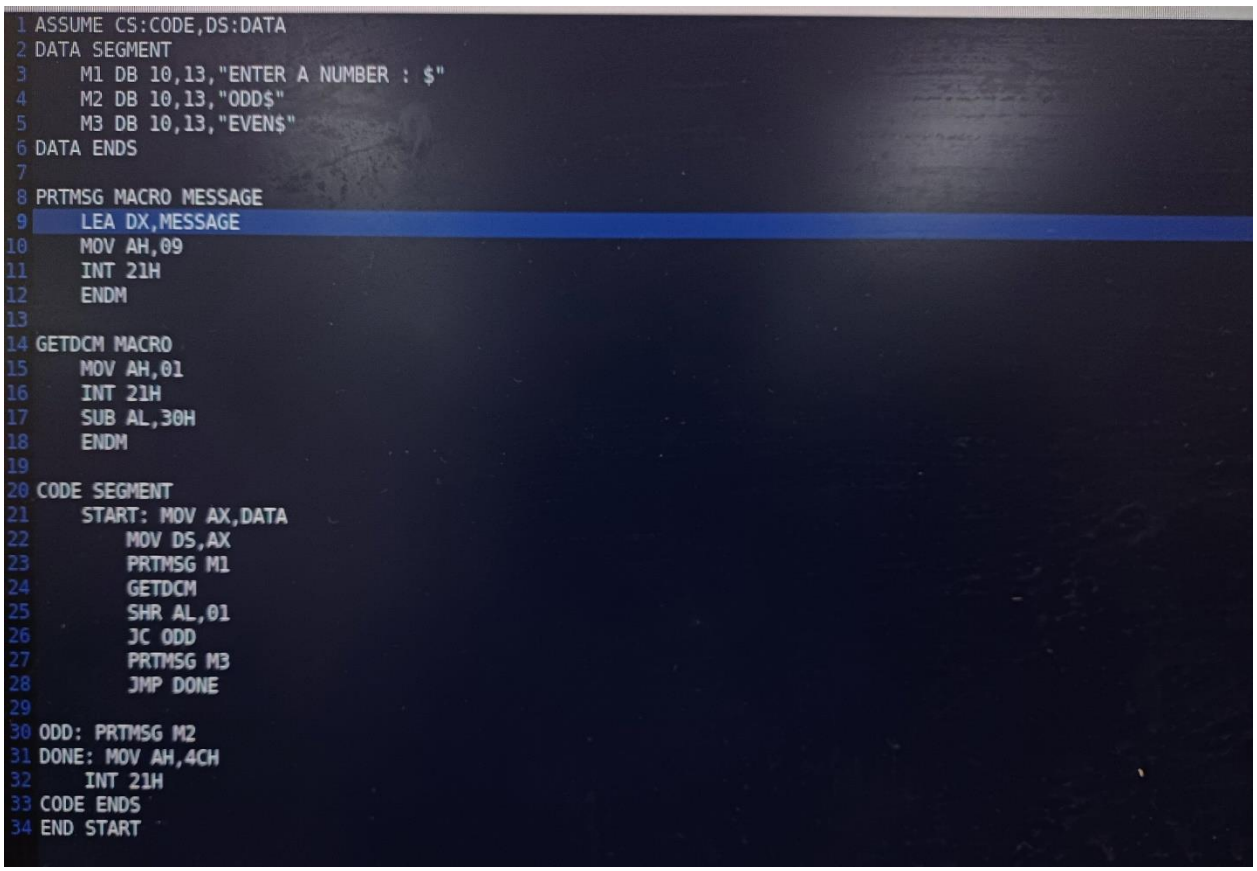
Step 1: Load data segment starting address to DS
Step 2: Get the first number and store it in AL
Step 3: Do Shift right with carry in AL
Step 4: If carry is present jump to ODD label
Step 5: If not present Print "EVEN"
Step 6: Jump to Stop
Step 7: ODD: print "ODD"
Step 8: Stop

Program

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
   M1 DB 10,13,"ENTER A NUMBER : $"
   M2 DB 10,13,"ODD$"
   M3 DB 10,13,"EVEN$"
DATA ENDS

