**Syllabus**

**CSL331 System Software and Microprocesors Lab**

|  |
| --- |

**Objectives**

* + To familiarize students with System Software , Operating System Programs and 8086 Microprcoessor and 8085 Microcontroller Kit

|  | |
| --- | --- |
| **List of Exercises/Experiments** |
| System Software /Operating Systems   * 1. CPU SCHEDULING   2. BANKER’S ALGORITHM FOR DEADLOCK AVOIDANCE   3. DISK SCHEDULING ALGORITHMS   4. PAGE REPLACEMENT ALGORITHMS   5. FILE ORGANISATION TECHNIQUES   6. PAGING TECHNIQUES OF MEMORY MANAGEMENT   7. FILE ALLOCATION STRATEGIES   8. PASS ONE OF TWO PASS ASSEMBLER   9. PASS TWO OF TWO PASS ASSEMBLER   10. ABSOLUTE LOADER   11. RELOCATING LOADER   12. TWO PASS MACRO PROCESSOR   13. SINGLE PASS MACRO PROCESSOR   MASM Programs   * 1. BASIC ARITHMETIC OPERATIONS   2. STRING DISPLAY   3. STRING CONCATENATENATION   4. SORTING   5. SEARCHING   8086 TRAINER KIT Programs   * 1. BASIC ARITHEMATIC AND LOGICAL OPERATIONS   2. SORTING-ASCENDING   3. CODE CONVERSIONS   8051 TRAINER KIT Prgrams   * 1. BASIC ARITHEMATIC AND LOGICAL OPERATIONS   2. CODE CONVERSIONS   **INDEX**   | Sl. No | List Of EXPERIMENTS | | Page No. | | | --- | --- | --- | --- | --- | | UNIVERSITY PRESCRIBED EXPERIMENTS | | | | | |  | **SYSTEM SOFTWARE / OPERATING SYSTEM PROGRAMS** | |  | | |  | CPU SCHEDULING ALGORITHMS | | 7 | | |  | BANKER’S ALGORITHM FOR DEADLOCK AVOIDANCE | | 17 | | |  | DISK SCHEDULING ALGORITHMS | | 23 | | |  | PAGE REPLACEMENT ALGORITHMS | | 30 | | |  | FILE ORGANISATION TECHNIQUES | | 38 | | |  | PAGING TECHNIQUES OF MEMORY MANAGEMENT | | 48 | | |  | FILE ALLOCATION STRATEGIES | | 51 | | |  | PASS ONE OF TWO PASS ASSEMBLER | | 57 | | |  | PASS TWO OF TWO PASS ASSEMBLER | | 62 | | |  | ABSOLUTE LOADER | | 67 | | |  | RELOCATING LOADER | | 70 | | |  | TWO PASS MACRO PROCESSOR | | 75 | | |  | SINGLE PASS MACRO PROCESSOR | | 78 | | |  | VIVA QUESTIONS AND ANSWERS | | 81 | | |  | **MASM PROGRAMS** | |  | | |  | BASIC ARITHMETIC OPERATIONS | | 90 | | |  | | STRING DISPLAY | | 99 | | |  | | STRING CONCATENATENATION | | 101 | | |  | | SORTING | | 103 | | |  | | SEARCHING | | 106 | | |  | | **8086 TRAINER KIT PROGRAMS** | |  | | |  | | BASIC ARITHEMATIC OPERATIONS | | 109 | | |  | | SORTING-ASCENDING | | 113 | | |  | | CODE CONVERSIONS | | 115 | | |  | | **8051 TRAINER KIT PROGRAMS** | |  | | |  | | BASIC ARITHEMATIC AND LOGICAL OPERATIONS | | 119 | | |  | | CODE CONVERSIONS | | 124 | | |  | | VIVA QUESTIONS AND ANSWERS | | 126 | | |
|  |

**Pgm.No.1**

**CPU SCHEDULING**

**AIM**

Similate the following non pre-emptive CPU scheduling algorithms to find turnaround time and waiting tme.

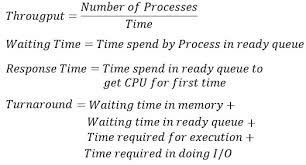
a). FCFS

b). SJF

c). Priority

d). Round Robin (Pre-emptive)

ALGORITHM



**PROGRAM**

CPU SCHEDULING - FIRST COME FIRST SERVE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <stdio.h>

#include <conio.h>

int main()

{

int bt[10],wt[10],tt[10],n,i;

float twt=0,ttt=0;

printf("Enter the number of process:");

scanf("%d",&n);

printf("Enter burst time of each process\n");

for(i=0;i<n;i++)

{

printf("process %d:",i+1);

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

}

for(i=0;i<n;i++)

{

if(i==0)

{

wt[i]=0;

}

else {

wt[i]=bt[i-1]+wt[i-1];

}

twt=twt+wt[i];

}

for(i=0;i<n;i++)

{

tt[i]=wt[i]+bt[i];

ttt=ttt+tt[i];

}

printf("\n\tProcess \tBurst time \tWaiting Time \tTurnaroung Time");

for(i=0;i<n;i++)

{

printf("\n\t%d \t\t%d \t\t%d \t\t%d\n",i+1,bt[i],wt[i],tt[i]);

}

printf("Total Waiting Time=%.2f\n",twt);

printf("Total Turnaround Time=%.2f\n",ttt);

printf("Average Waiting Time=%.2f\n",twt/n);

printf("Average Turnaround Time=%.2f\n",ttt/n);

getch();

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OUTPUT

Enter the number of process:4

Enter burst time of each process

process 1:5

process 2:4

process 3:3

process 4:2

Process Burst time Waiting Time Turnaroung Time

1 5 0 5

2 4 5 9

3 3 9 12

4 2 12 14

Total Waiting Time=26.00

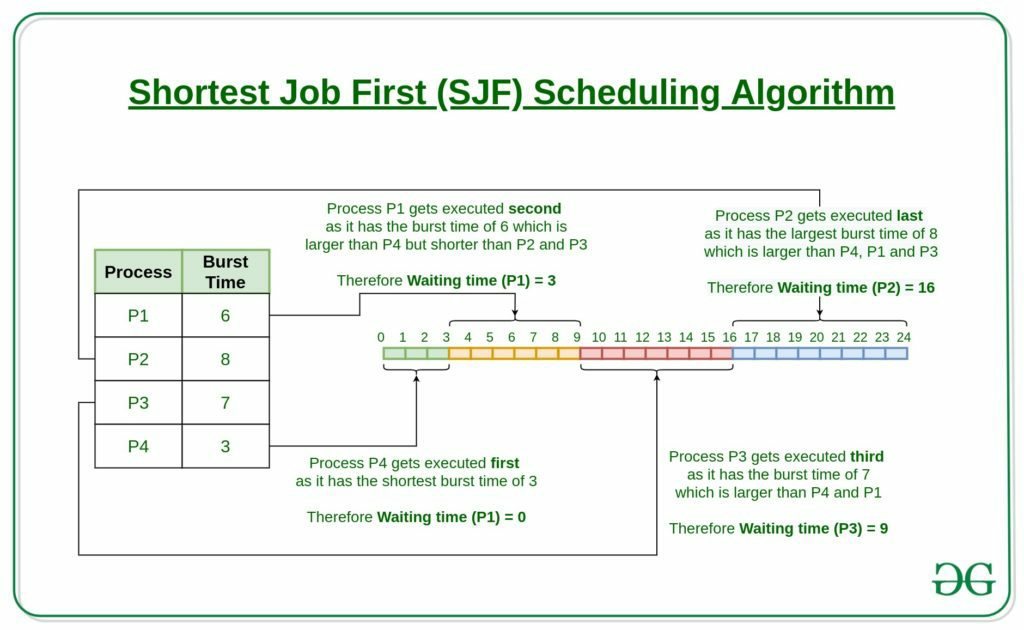
Total Turnaround Time=40.00

Average Waiting Time=6.50

Average Turnaround Time=10.00

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ALGORITHM



1.  As the name suggests, the shortest job first algorithm is an algorithm which executes the process whose burst time is least and has arrived before the current time. Therefore, in order to find the process which needs to be executed, sort all the processes from the given set of processes according to their arrival time. This ensures that the process with the shortest burst time which has arrived first is executed first.
2. Instead of finding the minimum burst time process among all the arrived processes by iterating the   
   whole struct array, the [range minimum](https://www.geeksforgeeks.org/segment-tree-set-1-range-minimum-query/) of the burst time of all the arrived processes upto the current time is calculated using [segment tree](https://www.geeksforgeeks.org/segment-tree-set-1-sum-of-given-range/).
3. After selecting a process which needs to be executed, the **completion time**, **turn around time** and **waiting time** is calculated by using arrival time and burst time of the process. The formulae to calculate the respective times are:
   * **Completion Time:** Time at which process completes its execution.

Completion Time = Start Time + Burst Time

* **Turn Around Time:** Time Difference between completion time and arrival time.

Turn Around Time = Completion Time – Arrival Time

* **Waiting Time(W.T):** Time Difference between turn around time and burst time.

Waiting Time = Turn Around Time – Burst Time

1. After calculating, the respective times are updated in the array and the burst time of the executed process is set to infinity in the segment tree base array so that it is not considered as the minimum burst time in the further queries.

**PROGRAM**

CPU SCHEDULING - SHORTEST JOB FIRST

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <stdio.h>

#include <stdio.h>

int main()

{

int bt[10],wt[10],tt[10],n,i,j,p[10],temp;

float twt=0,ttt=0;

printf("Enter the number of process:");

scanf("%d",&n);

printf("Enter burst time of the process:\n");

for(i=0;i<n;i++)

{

printf("process %d:",i+1);

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

p[i]=i+1;

}

for(i=0;i<n;i++)

{

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

{

if(bt[i]>bt[j])

{

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

temp=p[i];

p[i]=p[j];

p[j]=temp;

}

}

}

for(i=0;i<n;i++)

{

if(i==0)

{

wt[i]=0;

}

else {

wt[i]=bt[i-1]+wt[i-1];

}

twt=twt+wt[i];

}

for(i=0;i<n;i++)

{

tt[i]=wt[i]+bt[i];

ttt=ttt+tt[i];

}

printf("\n\tProcess \tBurst time \tWaiting Time \tTurnaroung Time");

for(i=0;i<n;i++)

{

printf("\n\t%d \t\t%d \t\t%d \t\t%d\n",p[i],bt[i],wt[i],tt[i]);

}

printf("Total Waiting Time=%.2f\n",twt);

printf("Total Turnaround Time=%.2f\n",ttt);

printf("Average Waiting Time=%.2f\n",twt/n);

printf("Average Turnaround Time=%.2f\n",ttt/n);

getch();

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OUTPUT

Enter the number of process:4

Enter burst time of the process:

process 1:5

process 2:4

process 3:3

process 4:2

Process Burst time Waiting Time Turnaroung Time

4 2 0 2

3 3 2 5

2 4 5 9

1 5 9 14

Total Waiting Time=16.00

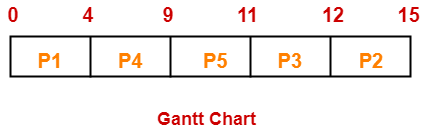
Total Turnaround Time=30.00

Average Waiting Time=4.00

Average Turnaround Time=7.50

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ALGORITHM



* Turn Around time = Exit time – Arrival time
* Waiting time = Turn Around time – Burst time

**PROGRAM**

CPU SCHEDULING - PRIORITY SCHEDULING

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <stdio.h>

#include <conio.h>

int main()

{

int bt[10],wt[10],tt[10],pr[10],n,i,j,p[10],temp;

float twt=0,ttt=0;

printf("Enter the number of process:");

scanf("%d",&n);

printf("Enter burst time and priority of the process:\n");

for(i=0;i<n;i++)

{

printf("process %d:",i+1);

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

p[i]=i+1;

}

for(i=0;i<n;i++)

{

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

{

if(pr[i]>pr[j])

{

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

temp=p[i];

p[i]=p[j];

p[j]=temp;

temp=pr[i];

pr[i]=pr[j];

pr[j]=temp;

}

}

}

for(i=0;i<n;i++)

{

if(i==0)

{

wt[i]=0;

}

else {

wt[i]=bt[i-1]+wt[i-1];

}

twt=twt+wt[i];

}

for(i=0;i<n;i++)

{

tt[i]=wt[i]+bt[i];

ttt=ttt+tt[i];

}

printf("\n\tProcess \tBurst time \tWaiting Time \tTurnaroung Time");

for(i=0;i<n;i++)

{

printf("\n\t%d \t\t%d \t\t%d \t\t%d\n",p[i],bt[i],wt[i],tt[i]);

}

printf("Total Waiting Time=%.2f\n",twt);

printf("Total Turnaround Time=%.2f\n",ttt);

printf("Average Waiting Time=%.2f\n",twt/n);

printf("Average Turnaround Time=%.2f\n",ttt/n);

getch();

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OUTPUT

Enter the number of process:4

Enter burst time and priority of the process:

process 1:5 4

process 2:4 3

process 3:3 2

process 4:2 1

Process Burst time Waiting Time Turnaroung Time

4 2 0 2

3 3 2 5

2 4 5 9

1 5 9 14

Total Waiting Time=16.00

Total Turnaround Time=30.00

Average Waiting Time=4.00

Average Turnaround Time=7.50

CPU SCHEDULING - ROUND ROBIN

**ALGORITHM**

1- Create an array **rem\_bt[]** to keep track of remaining

burst time of processes. This array is initially a

copy of bt[] (burst times array)

2- Create another array **wt[]** to store waiting times

of processes. Initialize this array as 0.