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

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

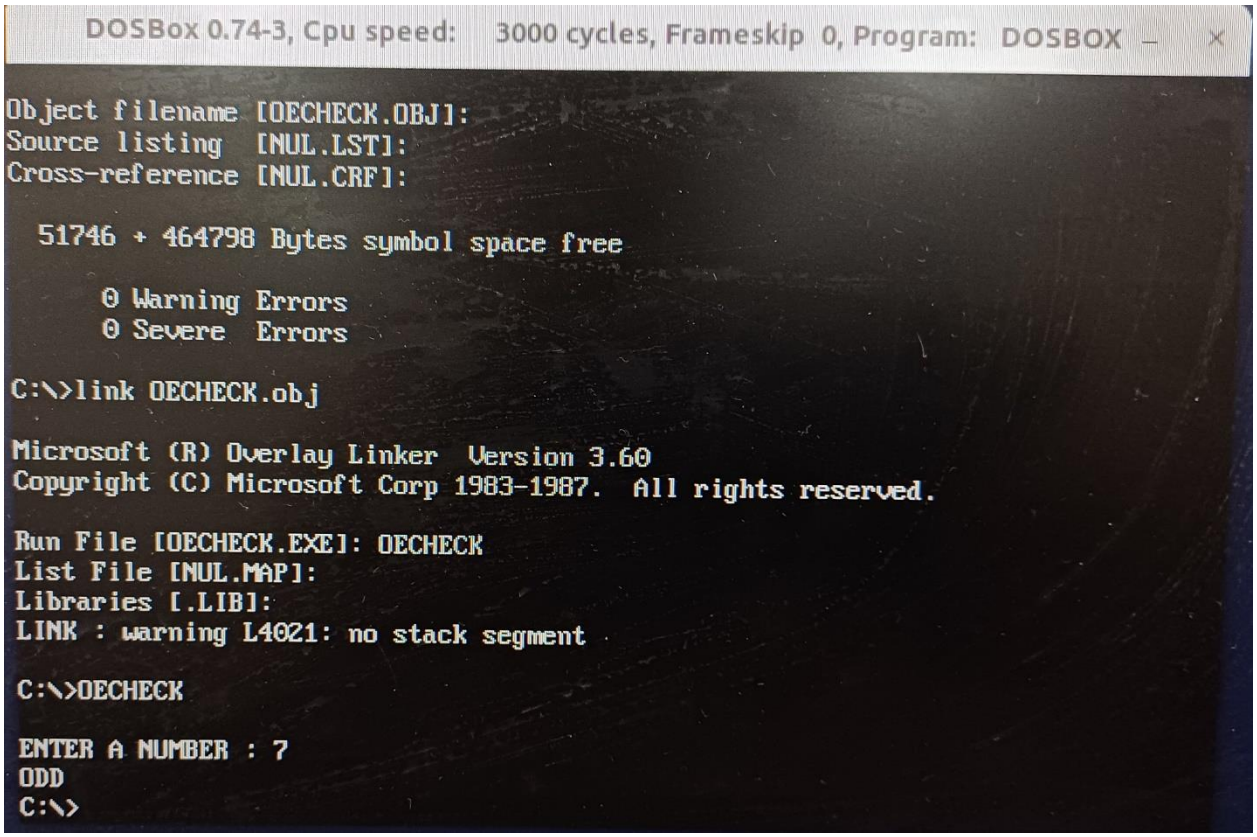
```
   ENDM

CODE SEGMENT
   START: MOV AX,DATA
      MOV DS,AX
      PRTMSG M1
      GETDCM
      SHR AL,01
      JC ODD
      PRTMSG M3
      JMP DONE

ODD: PRTMSG M2
DONE: MOV AH,4CH
   INT 21H
CODE ENDS
END START
```

Sample code and output



```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
    M1 DB 10,13,"ENTER A NUMBER : $"
    M2 DB 10,13,"ODD$"
    M3 DB 10,13,"EVEN$"
DATA ENDS

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

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

CODE SEGMENT
    START: MOV AX,DATA
        MOV DS,AX
        PRTMSG M1
        GETDCM
        SHR AL,01
        JC ODD
        PRTMSG M3
        JMP DONE

ODD: PRTMSG M2
DONE: MOV AH,4CH
    INT 21H
CODE ENDS
END START
```

Output



Result

Program verified with output

Exp. NO: 9 **JESSIN SUNNY**

Date: 30/10/2024 **ROLL NO: 37**

16-BIT ADDITION AND MULTIPLICATION USING MASM

Aim: Write a program to implement 16-bit addition and multiplication using MASM.

Input

Two numbers

Output

Sum and Product.

Addition

Algorithm

Step 1: Load data segment starting address to DS
Step 2: Get the first number and store it in AX
Step 3: Move it to BX
Step 4: Get second number and store it in AX
Step 5: Move it to CX
Step 6: Add CL add BL
Step 7: Copy BL to AL
Step 8: Covert to BCD using AAA
Step 9: Store destination address in SI
Step 10: Move AL to Destination
Step 11: ADD AH to BH
Step 12: Add CH to BH
Step 13: Mov BH to AL
Step 14: Covert to BDC format using AAA
Step 15: Increment SI
Step 16: Copy AL to address pointed by SI
Step 17: Print SI
Step 18: Stop

Program

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

```
   SUM DB 03
DATA ENDS

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

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

PRTDCM MACRO
   MOV DL,[SI]
   ADD DL,30H
   MOV AH,02
   INT 21H
   ENDM

CODE SEGMENT
START : MOV AX,DATA
      MOV DS,AX
      PRTMSG M1
      GETDCM
      MOV BH,AL
      GETDCM
      MOV BL,AL
      PRTMSG M2
      GETDCM
      MOV CH,AL
      GETDCM
      MOV CL,AL
      ADD BL,CL
      MOV AL,BL
      MOV AH,00
      AAA
      LEA SI,SUM
      MOV [SI],AL
      ADD BH,AH
      ADD BH,CH
      MOV AL,BH
      MOV AH,00
      AAA
      INC SI
```

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

Multiplication

Algorithm

Step 1: Load data segment starting address to DS
Step 2: Get the first number and store it in AX
Step 3: Get the second number and store in CX
Step 4: Copy destination address to SI
Step 5: Multiply CL and BL
Step 6: Store it in destination
Step 7: Multiply CL and store it in DX
Step 8: Add data in destination with DL
Step 9: Multiply BL with AL
Step 10: Add the value in AL with DL
Step 11: Store the value in destination
Step 12: Display the value
Step 13: Stop

Program

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
   M1 DB 10,13,"ENTER FIRST NUMBER:$"
   M2 DB 10,13,"ENTER SECOND NUMBER:$"
   M3 DB 10,13,"PRODUCT:$"
   PROD DB 3 DUP(00H)
DATA ENDS

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

GETDCM MACRO
```

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

PRTDCM MACRO
    MOV DL,[SI]
    ADD DL,30H
    MOV AH,02
    INT 21H
    ENDM

CODE SEGMENT
START: MOV AX,DATA
        MOV DS,AX
        PRTMSG M1
        GETDCM
        MOV BH,AL
        GETDCM
        MOV BL,AL
        PRTMSG M2
        GETDCM
        MOV CH,AL
        GETDCM
        MOV CL,AL
        LEA SI,PROD
        MOV AH,00H
        MUL BL
        AAM
        MOV[SI],AL
        INC SI
        MOV [SI],AH
        MOV AH,00H
        MOV AL,BH
        MUL CL
        AAM
        MOV DX,AX
        ADD DL,[SI]
        MOV AH,00H
        MOV AL,CH
        MUL BL
        AAM
        ADD DX,AX
        MOV AL,DL
        MOV AH,00H
        AAM
        ADD DH,AH
```

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

Sample code and output

Addition

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
    M1 DB 10,13,"ENTER FIRST NUMBER:$"
    M2 DB 10,13,"ENTER SECOND NUMBER:$"
    M3 DB 10,13,"SUM:$"
    SUM DB 03
DATA ENDS

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

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

PRTDCM MACRO
    MOV DL,[SI]
    ADD DL,30H
    MOV AH,02
    INT 21H
    ENDM

CODE SEGMENT
START : MOV AX,DATA
        MOV DS,AX
        PRTMSG M1
        GETDCM
        MOV BH,AL
        GETDCM
        MOV BL,AL
        PRTMSG M2
        GETDCM
        MOV CH,AL
        GETDCM
        MOV CL,AL
        ADD BL,CL
        MOV AL,BL
        MOV AH,00
        AAA
        LEA SI,SUM
        MOV [SI],AL
        ADD BH,AH
        ADD BH,CH
        MOV AL,BH
        MOV AH,00
        AAA
        INC SI
        MOV[SI],AL
        INC SI
        MOV [SI],AH
        PRTMSG M3
        PRTDCM
        DEC SI
        PRTDCM
        DEC SI
        PRTDCM
        MOV AH,4CH
        INT 21H
CODE ENDS
END START
```

Output