3- Initialize time : t = 0

4- Keep traversing the all processes while all processes

are not done. Do following for i'th process if it is

not done yet.

a- If rem\_bt[i] > quantum

(i) t = t + quantum

(ii) rem\_bt[i] -= quantum;

c- Else // Last cycle for this process

(i) t = t + rem\_bt[i];

(ii) wt[i] = t - bt[i]

(ii) rem\_bt[i] = 0; // This process is over

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <stdio.h>

#include <conio.h>

int main()

{

int bt[10],wt[10],tat[10],temp[10],sum=0,count=0,q,tn,n,i;

float twt=0,ttt=0;

printf("Enter the number of process:");

scanf("%d",&n);

tn=n;

printf("Enter burst time of the process:\n");

for(i=0;i<n;i++)

{

printf("process %d:",i+1);

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

temp[i]=bt[i];

}

printf("Enter the time slice or time quantum:");

scanf("%d",&q);

printf("\nProcess No\t\tBurst Time \tWaiting Time \tTurnaround Time");

for(i=0;tn!=0;)

{

if(temp[i]<=q && temp[i]>0)

{

sum=sum + temp[i];

temp[i]=0;

count=1;

}

else if(temp[i]>q)

{

temp[i]=temp[i] - q;

sum=sum + q;

}

if(temp[i]==0 && count==1)

{

tn--;

wt[i]=sum-bt[i];

tat[i]=sum;

printf("\nProcess %d\t\t%d\t\t%d\t\t%d",i+1,bt[i],sum-bt[i],sum);

ttt=ttt + tat[i];

twt=twt + wt[i];

count=0;

}

if(i==(n-1))

{

i=0;

}

else

{

i++;

}

}

printf("\n\nTotal Waiting Time=%.2f\n",twt);

printf("\nTotal Turnaround Time=%.2f\n",ttt);

printf("\nAverage Waiting Time=%.2f\n",twt/n);

printf("\nAverage Turnaround Time=%.2f\n",ttt/n);

getch();

return 0;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OUTPUT

Enter the number of process:4

Enter burst time of the process:

process 1:5

process 2:4

process 3:3

process 4:2

Enter the time slice or time quantum:2

Process No Burst Time Waiting Time Turnaround Time

Process 4 2 6 8

Process 2 4 8 12

Process 3 3 10 13

Process 1 5 9 14

Total Waiting Time=33.00

Total Turnaround Time=47.00

Average Waiting Time=8.25

Average Turnaround Time=11.75

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Pgm.No.2**

BANKER’S ALGORITHM FOR DEADLOCK AVOIDANCE

AIM

Implement banker’s algorithm for deadlock avoidance

ALGORITHM

### Banker’s algorithm comprises of two algorithms:

1. Safety algorithm
2. Resource request algorithm

## Safety Algorithm

A safety algorithm is an algorithm used to find whether or not a system is in its safe state. The algorithm is as follows:

1. Let Work and Finish be vectors of length **m** and **n**, respectively. Initially,
2. Find an index **i** such that both

If there is no such i present, then proceed to step 4.

1. Go to step 2.

When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

1. If Finish[i] == true for all i, then the system is in a safe state.

That means if all processes are finished, then the system is in safe state.

This algorithm may require an order of **mxn² operations** in order to determine whether a state is safe or not.

It means, we need to find an unfinished process whose needs can be satisfied by the available resources. If no such process exists, just go to step 4.

## Resource Request Algorithm

Now the next algorithm is a resource-request algorithm and it is mainly used to determine whether requests can be safely granted or not.

Let Requesti be the request vector for the process Pi. If Requesti[j]==k, then process Pi wants k instance of Resource type Rj.When a request for resources is made by the process Pi, the following are the actions that will be taken:

1. If Requesti <= Needi, then go to step 2;else raise an error condition, since the process has exceeded its maximum claim.

2.If Requesti <= Availablei then go to step 3; else Pi must have to wait as resources are not available.

3.Now we will assume that resources are assigned to process Pi and thus perform the following steps:

Available= Available-Requesti;

Allocationi=Allocationi +Requesti;

Needi =Needi - Requesti;

If the resulting resource allocation state comes out to be safe, then the transaction is completed and, process Pi is allocated its resources. But in this case, if the new state is unsafe, then Pi waits for Requesti, and the old resource-allocation state is restored.

PROGRAM

#include<stdio.h>

struct pro{

int all[10],max[10],need[10];

int flag;

};

int i,j,pno,r,nr,id,k=0,safe=0,exec,count=0,wait=0,max\_err=0;

struct pro p[10];

int aval[10],seq[10];

void safeState()

{

while(count!=pno){

safe = 0;

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

if(p[i].flag){

exec = r;

for(j=0;j<r;j++)

{

if(p[i].need[j]>aval[j]){

exec =0;

}

}

if(exec == r){

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

aval[j]+=p[i].all[j];

}

p[i].flag = 0;

seq[k++] = i;

safe = 1;

count++;

}

}

}

if(!safe)

{

printf("System is in Unsafe State\n");

break;

}

}

if(safe){

printf("\n\nSystem is in safestate \n");

printf("Safe State Sequence \n");

for(i=0;i<k;i++)

printf("P[%d] ",seq[i]);

printf("\n\n");

}

}

void reqRes(){

printf("\nRequest for new Resourses");

printf("\nProcess id ? ");

scanf("%d",&id);

printf("Enter new Request details ");

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

scanf("%d",&nr);

if( nr <= p[id].need[i])

{

if( nr <= aval[i]){

aval[i] -= nr;

p[id].all[i] += nr;

p[id].need[i] -= nr;

}

else

wait = 1;

}

else

max\_err = 1;

}

if(!max\_err && !wait)

safeState();

else if(max\_err){

printf("\nProcess has exceeded its maximum usage \n");

}

else{

printf("\nProcess need to wait\n");

}

}

void main()

{

printf("Enter no of process ");

scanf("%d",&pno);

printf("Enter no. of resourses ");

scanf("%d",&r);

printf("Enter Available Resourse of each type ");

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

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

}

printf("\n\n---Resourse Details---");

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

printf("\nResourses for process %d\n",i);

printf("\nAllocation Matrix\n");

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

scanf("%d",&p[i].all[j]);

}

printf("Maximum Resourse Request \n");

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

scanf("%d",&p[i].max[j]);

}

p[i].flag = 1;

}

// Calcualting need

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

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

p[i].need[j] = p[i].max[j] - p[i].all[j];

}

}

//Print Current Details

printf("\nProcess Details\n");

printf("Pid\t\tAllocattion\t\tMax\t\tNeed\n");

for(i=0;i<pno;i++)

{

printf("%d\t\t",i);

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

printf("%d ",p[i].all[j]);

}

printf("\t\t");

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

printf("%d ",p[i].max[j]);

}

printf("\t\t");

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

printf("%d ",p[i].need[j]);

}

printf("\n");

}

//Determine Current State in Safe State

safeState();

int ch=1;

do{

printf("Request new resourse ?[0/1] :");

scanf("%d",&ch);

if(ch)

reqRes();

}while(ch!=0);

//end:printf("\n");

}

**OUTPUT**

Enter no of process 5

Enter no. of resourses 3

Enter Available Resourse of each type 3

3

2

---Resourse Details---

Resourses for process 0

Allocation Matrix

0 1 0

Maximum Resourse Request

7 5 3

Resourses for process 1

Allocation Matrix

3 0 2

Maximum Resourse Request

3 2 2

Resourses for process 2

Allocation Matrix

3 0 2

Maximum Resourse Request

9 0 2

Resourses for process 3

Allocation Matrix

2 1 1

Maximum Resourse Request

2 2 2

Resourses for process 4

Allocation Matrix

0 0 2

Maximum Resource Request

4 3 3

Process Details

Pid Allocation Max Need

0 0 1 0 7 5 3 7 4 3

1 3 0 2 3 2 2 0 2 0

2 3 0 2 9 0 2 6 0 0

3 2 1 1 2 2 2 0 1 1

4 0 0 2 4 3 3 4 3 1

System is in safe state

Safe State Sequence

P[1] P[2] P[3] P[4] P[0]

Request new resource ?[0/1] :

**Pgm.No.3**

**DISK SCHEDULING ALGORITHMS**

**AIM**

Simulate the following disk scheduling algorithms

a). FCFS

b). SCAN

c). C-SCAN

**FIRST COME FIRST SERVE (FCFS)**

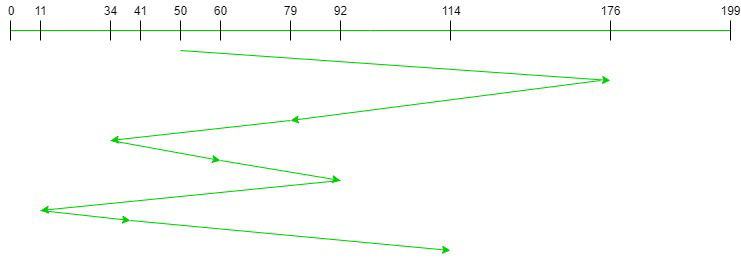
**ALGORITHM**

* **Seek Time:**Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
* **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
* **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
* **Disk Access Time:** Disk Access Time is:

Disk Access Time = Seek Time +

Rotational Latency +

Transfer Time



**FCFS:** FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.Let us understand this with the help of an example.

**PROGRAM**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

DISK SHEDULING ALGORITHM- FIRST COME FIRST SERVE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include<stdio.h>

#include<math.h>

#include<conio.h>

void main()

{

int n,q[10],i;

float total=0;

printf("Enter queue size:");

scanf("%d",&n);

printf("Enter list to be sheduled:");

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

{

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

}

printf("Enter initial head:");

scanf("%d",&q[0]);

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

{

total=total+abs(q[i]-q[i-1]);

}

printf("Seek Count = %.2f",total);

printf("\nAverage Seek Count = %.2f",total/n);

getch();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OUTPUT

------

Enter queue size:5

Enter list to be sheduled:90 179 28 3 150

Enter initial head:50

Seek Count = 452.00

Average Seek Count = 90.40

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DISK SHEDULING ALGORITHM - SCAN

ALGORITHM

## **SCAN Disk Scheduling Algorithm-**

1. As the name suggests, this algorithm scans all the cylinders of the disk back and forth.
2. Head starts from one end of the disk and move towards the other end servicing all the requests in between.
3. After reaching the other end, head reverses its direction and move towards the starting end servicing all the requests in between.
4. The same process repeats.

* SCAN Algorithm is also called as **Elevator Algorithm.**
* This is because its working resembles the working of an elevator.