```
DOSBox 0.74-3, Cpu speed:   3000 cycles, Frameskip  0, Program:  DOSBOX
Object filename [16BITADD.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  51656 + 464888 Bytes symbol space free

       0 Warning Errors
       0 Severe  Errors

C:\>link 16BITADD.obj

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Run File [16BITADD.EXE]: 16BITADD
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment

C:\>16BITADD

ENTER FIRST NUMBER:21
ENTER SECOND NUMBER:16
SUM:037
C:\>
```

Multiplication

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
    M1 DB 10,13,"ENTER FIRST NUMBER:$"
    M2 DB 10,13,"ENTER SECOND NUMBER:$"
    M3 DB 10,13,"PRODUCT:$"
    PROD DB 3 DUP(00H)
DATA ENDS

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

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

PRTDCM MACRO
    MOV DL,[SI]
    ADD DL,30H
    MOV AH,02
    INT 21H
    ENDM

CODE SEGMENT
START: MOV AX,DATA
       MOV DS,AX
       PRTMSG M1
       GETDCM
       MOV BH,AL
       GETDCM
       MOV BL,AL
       PRTMSG M2
       GETDCM
       MOV CH,AL
       GETDCM
       MOV CL,AL
       LEA SI,PROD
       MOV AH,00H
       MUL BL
       AAM
       MOV[SI],AL
       INC SI
       MOV [SI],AH
       MOV AH,00H
       MOV AL,BH
       MUL CL
       AAM
       MOV DX,AX
       ADD DL,[SI]
       MOV AH,00H
       MOV AL,CH
       MUL BL
       AAM
       ADD DX,AX
       MOV AL,DL
       MOV AH,00H
       AAM
       ADD DH,AH
       MOV DL,DH
       MOV DH,00H
       MOV [SI],AL
```

```
66        MOV [SI],AL
67        INC SI
68        MOV AH,00H
69        MOV AL,BH
70        MUL CH
71        AAM
72        ADD DX,AX
73        MOV AL,DL
74        MOV AH,00H
75        AAM
76        MOV [SI],AL
77        INC SI
78        ADD DH,AH
79        MOV AL,DH
80        MOV [SI],AL
81        PRTMSG M3
82        PRTDCM
83        DEC SI
84        PRTDCM
85        DEC SI
86        PRTDCM
87        DEC SI
88        PRTDCM
89        MOV AH,4CH
90        INT 21H
91 CODE ENDS
92 END START
```

Output

```
DOSBox 0.74-3, Cpu speed:   3000 cycles, Frameskip 0, Program:  DOSBOX

      0 Warning Errors
      0 Severe  Errors

C:\>link 16bitmul.obj

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Run File [16BITMUL.EXE]: 16bitmul
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment

C:\>16bitmul

ENTER FIRST NUMBER:25
ENTER SECOND NUMBER:25
PRODUCT:0625
C:\>16bitmul

ENTER FIRST NUMBER:21
ENTER SECOND NUMBER:37
PRODUCT:0777
C:\>
```

Result

Program verified with output

Exp. NO: 10 **JESSIN SUNNY**

Date: 30/10/2024 **ROLL NO: 37**

LINEAR SEARCH USING MASM

Aim: Write a program to implement linear search using MASM.

Input

Numbers

Output

Location Of Key

Algorithm

Step 1: Load data segment starting address to DS
Step 2: Copy value to be searched to AL
Step 3: Copy starting address of array to SI
Step 4: Store size of the array to CX
Step 5: UP:
Step 6: Move first element to BL
Step 7: Compare with AL
Step 8: If Zero jump to FO:
Step 9: Else
Step 10: Increment SI
Step 11: Decrement CX
Step 12: If CX not zero Jump to UP:
Step 13: Print NOT FOUND
Step 14: Jump to stop
Step 15: FO:
Step 16: Print FOUND
Step 17: Stop