PROGRAM

#include<stdio.h>

void main()

{

int n,q[10],i,j,s,d,temp,x,y;

float total=0;

printf("Enter disk size:");

scanf("%d",&s);

s--;

printf("Enter queue size:");

scanf("%d",&n);

printf("Enter list to be sheduled:");

for(i=1;i<=n;i++){ scanf("%d",&q[i]); }

printf("Enter initial head:");

scanf("%d",&q[0]);

printf("Enter move direction:");

scanf("%d",&d);

x=q[0];

for(i=0;i<=n;i++)

{

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

{

if(q[j]<q[i])

{

temp=q[i];

q[i]=q[j];

q[j]=temp;

}

}

}

for(i=0;i<=n;i++)

{

if(q[i]==x)

y=i;

}

printf("Disk Sheduling Order\n%d",q[y]);

if(d==1)

{

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

{

printf("->%d",q[i]);

total=total+q[i]-q[i-1];

}

printf("->%d",s);

total=total+s-q[--i];

printf("->%d",q[y-1]);

total=total+s-q[y-1];

for(i=y-2;i>=0;i--){

printf("->%d",q[i]);

total=total+q[i+1]-q[i];

}

}

if(d==0)

{

for(i=y-1;i>=0;i--)

{

printf("->%d",q[i]);

total=total+q[i+1]-q[i];

}

printf("->0");

total=total+q[++i];

printf("->%d",q[y+1]);

total=total+q[y+1];

for(i=y+2;i<=n;i++)

{

total=total+q[i]-q[i-1];

printf("->%d",q[i]);

}

}

printf("\nSeek time is:%.2f",total);

getch();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OUTPUT

------

Enter disk size:200

Enter queue size:8

Enter list to be sheduled:176 79 34 60 92 11 41 114

Enter initial head:50

Enter move direction:0

Disk Sheduling Order

50->41->34->11->0->60->79->92->114->176

Seek time is:226.00

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DISK SHEDULING ALGORITHM - CSCAN

ALGORITHM

### C-SCAN (Circular Elevator) Disk Scheduling Algorithm

The circular SCAN (C-SCAN) scheduling algorithm is a modified version of the SCAN disk scheduling algorithm that deals with the inefficiency of the SCAN algorithm by servicing the requests more uniformly. Like SCAN (Elevator Algorithm) C-SCAN moves the head from one end servicing all the requests to the other end. However, as soon as the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip (see chart below) and starts servicing again once reaches the beginning. This is also known as the “Circular Elevator Algorithm” as it essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

#### **Advantages of C-SCAN (Circular Elevator) Disk Scheduling Algorithm:**

* Works well with moderate to heavy loads.
* It provides better response time and uniform waiting time.

PROGRAM

#include<stdio.h>

void main()

{

int n,q[10],i,j,s,d,temp,x,y;

float total=0;

printf("Enter disk size:");

scanf("%d",&s);

s--;

printf("Enter queue size:");

scanf("%d",&n);

printf("Enter list to be sheduled:");

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

{

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

}

printf("Enter initial head:");

scanf("%d",&q[0]);

x=q[0];

for(i=0;i<=n;i++)

{

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

{

if(q[j]<q[i])

{

temp=q[i];

q[i]=q[j];

q[j]=temp;

}

}

}

for(i=0;i<=n;i++)

{

if(q[i]==x)

y=i;

}

printf("Disk Sheduling Order\n%d",q[y]);

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

{

printf("->%d",q[i]);

total=total+q[i]-q[i-1];

}

printf("->%d",s);

total=total+s-q[--i];

printf("->0->%d",q[0]);

total=total+q[0];

for(i=1;i<y;i++)

{

printf("->%d",q[i]);

total=total+q[i]-q[i-1];

}

printf("\nSeek time is:%.2f",total);

getch();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OUTPUT

------

Enter disk size:200

Enter queue size:8

Enter list to be sheduled:176 79 34 60 92 11 41 114

Enter initial head:50

Disk Sheduling Order

50->60->79->92->114->176->199->0->11->34->41

Seek time is:190.00

**Pgm No . 4**

**PAGE REPLACEMENT ALGORITHMS**

**AIM**

Simulate the following page replacement algorithms

1. FIFO
2. LRU
3. LFU

**ALGORITHM**

**Page Fault:** The page fault takes place when the main program accesses the memory page which is mapped into a virtual address space but is not loaded in physical memory.

When the Physical Memory is much smaller than the Virtual Memory in such a situation Page Fault happens.

FIFO which is also called First In First Out is one of the types of Replacement Algorithms. This algorithm is used in a situation where an Operating system replaces an existing page with the help of memory by bringing a new page from the secondary memory.

FIFO is the simplest among all algorithms which are responsible for maintaining all the pages in a queue for an operating system and also keeping track of all the pages in a queue.

The older pages are kept in the front and the newer ones are kept at the end of the queue. Pages that are in the front are removed first and the pages which are demanded are added.

**PROGRAM**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**PAGE REPLACEMENT - FIRST IN FIRST OUT**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include<stdio.h>

#include<conio.h>

void main()

{

int p,f,a[20],frame[10],i,j,c=0,k=0,flag;

printf("Enter number of pages:");

scanf("%d",&p);

printf("Enter number of frames:");

scanf("%d",&f);

printf("Enter page numbers:");

for(i=0;i<p;i++)

{

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

}

for(i=0;i<f;i++)

frame[i]=-1;

printf("\n");

for(i=0;i<p;i++)

{

flag=0;

for(j=0;j<f;j++)

{

if(frame[j]==a[i])

flag=1;

}

if(flag==0)

{

frame[k]=a[i];

k=(k+1)%f;

c++;

}

printf("%d:\t",a[i]);

for(j=0;j<f;j++)

{

printf("%d\t",frame[j]);

printf("\n");

}

printf("Number of page faults:%d",c);

}

getch();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OUTPUT

Enter number of pages:5

Enter number of frames:3

Enter page numbers:1 2 4 3 2

1: 1 -1 -1

2: 1 2 -1

4: 1 2 4

3: 3 2 4

2: 3 2 4

Number of page faults:4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PAGE REPLACEMENT - LEAST RECENTLY USED

ALGORITHM

Page replacement is a widely used approach in memory management. The idea is simple: the main memory cannot hold all the referred pages in memory. So whenever a new page is referenced, an existing page is replaced by the page that was newly called. The goal of any page replacement mechanism is to minimize the number of page faults.

**Least Recently Used (LRU)** algorithm is a page replacement technique used for memory management. According to this method, the page which is least recently used is replaced. Therefore, in memory, any page that has been unused for a longer period of time than the others is replaced.

This idea is somewhat based on *locality of reference*, that any page that has been unused for a great amount of time is more likely to remain unused. Therefore, it is better to replace that page.

Given a reference string: 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4 and given that there are a total of 3 memory locations available.

1. The first three pages, 0, 1, 2, will give rise to 3-page faults (look at the image above).
2. Now that the table is filled and the next page, 3, is not present in the table, a page must be replaced. Since 0 is the least recently used page, page 3 will replace 0. This will result in a 1 page fault.
3. This process is repeated for the entire reference string.
4. When a number from the reference string already exists in the table, there is no page fault and no effort is required to replace the page. In that case, the reference page will become the recently used page (like when 0 is referenced when the page frames contain 4, 0 and 1.)
5. Make sure you look at whether or not a given page is recently used from the reference string.

PROGRAM

#include<stdio.h>

#include<conio.h>

void main()

{

int p,f,page[20],frame[10],time[10],i,j,k,x,c=0,flag,count=1;

printf("Enter number of page:");

scanf("%d",&p);

printf("Enter number of frames:");

scanf("%d",&f);

printf("Enter page numbers:");

for(i=0;i<p;i++)

{

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

}

for(i=0;i<f;i++)

{

frame[i]=-1;

time[i]=0;

}

printf("\n");

for(i=0;i<p;i++)

{

flag=0;

for(j=0;j<f;j++)

{

if(frame[j]==page[i])

{

flag=1;

time[j]=count;

}

}

if(flag==0)

{

c++;

for(j=0,k=0;j<f;j++)

{

if(time[j]<time[k])

k=j;

}

frame[k]=page[i];

time[k]=count;

}

count++;

printf("%d:\t",page[i]);

for(j=0;j<f;j++)

printf("%d\t",frame[j]);

printf("\n");

}

printf("Number of page faults:%d",c);

getch();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OUTPUT

Enter number of page:8

Enter number of frames:3

Enter page numbers:4 2 3 2 2 3 4 1

4: 4 -1 -1

2: 4 2 -1

3: 4 2 3

2: 4 2 3

2: 4 2 3

3: 4 2 3

4: 4 2 3

1: 4 1 3

Number of page faults:4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PAGE REPLACEMENT - LEAST FREQUENTLY USED

ALGORITHM

In this, it is using the concept of paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when the new page comes in. Whenever a new page is referred to and is not present in memory, the page fault occurs and the Operating System replaces one of the existing pages with a newly needed page. LFU is one such page replacement policy in which the least frequently used pages are replaced. If the frequency of pages is the same, then the page that has arrived first is replaced first.

**Example :**

Given a sequence of pages in an array of pages[] of length N and memory capacity C, find the number of page faults using the Least Frequently Used (LFU) Algorithm.

**Example-1 :**

**Input :** N = 7, C = 3

pages = { 1, 2, 3, 4, 2, 1, 5 }

**Output :**

Page Faults = 6

Page Hits = 1

**Explanation :**

Capacity is 3, thus, we can store maximum 3 pages at a time.

Page 1 is required, since it is not present,

it is a page fault : page fault = 1

Page 2 is required, since it is not present,

it is a page fault : page fault = 1 + 1 = 2

Page 3 is required, since it is not present,

it is a page fault : page fault = 2 + 1 = 3

Page 4 is required, since it is not present,

it replaces LFU page 1 : page fault = 3 + 1 = 4

Page 2 is required, since it is present,

it is not a page fault : page fault = 4 + 0 = 4

Page 1 is required, since it is not present,

it replaces LFU page 3 : page fault = 4 + 1 = 5

Page 5 is required, since it is not present,

it replaces LFU page 4 : page fault = 5 + 1 = 6

**PROGRAM**

#include<stdio.h>

#include<conio.h>

void main()

{

int n,f,frame[50],page[50],count=1,fre[50],fault=0,time[50],flag,j,k,i;

printf("Enter the no pages:");

scanf("%d",&n);

printf("Enter the no of frames:");

scanf("%d",&f);

printf("Enter the ref string:");

for(i=0;i<n;i++)

{

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

}

for(i=0;i<f;i++)

{

frame[i]=-1;

time[i]=0;

fre[i]=0;

}

for(i=0;i<n;i++)

{

flag=0;

for(j=0;j<f;j++)

{

if(frame[j]==page[i])

{

flag=1;

fre[j]++;

}

}

if(flag==0)

{

fault++;

for(j=0,k=0;j<f;j++)

{

if(fre[j]<fre[k])

k=j;

else if(fre[j]==fre[k])

{

if(time[j]<time[k])

k=j;

}

}

frame[k]=page[i];

time[k]=count;

count++;

fre[k]=1;

}

printf("%d",page[i]);

for(j=0;j<f;j++)

{

printf("\t%d",frame[j]);

}

printf("\n");

}

printf("Page fault is %d",fault);

getch();

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

OUTPUT

Enter the no pages:12

Enter the no of frames:3

Enter the ref string:1 2 3 4 1 2 5 1 2 3 4 5

1 1 -1 -1

2 1 2 -1

3 1 2 3

4 4 2 3

1 4 1 3

2 4 1 2

5 5 1 2

1 5 1 2

2 5 1 2

3 3 1 2

4 4 1 2

5 5 1 2

Page fault is 10

**Pgm.No.5**

**FILE ORGANISATION TECHNIQUES**

**AIM**

Simulate the following file organization techniques

a). Single level

b). Two level

c). Hierarchical

**Single Level Directory**

**ALGORITHM**

The single-level directory is **the simplest directory structure**. In it, all files are contained in the same directory which makes it easy to support and understand. A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user.

**PROGRAM**

#include<stdio.h>

#include<string.h>

struct dirct{

char dir[20],file[20][10];

int findex;

};

void main(){

int i, ch=1;

struct dirct d;

char ser[20];

d.findex=0;

printf("Enter the directory name ");

scanf("%s",d.dir);

do{

printf("\n1<<<Create new file\t2<<<Delete a file\t3<<<Search a file\t\n4<<< List files\t\t0<<<Ezxit\n");

printf("Enter your choice ");

scanf("%d",&ch);

switch(ch){

case 1: printf("\nEnter the file name ");

scanf("%s",d.file[d.findex++]);

printf("File created\n");

break;

case 2: printf("\nEnter the file to delete ");

scanf("%s",ser);

for(i=0;i<d.findex;i++){

if(!strcmp(ser,d.file[i]))

{

printf("File removed \n");

strcpy(d.file[i],d.file[d.findex-1]);

d.findex--;

break;

}

}

if(i==d.findex)

printf("No such file or directory\n");

break;

case 3: printf("\nEnter the file to search ");

scanf("%s",ser);

for(i=0;i<d.findex;i++){

if(!strcmp(ser,d.file[i]))

{

printf("\nSearch completed\nFile found at %d position\n",i+1);

break;

}

}

if(i==d.findex){

printf("\nSearch completed\n");

printf("No such file or directory\n");

}

break;

case 4: printf("\nThe files in the directory %s are;\n",d.dir);

for(i=0;i<d.findex;i++)

printf("%s\n", d.file[i]);

break;

}

}while(ch);

printf("\n");

}

**OUTPUT**

Enter the directory name cse

1<<<Create new file 2<<<Delete a file 3<<<Search a file

4<<< List files 0<<<Exit

Enter your choice 1

Enter the file name a

File created

1<<<Create new file 2<<<Delete a file 3<<<Search a file

4<<< List files 0<<<Exit

Enter your choice 1

Enter the file name b

File created

1<<<Create new file 2<<<Delete a file 3<<<Search a file

4<<< List files 0<<<Exit

Enter your choice 1

Enter the file name c

File created

1<<<Create new file 2<<<Delete a file 3<<<Search a file

4<<< List files 0<<<Exit

Enter your choice 4

The files in the directory cse are;

a

b

c

1<<<Create new file 2<<<Delete a file 3<<<Search a file

4<<< List files 0<<<Exit

Enter your choice 3

Enter the file to search b

Search completed

File found at 2 position

1<<<Create new file 2<<<Delete a file 3<<<Search a file

4<<< List files 0<<<Exit

Enter your choice 2

Enter the file to delete b

File removed

1<<<Create new file 2<<<Delete a file 3<<<Search a file

4<<< List files 0<<<Exit

Enter your choice 4

The files in the directory cse are;

a

c

1<<<Create new file 2<<<Delete a file 3<<<Search a file

4<<< List files 0<<<Exit

Enter your choice 3

Enter the file to search b

Search completed

No such file or directory

1<<<Create new file 2<<<Delete a file 3<<<Search a file

4<<< List files 0<<<Exit

Enter your choice 0

**Two Level Directory**

**ALGORITHM**

The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architectureand there are no sub directories. Advantage of single level directory system is that it is easy to find a file in thedirectory. In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched. This effectively solves the name collision problem and isolates users from one another. Hierarchical directory structure allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name. A directory (or subdirectory) contains a set of files or subdirectories.

**PROGRAM**

#include<stdio.h>

#include<string.h>

struct dirct{

char dir[20],file[20][10];

int findex;

};

void main(){

int i,j,ch=1,dindex=0,found=0;

struct dirct d[10];

char ser[20];

for(i=0;i<10;i++)

d[i].findex=0;

do{

printf("\n1<<<Create new directory\t2<<<Create new file\n3<<<Delete new file\t\t");

printf("4<<<Search files\n5<<<List files\t\t\t0<<<Exit\nEnter your choice ");

scanf("%d",&ch);

switch(ch){

case 1: printf("\nEnter the directory name ");

scanf("%s",d[dindex].dir);

dindex++;

printf("Directoiry Created created\n");

break;

case 2: printf("\nEnter the directory name ");

scanf("%s",ser);

found = 0;

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

if(!strcmp(ser,d[i].dir))

{

printf("\nEnter the file name ");

scanf("%s",d[i].file[d[i].findex++]);

printf("File created\n");

break;

}

}

if(i==dindex){

printf("\nSearch completed\n");

printf("No such file or directory\n");

}

break;

case 3: printf("\nEnter the file name ");

scanf("%s",ser);

found = 0;

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

for(j=0;j<d[i].findex;j++){

if(!strcmp(ser,d[i].file[j]))

{

printf("%s is removed\n", d[i].file[j]);

strcpy(d[i].file[j],d[i].file[d[i].findex-1]);

d[i].findex--;

found=1;

break;

}

}

}

if(!found){

printf("\nSearch completed\n");

printf("No such file or directory\n");

}

break;

case 4: printf("\nEnter the file name ");

scanf("%s",ser);

found = 0;

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

for(j=0;j<d[i].findex;j++){

if(!strcmp(ser,d[i].file[j]))

{

printf("%s is removed\n", d[i].file[j]);

found=1;

break;

}

}

}

if(!found){

printf("\nSearch completed\n");

printf("No such file or directory\n");

}

break;

case 5: for(i=0;i<dindex;i++){

printf("\nThe files in the directory %s are;\n",d[i].dir);

for(j=0;j<d[i].findex;j++)

printf("%s\n", d[i].file[j]);

}

break;

}

}while(ch);

printf("\n");

}

**OUTPUT**

1<<<Create new directory 2<<<Create new file

3<<<Delete new file 4<<<Search files

5<<<List files 0<<<Exit

Enter your choice 1

Enter the directory name cse

Directory Created

1<<<Create new directory 2<<<Create new file

3<<<Delete new file 4<<<Search files

5<<<List files 0<<<Exit

Enter your choice 1

Enter the directory name eee

Directoiry Created created

1<<<Create new directory 2<<<Create new file

3<<<Delete new file 4<<<Search files

5<<<List files 0<<<Exit

Enter your choice 2

Enter the directory name cse

Enter the file name cg

File created

1<<<Create new directory 2<<<Create new file

3<<<Delete new file 4<<<Search files

5<<<List files 0<<<Exit

Enter your choice 2

Enter the directory name cse

Enter the file name csaa

File created

1<<<Create new directory 2<<<Create new file

3<<<Delete new file 4<<<Search files

5<<<List files 0<<<Exit

Enter your choice 2

Enter the directory name eee

Enter the file name cp

File created

1<<<Create new directory 2<<<Create new file

3<<<Delete new file 4<<<Search files

5<<<List files 0<<<Exit

Enter your choice 5

The files in the directory cse are;

cg

csaa

The files in the directory eee are;

cp

1<<<Create new directory 2<<<Create new file

3<<<Delete new file 4<<<Search files

5<<<List files 0<<<Exit

Enter your choice 4

Enter the file name cp

cp is found

1<<<Create new directory 2<<<Create new file

3<<<Delete new file 4<<<Search files

5<<<List files 0<<<Exit

Enter your choice 3

Enter the file name cg

cg is removed

1<<<Create new directory 2<<<Create new file

3<<<Delete new file 4<<<Search files

5<<<List files 0<<<Exit

Enter your choice 4

Enter the file name cg

Search completed

No such file or directory

1<<<Create new directory 2<<<Create new file

3<<<Delete new file 4<<<Search files

5<<<List files 0<<<Exit

Enter your choice 5

The files in the directory cse are:

csaa

The files in the directory eee are:

cp

**Pgm.No.6**

**PAGING TECHNIQUES OF MEMORY MANAGEMENT**

**AIM**

Implement different paging techniques of memory management

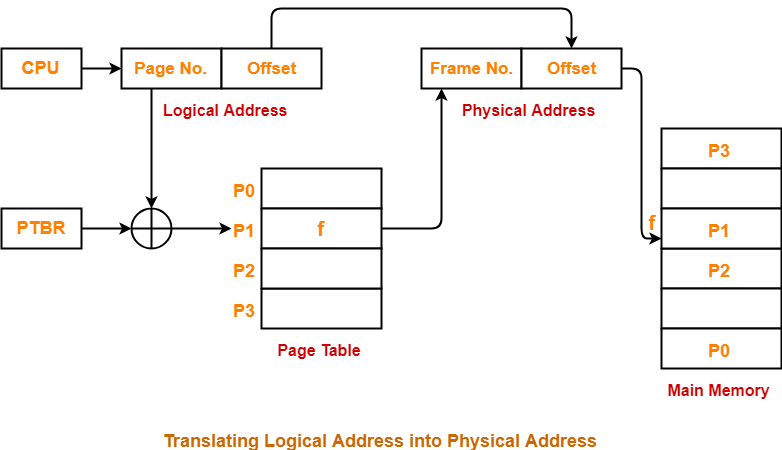
ALGORITHM

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as paging technique.

* The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
* The Logical address Space is also splitted into fixed-size blocks, called **pages**.
* Page Size = Frame Size

Let us consider an example:

* Physical Address = 12 bits, then Physical Address Space = 4 K words
* Logical Address = 13 bits, then Logical Address Space = 8 K words
* Page size = frame size = 1 K words (assumption)



Address generated by CPU is divided into

* **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
* **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into

* **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
* **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

**PROGRAM**

#include<stdio.h>  
void main()  
{  
int memsize=15;  
int pagesize,nofpage;  
int p[100];  
int frameno,offset;  
int logadd,phyadd;  
int i;  
int choice=0;  
printf("\nYour memsize is %d ",memsize);  
printf("\nEnter page size:");  
scanf("%d",&pagesize);  
  
nofpage=memsize/pagesize;  
  
for(i=0;i<nofpage;i++)  
{  
printf("\nEnter the frame of page%d:",i+1);  
scanf("%d",&p[i]);  
}  
  
do  
{  
printf("\nEnter a logical address:");  
scanf("%d",&logadd);  
frameno=logadd/pagesize;  
offset=logadd%pagesize;  
phyadd=(p[frameno]\*pagesize)+offset;  
printf("\nPhysical address is:%d",phyadd);  
printf("\nDo you want to continue(1/0)?:");  
scanf("%d",&choice);  
}while(choice==1);  
}

**OUTPUT:**

Your memsize is 15

Enter page size:5

Enter the frame of page1:2

Enter the frame of page2:4

Enter the frame of page3:7

Enter a logical address:3

Physical address is:13

Do you want to continue(1/0)?:1

Enter a logical address:1

Physical address is:11

Do you want to continue(1/0)?:0

**Pgm.No.7**

**FILE ALLOCATION STRATEGIES**

**AIM**

Simulate following file allocation strategies.

a) Sequential

b) Indexed

c) Linked

ALGORITHM

**Sequntial File allocation**

STEP 1: Start the program.

STEP 2: Gather information about the number of files.

STEP 3: Gather the memory requirement of each file.

STEP 4: Allocate the memory to the file in a sequential manner.

STEP 5: Select any random location from the available location.

STEP 6: Check if the location that is selected is free or not.

STEP 7: If the location is allocated set the flag = 1.

STEP 8: Print the file number, length, and the block allocated.

STEP 9: Gather information if more files have to be stored.

STEP 10: If yes, then go to STEP 2.

STEP 11: If no, Stop the program.

**PROGRAM**

**Sequencial**

#include < stdio.h>

//#include<conio.h>

void main()

{

int f[50], i, st, len, j, c, k, count = 0;

//clrscr();

for(i=0;i<50;i++)

f[i]=0;

printf("Files Allocated are : \n");

x: count=0;

printf(“Enter starting block and length of files: ”);

scanf("%d%d", &st,&len);

for(k=st;k<(st+len);k++)

if(f[k]==0)

count++;

if(len==count)

{

for(j=st;j<(st+len);j++)

if(f[j]==0)

{

f[j]=1;

printf("%d\t%d\n",j,f[j]);

}

if(j!=(st+len-1))

printf(” The file is allocated to disk\n");

}

else

printf(” The file is not allocated \n");

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

goto x;

else

//exit();

return 0

//getch();

}

**OUTPUT**

Files Allocated are :

Enter starting block and length of files: 14 3

14 1

15 1

16 1

The file is allocated to disk

Do you want to enter more file(Yes - 1/No - 0)1

Enter starting block and length of files: 14 1

The file is not allocated

Do you want to enter more file(Yes - 1/No - 0)1

Enter starting block and length of files: 14 4

The file is not allocated

Do you want to enter more file(Yes - 1/No - 0)0

**Indexed**

**ALGORITHM**

STEP 1: Start the program.

STEP 2: Get information about the number of files.

STEP 3: Get the memory requirement of each file.

STEP 4: Allocate the memory to the file by selecting random locations.

STEP 5: Check if the location that is selected is free or not.

STEP 6: If the location is allocated set the flag = 1, and if free set flag = 0.

STEP 7: Print the file number, length, and the block allocated.

STEP 8: Gather information if more files have to be stored.

STEP 9: If yes, then go to STEP 2.

STEP 10: If no, Stop the program.

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void main()

{

int f[50], index[50],i, n, st, len, j, c, k, ind,count=0;

clrscr();

for(i=0;i<50;i++)

f[i]=0;

x:printf("Enter the index block: ");

scanf("%d",&ind);

if(f[ind]!=1)

{

printf("Enter no of blocks needed and no of files for the index %d on the disk : \n", ind);

scanf("%d",&n);

}

else

{

printf("%d index is already allocated \n",ind);

goto x;

}

y: count=0;

for(i=0;i<n;i++)

{

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

if(f[index[i]]==0)

count++;

}

if(count==n)

{

for(j=0;j<n;j++)

f[index[j]]=1;

printf("Allocated\n");

printf("File Indexed\n");

for(k=0;k<n;k++)

printf("%d-------->%d : %d\n",ind,index[k],f[index[k]]);

}

else

{

printf("File in the index is already allocated \n");

printf("Enter another file indexed");

goto y;

}

printf("Do you want to enter more file(Yes - 1/No - 0)");

scanf("%d", &c);

if(c==1)

goto x;

else

exit(0);

getch();

}

**OUTPUT**

Enter the index block: 5

Enter no of blocks needed and no of files for the index 5 on the disk :

4

1 2 3 4

Allocated

File Indexed

5-------->1 : 1

5-------->2 : 1

5-------->3 : 1

5-------->4 : 1

Do you want to enter more file(Yes - 1/No - 0)1

Enter the index block: 4

4 index is already allocated

Enter the index block: 6

Enter no of blocks needed and no of files for the index 6 on the disk :

2

7 8

A5llocated

File Indexed

6-------->7 : 1

6-------->8 : 1

Do you want to enter more file(Yes - 1/No - 0)0

**Linked**

**ALGORITHM**

STEP 1: Start the program.