Program

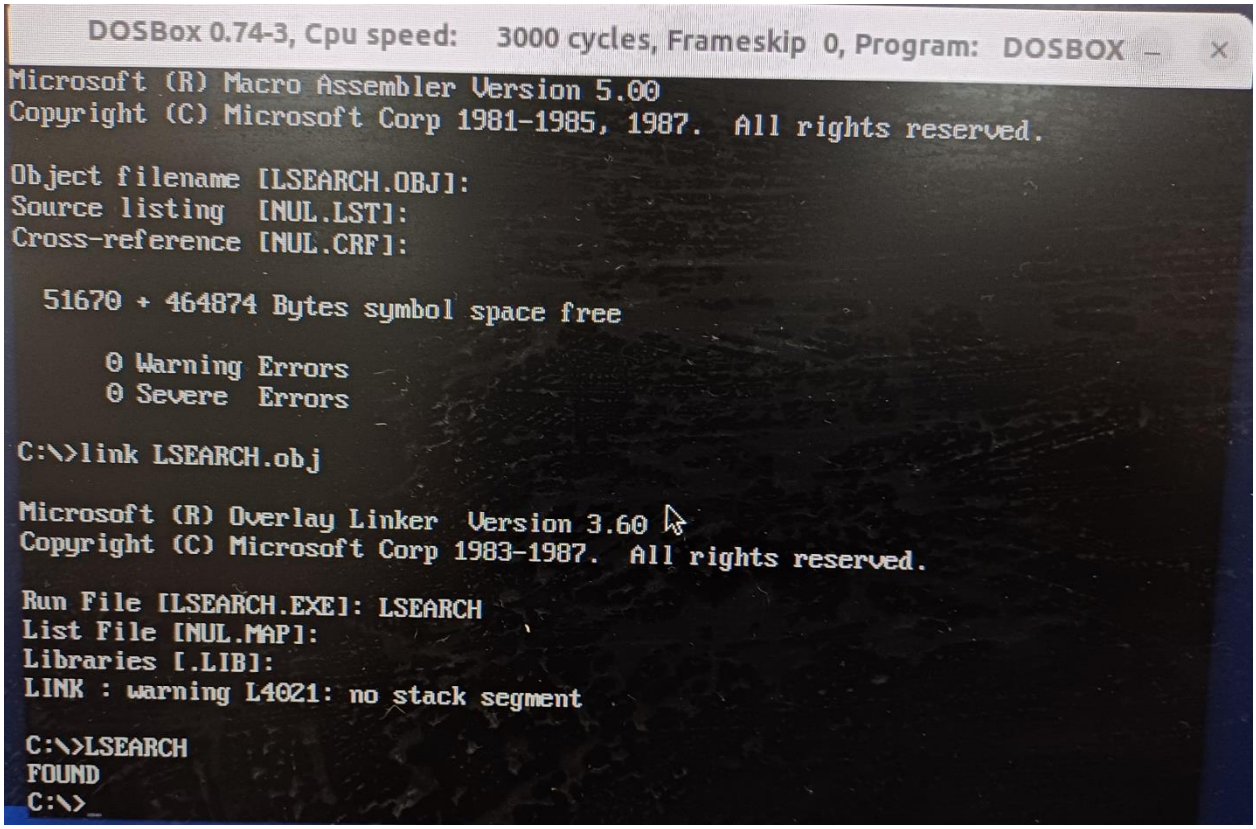
```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
   STRING1 DB 11H,22H,33H,44H,55H
   MSG1 DB "FOUND$"
   MSG2 DB "NOT FOUND$"
   SE DB 22H
DATA ENDS

PRTMSG MACRO MESSAGE
```

```
    MOV AH,09H
    LEA DX,MESSAGE
    INT 21H
    INT 3
    ENDM

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

    UP:
    MOV BL,[SI]
    CMP AL,BL
    JZ FO
    INC SI
    DEC CX
    JNZ UP
    PRTMSG MSG2
    JMP END1

    FO:
    PRTMSG MSG1
    END1: MOV AH,4CH
    INT 21H
CODE ENDS
END START
```

Sample code and output

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
    STRING1 DB 11H,22H,33H,44H,55H
    MSG1 DB "FOUND$"
    MSG2 DB "NOT FOUND$"
    SE DB 22H
DATA ENDS

PRTMSG MACRO MESSAGE
    MOV AH,09H
    LEA DX,MESSAGE
    INT 21H
    INT 3
    ENDM

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

    UP:
    MOV BL,[SI]
    CMP AL,BL
    JZ FO
    INC SI
    DEC CX
    JNZ UP
    PRTMSG MSG2
    JMP END1

    FO:
    PRTMSG MSG1
    END1: MOV AH,4CH
    INT 21H
CODE ENDS
END START
```

Output



Result

Program verified with output

Exp. NO: 11 **JESSIN SUNNY**

Date: 30/10/2024 **ROLL NO: 37**

STRING MANIPULATION USING MASM

Aim: To find the number of vowels, consonants and digits in a string

Input

String

Output

Count

Algorithm

Step 1) Load data segment starting address to DS
Step 2) Load extra segment starting address to ES
Step 3) Copy starting address of the string to SI
Step 4) Move maxlength to CL
Step 5) GETC: call interrupt 21
Step 6) Compare AL with DELIM
Step 7) Jump to ENDET: if equal
Step 8) Increament BL
Step 9) Move AL to Destination
Step 10)Increment SI
Step 11) Loop GETC
Step 12)ENDGET: CLD
Step 13) Copy starting address of string to SI
Step 14) Move content of SI to AX
Step 15) Increment SI
Step 16) Copy the starting address of vowels to DI
Step 17) Repeat when not zero SCASB:
Step 18) Jump on not equal CHKC
Step 19) Increment VCNT
Step 20) Jump to ENDC
Step 21) Display number of vowels, consonants, digits in the string

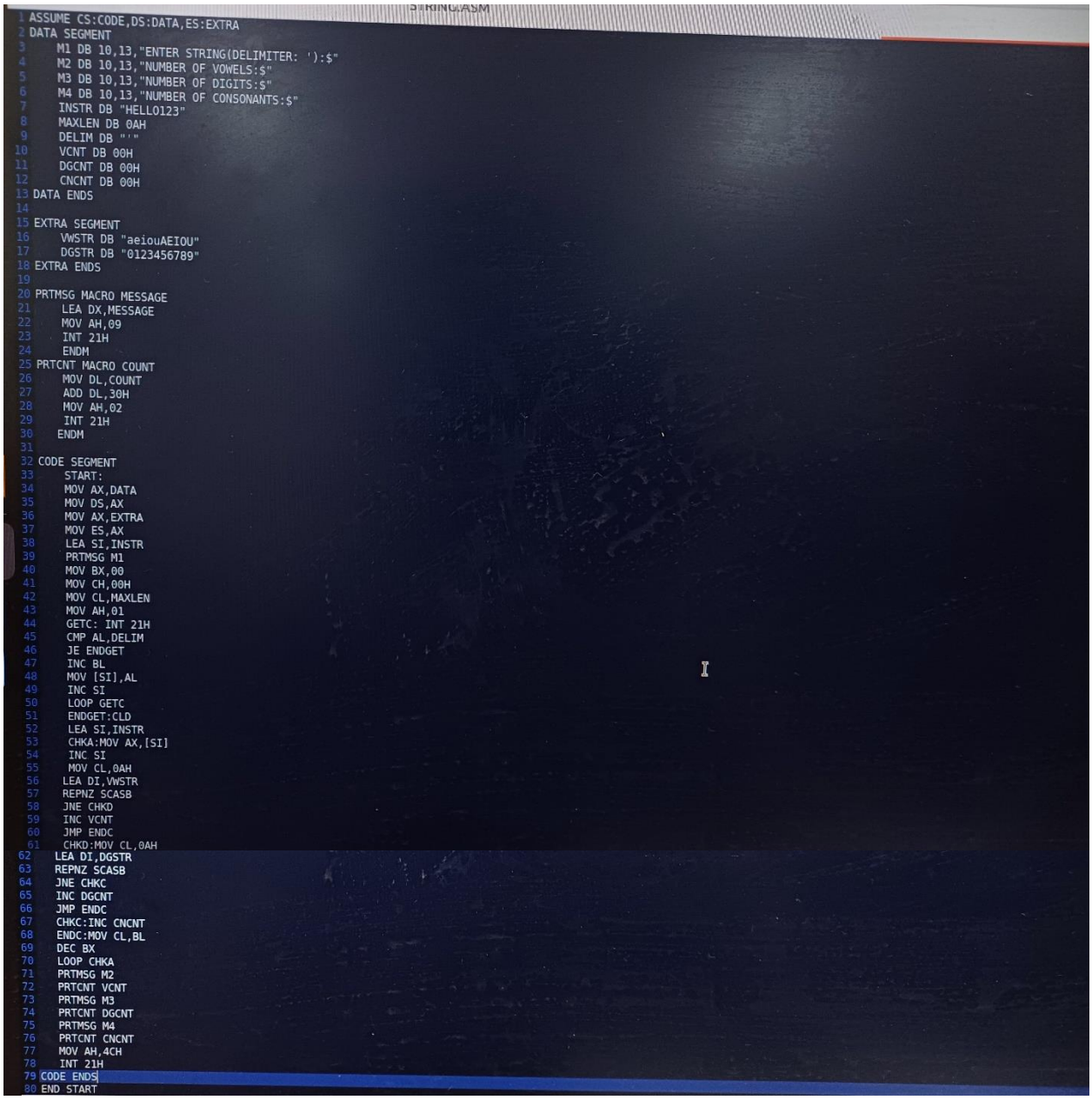
Program