STEP 2: Gather information about the number of files.

STEP 3: Allocate random locations to the files.

STEP 4: Check if the location that is selected is free or not.

STEP 5: If the location is free set the flag=0 a location is allocated set the flag = 1.

STEP 6: Print the file number, length, and the block allocated.

STEP 7: Gather information if more files have to be stored.

STEP 8: If yes, then go to STEP 2.

STEP 9: If no, Stop the program.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

struct file

{

 char fname[10];

 int start,size,block[10];

}f[10];

main()

{

 int i,j,n;

 clrscr();

 printf("Enter no. of files:");

 scanf("%d",&n);

 for(i=0;i<n;i++)

 {

 printf("Enter file name:");

 scanf("%s",&f[i].fname);

 printf("Enter starting block:");

 scanf("%d",&f[i].start);

 f[i].block[0]=f[i].start;

 printf("Enter no.of blocks:");

 scanf("%d",&f[i].size);

 printf("Enter block numbers:");

 for(j=1;j<=f[i].size;j++)

 {

          scanf("%d",&f[i].block[j]);

 }

 }

 printf("File\tstart\tsize\tblock\n");

 for(i=0;i<n;i++)

 {

          printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);

          for(j=1;j<=f[i].size-1;j++)

                   printf("%d--->",f[i].block[j]);

          printf("%d",f[i].block[j]);

          printf("\n");

 }

 getch();

}

**OUTPUT**

Enter no. of files: 2

Enter file name:a

Enter starting block:1

Enter no. of blocks:2

Enter block number:1

2

Enter filr name: b

Enter starting block:5

Enter no. of blocks:2

Enter block number:3

4

File start size block

a 1 2 1-->2

b 5 2 3-->2

**Pgm.No.8**

**PASS ONE OF TWO PASS ASSEMBLER**

**AIM**

Write a C program to implement pass one of two pass assembler

ALGORITHM

Begin

read first input line

if OPCODE = ‘START’ then begin

save #[Operand] as starting addr

initialize LOCCTR to starting address

write line to intermediate file

read next line

end( if START)

Else

initialize LOCCTR to 0

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

(if symbol)

search OPTAB for OPCODE

if found then

add 3 (instr 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

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}

write last line to intermediate file

Save (LOCCTR – starting address) as program length

End {pass 1}

**PROGRAM**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<conio.h>

char opcode[10],operand[10],label[10],ch,t1[20],t2[20],t3[10];

int locctr,start,len,s=-1,o=-1,i,j=0,flag,size=0,opd;

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

struct SYMTAB

{

char label[10];

int addr;

}ST[30];

struct OPTAB

{

char opcode[10],hexcode[10];

}OT[30];

void read\_OPTAB()

{

while(1)

{

o++;

fscanf(fp2,"%s%s",OT[o].opcode,OT[o].hexcode);

if(getc(fp2)==EOF) break;

}

}

void read\_line()

{

strcpy(t1,"");strcpy(t2,"");strcpy(t3,"");

fscanf(fp1,"%s",t1);

if(getc(fp1)!='\n')

{

fscanf(fp1,"%s",t2);

if(getc(fp1)!='\n')

{

fscanf(fp1,"%s",t3);

}

}

if(strcmp(t1,"")!=0 && strcmp(t2,"")!=0 && strcmp(t3,"")!=0 )

{

strcpy(label,t1);strcpy(opcode,t2); strcpy(operand,t3);

}

else if(strcmp(t1,"")!=0 && strcmp(t2,"")!=0 && strcmp(t3,"")==0 )

{

strcpy(label,"");strcpy(opcode,t1); strcpy(operand,t2);

}

else if(strcmp(t1,"")!=0 && strcmp(t2,"")==0 && strcmp(t3,"")==0 )

{

strcpy(label,"");strcpy(opcode,t1);strcpy(operand,"");

}

}

int main()

{

fp1=fopen("z:\\input.txt","r");

fp2=fopen("z:\\optab.txt","r");

fp3=fopen("z:\\symtab.txt","w");

fp4=fopen("z:\\intermed.txt","w");

fp5=fopen("z:\\length.txt","w");

read\_OPTAB(); // read OPTAB

//read\_line(); // read the first line

fscanf(fp1,"%s%s%x",label,opcode,&opd);

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

{

start=opd;

locctr=start;

fprintf(fp4,"\t%s\t%s\t%x\n",label,opcode,opd);

read\_line(); // read the next line

}

else

locctr=0;

// do until END

while(strcmp(opcode,"END")!=0)

{

fprintf(fp4,"%x\t%s\t%s\t%s\n",locctr,label,opcode,operand); //Write this line into intermediate file

if(strcmp(label,"")!=0) //if label present

{

for(i=0;i<=s;i++)

{

if(strcmp(ST[i].label,label)==0) //if SYMTAB contains the present label

{

printf("Error");

exit(0);

}

}

s++;

strcpy(ST[s].label,label); ST[s].addr=locctr; //if SYMTAB doesnot contains the present label

}

// Search OPTAB for opcode

flag=0;

for(i=0;i<=o;i++)

{

if(strcmp(opcode,OT[i].opcode)==0) //if opcode is in OPTAB

{

locctr+=0x3;

size=size+3;

flag=1;

break;

}

}

if(flag==0)

{

if(strcmp(opcode,"WORD")==0) //if opcode = "WORD"

{

locctr+=0x3;

size=size+3;

}

else if(strcmp(opcode,"RESW")==0) //if opcode = "RESW"

locctr+=(0x3\*(atoi(operand)));

else if(strcmp(opcode,"RESB")==0) //if opcode = "RESB"

locctr+=(atoi(operand));

else if(strcmp(opcode,"BYTE")==0) //if opcode = "BYTE"

{

locctr+=0x1;

size=size+1;

}

}

read\_line(); //read the next line

}

fprintf(fp4,"\t%s\t%s\t%s\n",label,opcode,operand); //Write last line into intermediate file

//write the contents of ST to symboltab.txt

for(i=0;i<=s;i++)

{

fprintf(fp3,"%s\t%d",ST[i].label,ST[i].addr);

if(i!=s) fprintf(fp3,"\n");

}

fprintf(fp5,"%x\n%x",locctr-start,size);

fclose(fp1);

fclose(fp2);

fclose(fp3);

fclose(fp4);

fclose(fp5);

}

**Pgm.No.9**

**PASS TWO OF TWO PASS ASSEMBLER**

**AIM**

Write a program to implement pass one of two pass assembler

**ALGORITHM**

Begin

read 1st input  line

if OPCODE = ‘START’ then

begin

write listing line

read next input line

end

write Header record to object program

initialize 1st Text record

while OPCODE != ‘END’ do

begin

if this is not comment line then

begin

search OPTAB for OPCODE

if found then

Begin

if there is a symbol in OPERAND field then

Begin

search SYMTAB for OPERAND field then

if found then

Begin

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

else if OPCODE = ‘BYTE’ or ‘WORD” then

convert constant to object code

if object code doesn’t fit into current Text record then

Begin

Write text record to object code

initialize new Text record

end

add object code to Text record

end {if not comment}

write listing line

read next input line

end

write listing line

read next input line

write last listing line

End {Pass 2}

**PROGRAM**

#include<stdio.h>

#include<string.h>

#include<math.h>

#include<stdlib.h>

#include<conio.h>

char t1[20],t2[20],t3[10],t4[10],address[10],label[10],opcode[10],operand[10],length[10],size[10],a[10],ad[10],symbol[10],add[10],st\_addr[10]; //ch;

int s=-1,o=-1,i,j;

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

struct SYMTAB

{

char label[10];

char addr[10];

}ST[30];

struct OPTAB

{

char opcode[10],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("intermed.txt","r");

fp2=fopen("symtab.txt","r");

fp3=fopen("optab.txt","r");

fp4=fopen("length.txt","r");

fp5=fopen("assmlist.txt","w");

fp6=fopen("objcode.txt","w");

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

read\_OPTAB();

read\_SYMTAB();

fscanf(fp1,"%s%s%s",label,opcode,operand);

strcpy(st\_addr,operand);

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

{

fprintf(fp5,"\t%s\t%s\t%s\n",label,opcode,operand);

fprintf(fp6,"H%s 00%s0000%s\n",label,operand,length);

fprintf(fp6,"T00%s%x",operand,atoi(size));

read\_line();

}

while(strcmp(opcode,"END")!=0)

{

if(strcmp(opcode,"BYTE")==0)

{

fprintf(fp5,"%s\t%s\t%s\t%s\t",address,label,opcode,operand);

sprintf(ad,"%x",operand[2]);

fprintf(fp5,"%s\n",ad);

fprintf(fp6,"%s",ad);

}

else if(strcmp(opcode,"WORD")==0)

{

sprintf(a,"%x",atoi(operand));

fprintf(fp5,"%s\t%s\t%s\t%s\t00000%s\n",address,label,opcode,operand,a);

fprintf(fp6,"00000%s",a);

}

else if(strcmp(opcode,"RESB")==0)

{

fprintf(fp5,"%s\t%s\t%s\t%s\n",address,label,opcode,operand);

}

else if(strcmp(opcode,"RESW")==0)

{

fprintf(fp5,"%s\t%s\t%s\t%s\n",address,label,opcode,operand);

}

else

{ j=0;

while(strcmp(opcode,OT[j].opcode)!=0)

j++;

i=0;

while(strcmp(operand,ST[i].label)!=0)

i++;

fprintf(fp5,"%s\t%s\t%s\t%s\t%s%s\n",address,label,opcode,operand,OT[j].hexcode,ST[i].addr);

fprintf(fp6,"%s%s",OT[j].hexcode,ST[i].addr);

}

read\_line();

}

fprintf(fp5,"\t%s\t%s\t%s",label,opcode,operand);

fprintf(fp6,"\nE^00%s",st\_addr);

getch();

}

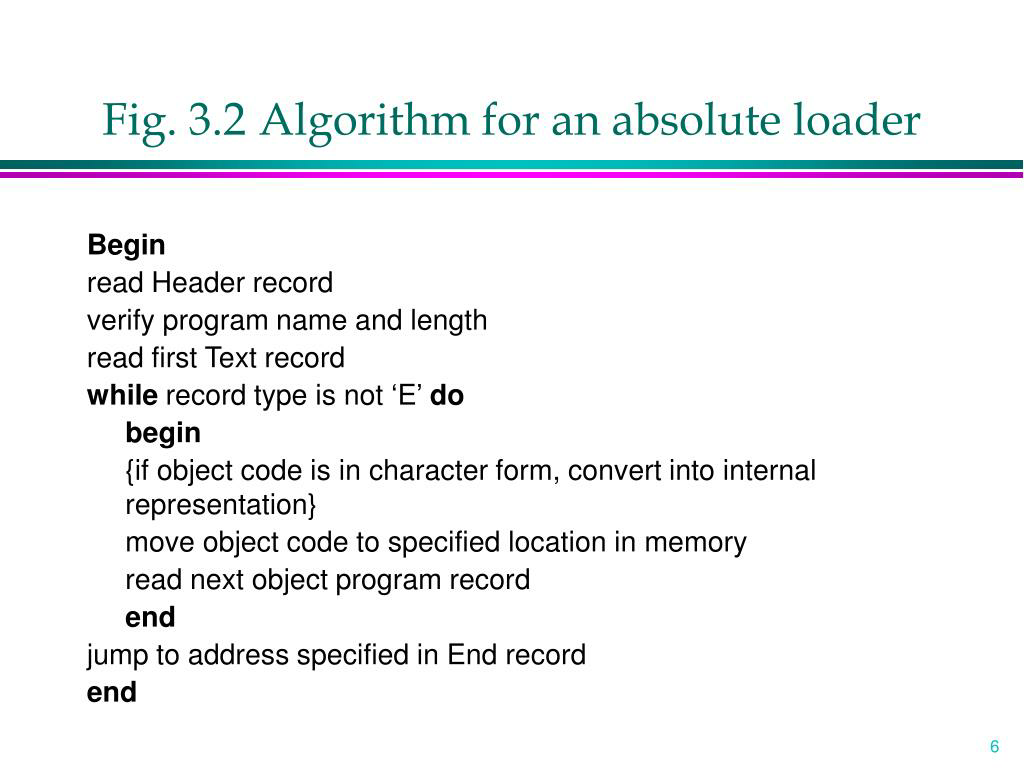
**Pgm.No.10**

**ABSOLUTE LOADER**

**AIM**

Write a C program to implement Absolute Loader

ALGORITHM



**PROGRAM**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void main()

{

FILE \*fp;

int i,addr1,l,j,staddr1;

char name[10],line[50],name1[10],addr[10],rec[10],ch,staddr[10],exeaddr[10];

printf("enter program name:");

scanf("%s",&name);

fp=fopen("Z:\\KRISH\\absolue\\OBJECTCODE.txt","r");

fscanf(fp,"%s",line);

for(i=2,j=0;i<8,j<6;i++,j++)