```
ASSUME CS:CODE,DS:DATA,ES:EXTRA
DATA SEGMENT
   M1 DB 10,13,"ENTER STRING(DELIMITER: '):$"
   M2 DB 10,13,"NUMBER OF VOWELS:$"
   M3 DB 10,13,"NUMBER OF DIGITS:$"
   M4 DB 10,13,"NUMBER OF CONSONANTS:$"
   INSTR DB "HELLO123"
```

```
   MAXLEN DB 0AH
   DELIM DB "'"
   VCNT DB 00H
   DGCNT DB 00H
   CNCNT DB 00H
DATA ENDS

EXTRA SEGMENT
   VWSTR DB "aeiouAEIOU"
   DGSTR DB "0123456789"
EXTRA ENDS

PRTMSG MACRO MESSAGE
   LEA DX,MESSAGE
   MOV AH,09
   INT 21H
   ENDM
PRTCNT MACRO COUNT
   MOV DL,COUNT
   ADD DL,30H
   MOV AH,02
   INT 21H
  ENDM

CODE SEGMENT
   START:
   MOV AX,DATA
   MOV DS,AX
   MOV AX,EXTRA
   MOV ES,AX
   LEA SI,INSTR
   PRTMSG M1
   MOV BX,00
   MOV CH,00H
   MOV CL,MAXLEN
   MOV AH,01
   GETC: INT 21H
   CMP AL,DELIM
   JE ENDGET
   INC BL
   MOV [SI],AL
   INC SI
   LOOP GETC
   ENDGET:CLD
   LEA SI,INSTR
   CHKA:MOV AX,[SI]
   INC SI
```

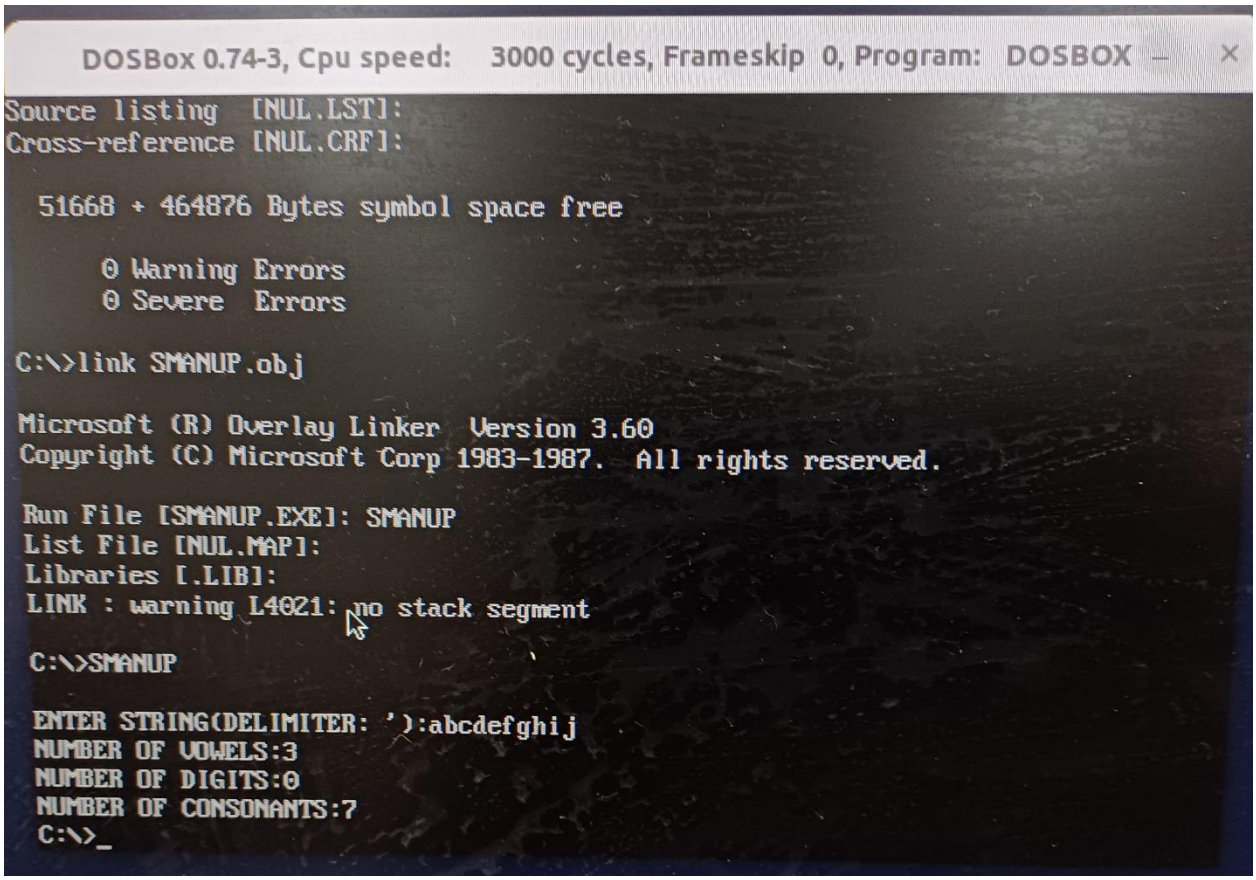
```
    MOV CL,0AH
   LEA DI,VWSTR
   REPNZ SCASB
   JNE CHKD
   INC VCNT
   JMP ENDC
   CHKD:MOV CL,0AH
   LEA DI,DGSTR
   REPNZ SCASB
   JNE CHKC
   INC DGCNT
   JMP ENDC
   CHKC:INC CNCNT
   ENDC:MOV CL,BL
   DEC BX
   LOOP CHKA
   PRTMSG M2
   PRTCNT VCNT
   PRTMSG M3
   PRTCNT DGCNT
   PRTMSG M4
   PRTCNT CNCNT
   MOV AH,4CH
   INT 21H
CODE ENDS
END START
```

Sample code and output