{

name1[j]=line[i];

}

name1[j]='\0';

printf("name from obj.%s\n",name1);

if(strcmp(name,name1)==0)

{

while(!feof(fp))

{

fscanf(fp,"%s",line);

while(line[0]=='T')

{

for(i=2,j=0;i<8,j<6;i++,j++)

{

staddr[j]=line[i];

}

staddr[j]='\0';

staddr1=atoi(staddr);

i=12;

while(line[i]!='$')

{

if(line[i]!='^')

{

printf("00%d\t%c%c\n",staddr1,line[i],line[i+1]);

staddr1++;

i=i+2;

}

else

i++;

}

fscanf(fp,"%s",line);

}

if(line[0]='E')

{

for(i=2,j=0;i<8,j<6;i++,j++)

{

exeaddr[j]=line[i];

}

exeaddr[j]='\0';

printf("jump to execution address:%s",exeaddr);

fscanf(fp,"%s",line);

}

}

}

fclose(fp);

}

objectcode.txt

H^SAMPLE^001000^0035

T^001000^0C^001003^071009$

T^002000^03^111111$

H^SAMPLE^001000^0035

T^001000^0C^001003^071009$

T^002000^03^111111$

E^001000

OUTPUT

enter program name:SAMPLE

name from obj. SAMPLE

001000   00

001001   10

001002   03

001003   07

001004   10

001005   09

002000   11

002001   11

002002   11

jump to execution address:001000

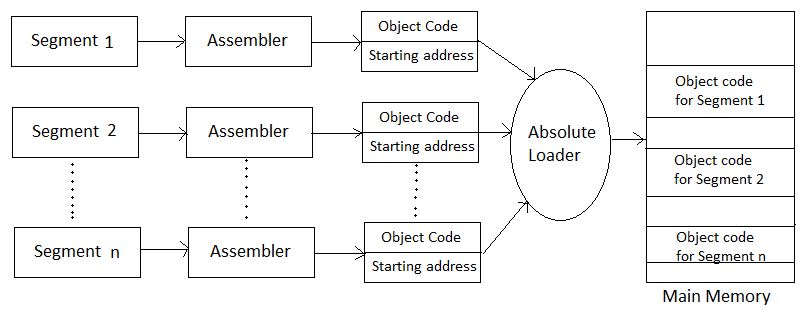
**Pgm.No.11**

**RELOCATING LOADER**

**AIM**

Write a C program to implement relocating loader

ALGORITHM

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

void convert(char h[12]);

char bitmask[12];

char bit[12]={0};

void main()

{char add[6],length[10],input[10],binary[12],relocbit,ch,pn[5];

int start,inp,len,i,address,opcode,addr,actualadd,tlen;

FILE \*fp1,\*fp2;

clrscr();

printf("\n\n Enter the actual starting address : ");

scanf("%x",&start);

fp1=fopen("RLIN.txt","r");

fp2=fopen("RLOUT.txt","w");

fscanf(fp1,"%s",input);

fprintf(fp2," ----------------------------\n");

fprintf(fp2," ADDRESS\tCONTENT\n");

fprintf(fp2," ----------------------------\n");

while(strcmp(input,"E")!=0)

{

if(strcmp(input,"H")==0)

{

fscanf(fp1,"%s",pn);

fscanf(fp1,"%x",add);

fscanf(fp1,"%x",length);

fscanf(fp1,"%s",input);

}

if(strcmp(input,"T")==0)

{

fscanf(fp1,"%x",&address);

fscanf(fp1,"%x",&tlen);

fscanf(fp1,"%s",bitmask);

address+=start;

convert(bitmask);

len=strlen(bit);

if(len>=11)

len=10;

for(i=0;i<len;i++)

{

fscanf(fp1,"%x",&opcode);

fscanf(fp1,"%x",&addr);

relocbit=bit[i];

if(relocbit=='0')

actualadd=addr;

else

actualadd=addr+start;

fprintf(fp2,"\n %x\t\t%x%x\n",address,opcode,actualadd);

address+=3;

}

fscanf(fp1,"%s",input);

}

}

fprintf(fp2," ----------------------------\n");

fcloseall();

printf("\n\n The contents of output file(RLOUT.TXT n\n");

fp2=fopen("RLOUT.txt","r");

ch=fgetc(fp2);

while(ch!=EOF)

{

printf("%c",ch);

ch=fgetc(fp2);

}

fclose(fp2);

getch();

}

void convert(char h[12])

{

int i,l;

strcpy(bit,"");

l=strlen(h);

for(i=0;i<l;i++)

{

switch(h[i])

{

case '0':

strcat(bit,"0");

break;

case '1':

strcat(bit,"1");

break;

case '2':

strcat(bit,"10");

break;

case '3':

strcat(bit,"11");

break;

case '4':

strcat(bit,"100");

break;

case '5':

strcat(bit,"101");

break;

case '6':

strcat(bit,"110");

break;

case '7':

strcat(bit,"111");

break;

case '8':

strcat(bit,"1000");

break;

case '9':

strcat(bit,"1001");

break;

case 'A':

strcat(bit,"1010");

break;

case 'B':

strcat(bit,"1011");

break;

case 'C':

strcat(bit,"1100");

break;

case 'D':

strcat(bit,"1101");

break;

case 'E':

strcat(bit,"1110");

break;

case 'F':

strcat(bit,"1111");

break;

}

}

}

**INPUT file:**  
  
RLIN.TXT  
  
H COPY 000000 00107A  
T 000000 1E FFC 14 0033 48 1039 10 0036 28 0030 30 0015 48 1061 3C 0003 20 002A 1C 0039 30 002D  
T 002500 15 E00 1D 0036 48 1061 18 0033 4C 1000 80 1000 60 1003  
E 000000  
  
**OUTPUT**  
  
Enter the actual starting address : 4000  
The contents of output file(RLOUT.TXT):  
-----------------------------------------------------  
ADDRESS CONTENT  
-----------------------------------------------------  
4000 144033  
4003 485039  
4006 104036  
4009 284030  
400c 304015  
400f 485061  
4012 3c4003  
4015 20402a  
4018 1c4039  
401b 30402d  
6503 1d4036  
6506 184033  
6509 4c1000  
650c 801000  
650f 601003

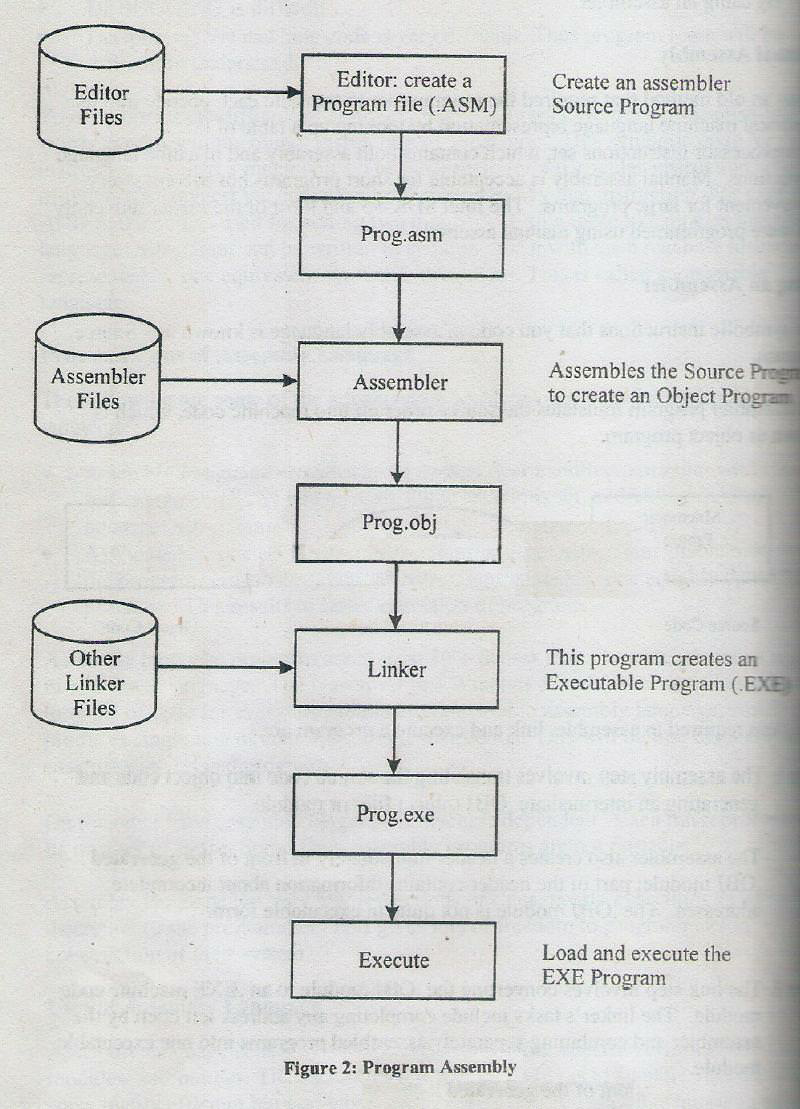
**Pgm.No.12**

**TWO PASS MACRO PROCESSOR**

**AIM**

Write a C program to implement two pass macro processor

**ALGORITHM**

**PROGRAM**

**Pass one of two pass macro processor**

#include<stdio.h>

#include<string.h>

void main()

{

char macros[20][10], label[20],opcode[20],operand[20];

int i, j, n,m=0;

FILE \*fp1, \*fp[10];

fp1=fopen("inputm.txt","r");

fscanf(fp1,"%s%s%s",label,opcode,operand);

while(strcmp(opcode,"END")!=0)

{

if(!strcmp(opcode,"MACRO")){

fp[m]=fopen(operand,"w");

m++;

fscanf(fp1,"%s%s%s",label,opcode,operand);

while(strcmp(opcode,"MEND")!=0){

fprintf(fp[m-1],"%s\t%s\t%s\n",label,opcode,operand);

fscanf(fp1,"%s%s%s",label,opcode,operand);

}

}

fscanf(fp1,"%s%s%s",label,opcode,operand);

}

}

**INPUT FILES**

**inputm.txt**

\*\* MACRO m1

\*\* LDA ALPHA

\*\* STA BETA

\*\* MEND \*\*

\*\* MACRO m2

\*\* MOV a,b

\*\* MEND \*\*

\*\* START 1000

\*\* LDA a

\*\* CALL m1

\*\* CALL m2

\*\* END \*\*

**OUTPUT FILES**

**m1.txt**

\*\* LDA ALPHA

\*\* STA BETA

**m2.txt**

\*\* MOV a,b

**Pass two of two pass assemblers**

**PROGRAM**

#include<stdio.h>

#include<string.h>

void main()

{

char macros[20][10], label[20],opcode[20],operand[20];

int i, j, n,m=0;

FILE \*fp1, \*fp[10],\*fp2;

fp1=fopen("inputm.txt","r");

fp2=fopen("macro\_out.txt","w");

fscanf(fp1,"%s%s%s",label,opcode,operand);

while(strcmp(opcode,"END")!=0)

{

if(!strcmp(opcode,"CALL"))

{

fp[m]=fopen(operand,"r");

m++;

fscanf(fp[m-1],"%s%s%s",label,opcode,operand);

while(!feof(fp[m-1]))

{

fprintf(fp2,"%s\t%s\t%s\n",label,opcode,operand);

fscanf(fp[m-1],"%s%s%s",label,opcode,operand);

}

}

else

{

fprintf(fp2,"%s\t%s\t%s\n",label,opcode,operand);

}

fscanf(fp1,"%s%s%s",label,opcode,operand);

}

fprintf(fp2,"%s\t%s\t%s\n",label,opcode,operand);

}

**INPUT FILES**

**inputm.txt**

\*\* MACRO m1

\*\* LDA ALPHA

\*\* STA BETA

\*\* MEND \*\*

\*\* MACRO m2

\*\* MOV a,b

\*\* MEND \*\*

\*\* START 1000

\*\* LDA a

\*\* CALL m1

\*\* CALL m2

\*\* END \*\*

**OUTPUT FILES**

**m1.txt**

\*\* LDA ALPHA

\*\* STA BETA

**m2.txt**

\*\* MOV a,b

**output file**

\*\* MACRO m1

\*\* LDA ALPHA

\*\* STA BETA

\*\* MEND \*\*

\*\* MACRO m2

\*\* MOV a,b

\*\* MEND \*\*

\*\* START 1000

\*\* LDA a

\*\* END \*\*

**Pgm.No.13**

**SINGLE PASS MACRO PROCESSOR**

**AIM**

Write a C program to implement single pass macro processor

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

#include<string.h>

int m=0,i,j,flag=0;

char c,\*s1,\*s2,\*s3,\*s4,str[50]=" ",str1[50]=" ";

char mac[10][10];

void main()

{

FILE \*fpm=fopen("macro.txt","r");

FILE \*fpi=fopen("minput.txt","r");

FILE \*fpo=fopen("moutput.txt","w");

clrscr();

while(!feof(fpm))

{

fgets(str,50,fpm);

s1=strtok(str," ");

s2=strtok(NULL," ");

if(strcmp(s1,"MACRO")==0)

{

strcpy(mac[m],s2);

m++;

}

s1=s2=NULL;

}

fgets(str,50,fpi);

while(!feof(fpi))

{

flag=0;

strcpy(str1,str);

for(i=0;i<m;i++)

{

if(strcmp(str1,mac[i])==0)

{

rewind(fpm);

while(!feof(fpm))

{

fgets(str,50,fpm);

s2=strtok(str," ");

s3=strtok(NULL," ");

if(strcmp(s2,"MACRO")==0&&strcmp(s3,str1)==0)

{

fgets(str,50,fpm);

strncpy(s4,str,4);

s4[4]='\0';

while(strcmp(s4,"MEND")!=0)

{

fprintf(fpo,"%s",str);

printf("\n####%s",str);

fgets(str,50,fpm);

strncpy(s4,str,4);

s4[4]='\0';

}

}

}

flag=1;

break;

}

}

if(flag==0)

{

fprintf(fpo,"%s",str);

printf("%s",str);

}

fgets(str,50,fpi);

}

fclose(fpm);

fclose(fpi);

fclose(fpo);

}

**INPUT FILES**

**Macro.txt**

MACRO ADD1

MOV A,B

ADD C

MEND

MACRO SUB1

STORE C

MEND

**MInput.txt**

MOV B,10

MOV C,20

ADD1

MUL C

SUB1

END

**OUTPUT**

**MOutput.txt**

MOV B,10

MOV C,20

MOV A,B

ADD C

MUL C

STORE C

END

**Viva Questions and Answers**

1. What is CPU Scheduler?  
   Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

CPU scheduling decisions may take place when a process:

* 1. .Switches from running to waiting state.
  2. .Switches from running to ready state. c
  3. .Switches from waiting to ready.
  4. Terminates.

Scheduling under a. and d. is non-pre-emptive.

All other scheduling is pre-emptive

1. What are all the scheduling algorithms?
2. FCFS(First Come First Serve)
3. SJF(Shortest Job First)
4. Round robin
5. Priority Scheduling algorithms
6. Explain FCFS(First Come First Served)?
   1. The process that requests the CPU first is allocated the CPU first. The code for
   2. FCFS scheduling is simple to write and understand.
   3. Explain SJF(Shortest Job First)?
   4. The process which has the less burst time execute first. If both process have same burst time then FCFS will be used.
7. What is CPU utilization?

We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent.

1. What is Throughput?

The amount of work is being done by the CPU. One unit of work is the number of processes that are completed per unit time, called throughput

1. What is Turnaround time.

The interval from the time of submission of a process to the time of completion is the turnaround time

1. What is waiting time?

Waiting time is the sum of the periods spent waiting in the ready queue.

1. What is Response time?

the time from the submission of a request until the first response is produced.

1. What are short, long and medium-term scheduling?
2. Long term scheduler determines which programs are admitted to the system for processing. It controls the degree of multiprogramming. Once admitted, a job becomes a process.
3. Medium term scheduling is part of the swapping function. This relates to processes that are in a blocked or suspended state. They are swapped out of real-memory until they are ready to execute. The swapping-in decision is based on memory-management criteria.
4. Short term scheduler, also known as a dispatcher executes most frequently, and makes the finest-grained decision of which process should execute next. This scheduler is invoked whenever an event occurs. It may lead to interruption of one process by pre-emption.
5. What are turnaround time and response time?

Turnaround time is the interval between the submission of a job and its completion.

1. What is pre-emptive and non-pre-emptive scheduling?
2. Pre-emptive scheduling: The pre-emptive scheduling is prioritized. The highest priority process should always be the process that is currently utilized.
3. Non-Pre-emptive scheduling: When a process enters the state of running, the state of that process is not deleted from the scheduler until it finishes its service time.
4. What is deadlock?

Deadlock is a situation that when two or more process waiting for each other and holding the resource which is required by another process.

1. What are the necessary conditions to occur deadlock?

Mutual exclusion: At least one resource must be held in a non-sharable mode, that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.   
Hold and wait: A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.   
No pre-emption: Resources cannot be pre-empted.; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.   
Circular wait: A set {P$, Pi, ..., Pn} of waiting processes must exist such that P-0 is waiting for a resource held by P\, P\ is waiting for a resource held by P?, •••, P.,--i is waiting for a resource held by Pn, and P, is waiting for a resource held by Pn.

1. Explain about resource allocation graph?

Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph. If the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.

1. What are the methods to handle the dead locks?
2. We can use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlock state.
3. We can allow the system to enter a deadlock state, detect it, and recover.
4. We can ignore the problem altogether and pretend that deadlocks never occur in the system.
5. The third solution is the one used by most operating systems
6. What are the deadlock avoidance algorithms?

A dead lock avoidance algorithm dynamically examines there source-allocation state to ensure that a circular wait condition can never exist. The resource allocation state is defined by the number of available and allocated resources, and the maximum demand of the process. There are two algorithms:

Resource allocation graph algorithm

1. Banker’s algorithm
2. Safety algorithm
3. Resource request algorithm
4. What is Bankers Algorithm.

It is an algorithm which used in a banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.

1. What is a Safe State and what is its use in deadlock avoidance?

When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state. System is in safe state if there exists a safe sequence of all processes. Deadlock Avoidance: ensure that a system will never enter an unsafe state.

1. What is starvation and aging?

Starvation is Resource management problem where a process does not get the resources it needs for a long time because the resources are being allocated to other processes.

1. What is a Safe State and its’ use in deadlock avoidance?

When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state

* System is in safe state if there exists a safe sequence of all processes.
* Sequence is safe if for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished. When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate. When Pi terminates, Pi+1 can obtain its needed resources, and so on.
* Deadlock Avoidance Þ ensure that a system will never enter an unsafe state.

1. Recovery from Deadlock?

* Process Termination:

->Abort all deadlocked processes.

->Abort one process at a time until the deadlock cycle iseliminated.

->In which order should we choose to abort?

* Priority of the process.

How long process has computed, and how much longer tocompletion.

Resources the process has used.

Resources process needs to complete.

How many processes will need to be terminated?

Is process interactive or batch?

* Resource Preemption:

->Selecting a victim – minimize cost.

->Rollback – return to some safe state, restart process for thatstate.

->Starvation – same process may always be picked as victim,include number of rollback in cost factor.

1. Why paging is used?   
   Paging is solution to external fragmentation problem which is to permit the logical address space of a process to be non-contiguous, thus allowing a process to be allocating physical memory wherever the latter is available.
2. What is virtual memory?

Virtual memory is memory management technique which is used to execute the process which has more than actual memory size.

1. What is Demand Paging?

It is memory management technique used in virtual memory such that page will not load into the memory until it is needed.

1. What are all page replacement algorithms?
   1. FIFO(First in First out)  
      2. Optimal Page Replacement   
      3. LRU(Least-Recently-used)
2. Which page replacement algorithm will have less page fault rate?

Optimal Page Replacement

1. What is thrashing?

It is situation that CPU spends more time on paging than executing.

1. What is swapping

A process must be in memory to be executed. A process, however, can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution. This process is called swapping.

1. What is fragmentation?

fragmentation is a phenomenon in which storage space is used inefficiently, reducing capacity or performance.

1. Explain External fragmentation?

As processes are loaded and removed from memory, the free memory space is broken into little pieces. External fragmentation exists when there is enough total memory space to satisfy a request, but the available spaces are not contiguous.

1. Explain Internal fragmentation?

Consider a multiple-partition allocation scheme with a hole of 18,464 bytes. Suppose that the next process requests 18,462 bytes. If we allocate exactly the requested block, we are left with a hole of 2 bytes. The overhead to keep track of this hole will be substantially larger than the hole itself. The general approach to avoiding this problem is to break the physical memory into fixed-sized blocks and allocate memory in units based on block size. With this approach, the memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is internal fragmentation.

1. What is paging?

Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous. Paging avoids the considerable problem of fitting memory chunks of varying sizes onto the backing store.

1. What is frame?

Breaking main memory into fixed number of blocks called frames.

1. What is page?

Breaking logical memory into blocks of same size is page.

1. What is the best page size when designing an operating system?

The best paging size varies from system to system, so there is no single best when it comes to page size. There are different factors to consider in order to come up with a suitable page size, such as page table, paging time, and its effect on the overall efficiency of the operating system.

1. What is virtual memory?   
   Virtual memory is hardware technique where the system appears to have more memory that it actually does. This is done by time-sharing, the physical memory and storage parts of the memory one disk when they are not actively being used.
2. What is Throughput, Turnaround time, waiting time and Response time?   
   Throughput – number of processes that complete their execution per time unit. Turnaround time – amount of time to execute a particular process. Waiting time – amount of time a process has been waiting in the ready queue. Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment).
3. Explain Segmentation with paging?

Segments can be of different lengths, so it is harder to find a place for a segment in memory than a page. With segmented virtual memory, we get the benefits of virtual memory but we still have to do dynamic storage allocation of physical memory. In order to avoid this, it is possible to combine segmentation and paging into a two-level virtual memory system. Each segment descriptor points to page table for that segment. This give some of the advantages of paging (easy placement) with some of the advantages of segments (logical division of the program).

1. Under what circumstances do page faults occur? Describe the actions taken by the operating system when a page fault occurs?

A page fault occurs when an access to a page that has not been brought into main memory takes place. The operating system verifies the memory access, aborting the program if it is invalid. If it is valid, a free frame is located and I/O is requested to read the needed page into the free frame. Upon completion of I/O, the process table and page table are updated and the instruction is restarted

40 . Define the basic functions of assembler.  
\* Translating mnemonic operation codes to their machine language  
equivalents.  
\* Assigning machine addresses to symbolic labels used by the  
programmer.  
  
41. What is meant by assembler directives? Give example.  
These are the statements that are not translated into machine instructions, but they provide instructions to assembler itself.  
example START,END,BYTE,WORD,RESW and RESB.  
42. What are forward references?  
It is a reference to a label that is defined later in a program.  
Consider the statement  
10 1000 STL RETADR  
80 1036 RETADR RESW 1  
The first instruction contains a forward reference RETADR. If we attempt to translate the program line by line, we will unable to process the statement in line10 because we do not know the address that will be assigned to RETADR .The address is assigned later(in line 80) in the program.  
  
43. What are the three different records used in object program?  
The header record, text record and the end record are the three different records used in object program.  
The header record contains the program name, starting address and  
length of the program.  
Text record contains the translated instructions and data of the program.  
End record marks the end of the object program and specifies the address in the program where execution is to begin.  
  
44. What is the need of SYMTAB (symbol table) in assembler?  
The symbol table includes the name and value for each symbol in the  
source program, together with flags to indicate error conditions. Some times it may contain details about the data area. SYMTAB is usually organized as a hash table for efficiency of insertion and retrieval.  
  
45. What is the need of OPTAB (operation code table) in assembler?  
The operation code table contains the mnemonic operation code and its  
machine language equivalent. Some assemblers it may also contain information about instruction format and length. OPTAB is usually organized as a hash table, with mnemonic operation code as the key.  
  