```
ASSUME CS:CODE,DS:DATA,ES:EXTRA
DATA SEGMENT
    M1 DB 10,13,"ENTER STRING(DELIMITER: '):$"
    M2 DB 10,13,"NUMBER OF VOWELS:$"
    M3 DB 10,13,"NUMBER OF DIGITS:$"
    M4 DB 10,13,"NUMBER OF CONSONANTS:$"
    INSTR DB "HELLO123"
    MAXLEN DB 0AH
    DELIM DB "'"
    VCNT DB 00H
    DGCNT DB 00H
    CNCNT DB 00H
DATA ENDS

EXTRA SEGMENT
    VWSTR DB "aeiouAEIOU"
    DGSTR DB "0123456789"
EXTRA ENDS

PRTMSG MACRO MESSAGE
    LEA DX,MESSAGE
    MOV AH,09
    INT 21H
    ENDM
PRTCNT MACRO COUNT
    MOV DL,COUNT
    ADD DL,30H
    MOV AH,02
    INT 21H
   ENDM

CODE SEGMENT
    START:
    MOV AX,DATA
    MOV DS,AX
    MOV AX,EXTRA
    MOV ES,AX
    LEA SI,INSTR
    PRTMSG M1
    MOV BX,00
    MOV CH,00H
    MOV CL,MAXLEN
    MOV AH,01
    GETC: INT 21H
    CMP AL,DELIM
    JE ENDGET
    INC BL
    MOV [SI],AL
    INC SI
    LOOP GETC
    ENDGET:CLD
    LEA SI,INSTR
    CHKA:MOV AX,[SI]
    INC SI
    MOV CL,0AH
   LEA DI,VWSTR
   REPNZ SCASB
   JNE CHKD
   INC VCNT
   JMP ENDC
   CHKD:MOV CL,0AH
  LEA DI,DGSTR
  REPNZ SCASB
  JNE CHKC
  INC DGCNT
  JMP ENDC
  CHKC:INC CNCNT
  ENDC:MOV CL,BL
  DEC BX
  LOOP CHKA
  PRTMSG M2
  PRTCNT VCNT
  PRTMSG M3
  PRTCNT DGCNT
  PRTMSG M4
  PRTCNT CNCNT
  MOV AH,4CH
  INT 21H
CODE ENDS
END START
```

Output



Result

Program verified with output