46. Write the steps required to translate the source program to object program.  
• Convert mnemonic operation codes to their machine language  
equivalents.  
• Convert symbolic operands to their equivalent machine addresses  
• Build the machine instruction in the proper format.  
• Convert the data constants specified in the source program into their internal machine representation  
• Write the object program and assembly listing.  
  
47. What is the use of the variable LOCCTR (location counter) in assembler?  
This variable is used to assign addresses to the symbols. LOCCTR is  
initialized to the beginning address specified in the START statement. After each source statement is processed the length of the assembled instruction or data area to be generated is added to LOCCTR and hence whenever we reach a label in the source program the current value of  
LOCCTR gives the address associated with the label.  
48 . Define load and go assembler.  
One pass assembler that generates their object code in memory for  
immediate execution is known as load and go assembler. Here no object programmer is written out and hence no need for loader.  
  
49. What are the two different types of jump statements used in MASM assembler?  
• Near jump  
A near jump is a jump to a target in the same segment and it is  
assembled by using a current  
code segment CS.  
• Far jump  
A far jump is a jump to a target in a different code segment and it is  
assembled by using different segment registers .  
  
50. Differentiate the assembler directives RESW and RESB.  
RESW –It reserves the indicated number of words for data area.  
Eg: 10 1003 THREE RESW 1  
In this instruction one word area (3 bytes) is reserved for the symbol  
THREE. If the memory is byte addressable then the address assigned for the next symbol is 1006.  
RESB –It reserves the indicated number of bytes for data area.  
Eg: 10 1008 INPUT RESB 1  
In this instruction one byte area is reserved for the symbol INPUT .Hence the address assigned for the next symbol is 1009.  
  
51. Write down the pass numbers (PASS 1/ PASS 2) of the following activities that occur in a two pass assembler:  
a. Object code generation  
b. Literals added to literal table  
c. Listing printed  
d. Address location of local symbols  
*Answer:*  
a. Object code generation - PASS 2  
b. Literals added to literal table – PASS 1  
c. Listing printed – PASS2  
d. Address location of local symbols – PASS1  
52. What is meant by machine independent assembler features?  
The assembler features that do not depend upon the machine  
architecture are known as machine independent assembler features.  
Eg: program blocks, Literals.  
  
53. What is meant by external references?  
Assembler program can be divided into many sections known as control  
sections and each control section can be loaded and relocated independently of the others. If the instruction in one control section need to refer instruction or data in another control section.the assembler is unable to process these references in normal way. Such  
references between control are called external references.  
54. What is the use of the assembler directive START?  
The assembler directive START gives the name and starting address of  
the program.  
The format is  
PN START 1000  
Here  
PN – Name of the program  
1000 - Starting address of the program.

55. What are the basic functions of loaders?  
• Loading – brings the object program into memory for execution  
• Relocation – modifies the object program so that it can be loaded at an address different from the location originally specified  
• Linking – combines two or more separate object programs and also  
supplies the information needed to reference them.

56. Define macro processor.  
Macro processor is system software that replaces each macro instruction with the corresponding group of source language statements. This is also called as expanding of macros.

57. What do macro expansion statements mean?  
These statements give the name of the macro instruction being invoked  
and the arguments to be used in expanding the macros. These statements are also known as macro call.

58. What are the directives used in macro definition?  
MACRO - it identifies the beginning of the macro definition  
MEND - it marks the end of the macro definition  
  
59. What are the data structures used in macro processor?  
DEFTAB – the macro definitions are stored in a definition table i.e. it contains a macro prototype and the statements that make up the macro body.  
NAMTAB – it is used to store the macro names and it contains two  
pointers for each macro instruction which indicate the starting and end location of macro definition in DEFTAB. it also serves as an index to DEFTAB  
ARGTAB – it is used to store the arguments during the expansion of macro invocations.

60. Define conditional macro expansion.  
If the macro is expanded depends upon some conditions in macro  
definition (depending on the arguments supplied in the macro expansion) then it is called as conditional macro expansion.

61. What is the use of macro time variable?  
Macro time variable can be used to store working values during the macro expansion. Any symbol that begins with the character & and then is not a macro instruction parameter is assumed to be a macro time variable.

62. What are the statements used for conditional macro expansion?  
IF-ELSE-ENDIF statement  
WHILE-ENDW statement

63. What is meant by positional parameters?  
If the parameters and arguments were associated with each other  
according to their positions in the macro prototype and the macro invocation statement, then these parameters in macro definitions are called as positional parameters.  
  
64. What are known as nested macro call?  
The statement, in which a macro calls on another macro, is called nested macro call. In the nested macro call, the call is done by outer macro and the macro called is the inner macro.  
  
65. How the macro is processed using two passes?  
Pass1: processing of definitions  
Pass 2:actual-macro expansion.  
  
66. Give the advantage of line by line processors.  
• It avoids the extra pass over the source program during assembling.  
• It may use some of the utility that can be used by language translators so that can be loaded once.  
  
67. What is meant by line by line processor?  
This macro processor reads the source program statements, process the  
statements and then the output lines are passed to the language translators as they are generated, instead of being written in an expanded file.

68. Give the advantages of general-purpose macro processors.  
• The programmer does not need to learn about a macro facility for each compiler.  
• Overall saving in software development cost and maintenance cost.  
  
69. What is meant by general-purpose macro processors?  
The macro processors that are not dependent on any particular  
programming language, but can be used with a variety of different languages are known as general purpose macro processors.  
Eg. The ELENA macro processor.

70. What are the important factors considered while designing general purpose macro processors?  
• comments  
• grouping of statements  
• tokens  
• syntax used for macro definitions  
  
71. How the nested macro calls are executed?  
The execution of nested macro call follows the LIFO rule. In case of nested macro calls the expansion of the latest macro call is completed first.

72. Mention the tasks involved in macro expansion.  
• identify the macro calls in the program  
• the values of formal parameters are identified  
• maintain the values of expansion time variables declared in a macro  
• expansion time control flow is organized  
• determining the values of sequencing symbols  
• expansion of a model statement is performed  
  
73. How to design the pass structure of a macro assembler?  
To design the structure of macro-assembler, the functions of macro  
pre-processor and the conventional assembler are merged. After merging, the functions are structured into passes of the macro assembler.

**PROGRAMS ON MASM**

**Commands to be followed in DOS box to run MASM programs**

**mount c c:\masm**

**edit pgmname.asm**

**masm pgmname.asm**

**link pgmname.obj**

**debug pgmname.exe**

**Pgm.No.18**

**BASIC ARITHMETIC OPERATIONS (16 bit and 32 bit)**

**AIM**

Write a program to perform basic arithmetic operations and logical operations

**16 BIT ADDITION**

**ALGORITHM**

* Step I : Initialize the data segment.
* Step II : Get the first number in AX register.
* Step III : Get the second number in BX register.
* Step IV : Add the two numbers.
* Step V : Display the result.

**PROGRAM**

DATA SEGMENT

N1 DW 1731H

N2 DW 9212H

N3 DW ?

DATA ENDS

CODE SEGMENT

ASSUME CS :CODE;DS:DATA

START:

MOV AX,DATA

MOV DS,AX

XOR AX,AX

MOV BX,AX

MOV AX,N1

ADD AX,N2

MOV N3,AX

JNC STOP

INC BX

STOP:

MOV CX,AX

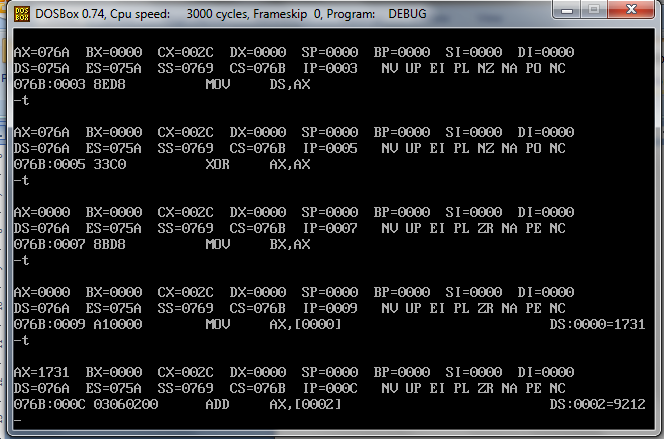
MOV AH,4CH

INT 21H

CODE ENDS

END START

**OUTPUT**

****

**32 BIT ADDITION**

* **ALGORITHM**Step II : Load the LSB of first number into AX register.
* Step III : Load the MSB of first number into BX register.
* Step IV : Load the LSB of the second number into CX register.
* Step V : Load the MSB of the second number into DX register.
* Step VI : Add the LSBs of two number.

**PROGRAM**

DATA SEGMENT

LIST DD 12121212H,12121212H

N3 DW ?

N4 DW ?

DATA ENDS

CODE SEGMENT

ASSUME CS :CODE;DS:DATA

START:

MOV AX,DATA

MOV DS,AX

XOR AX,AX

MOV CL,AL

MOV AX,[SI]

ADD AX,[SI+4]

MOV BX,AX

MOV N3,BX

MOV AX,[SI+2]

ADD AX,[SI+6]

MOV DX,AX

MOV N4,DX

JNC STOP

INC CL

STOP:

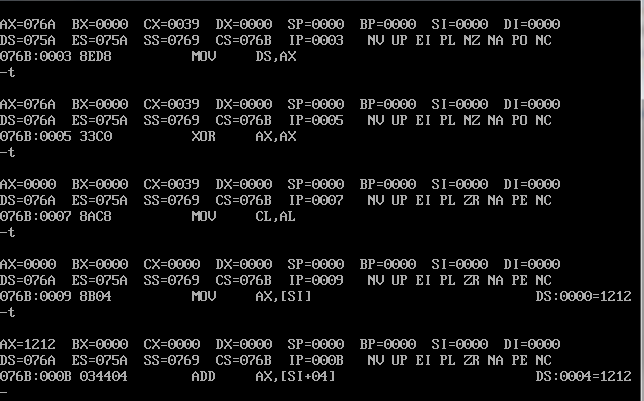
MOV AX,4CH

INT 21H

CODE ENDS

END START

**OUTPUT**

****

**16 BIT SUBTRACTION**

**ALGORITHM**

Step I      :    Initialize the data segment.

Step II     :    Get the first number in AX register.

Step III    :    Get the second number in BX register.

Step IV    :    Add the two numbers.

Step V     :    Display the result.

Step VI    :    Stop

**PROGRAM**

DATA SEGMENT

N1 DW 8888H

N2 DW 4444H

N3 DW ?

DATA ENDS

CODE SEGMENT

ASSUME CS :CODE;DS:DATA

START:

MOV AX,DATA

MOV DS,AX

XOR AX,AX

MOV BX,AX

MOV AX,N1

SUB AX,N2

MOV N3,AX

JNC STOP

INC BX

STOP:

MOV CX,AX

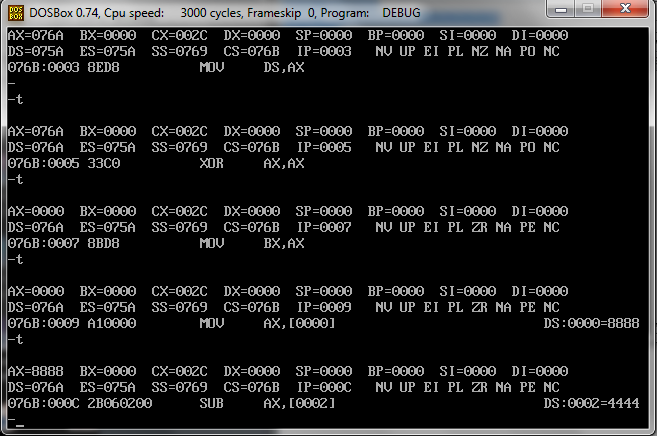
MOV AH,4CH

INT 21H

CODE ENDS

END START

**OUTPUT**

****

**32 BIT SUBTRACTION**

**ALGORITHM**

* Step I : Initialize the data segment.
* Step II : Load the LSB of first number into AX register.
* Step III : Load the MSB of first number into BX register.
* Step IV : Load the LSB of the second number into CX register.
* Step V : Load the MSB of the second number into DX register.

**PROGRAM**

DATA SEGMENT

LIST DD 12121212H,12121212H

N3 DW ?

N4 DW ?

DATA ENDS

CODE SEGMENT

ASSUME CS :CODE;DS:DATA

START:

MOV AX,DATA

MOV DS,AX

XOR AX,AX

MOV CL,AL

MOV AX,[SI]

ADD AX,[SI+4]

MOV BX,AX

MOV N3,BX

MOV AX,[SI+2]

ADD AX,[SI+6]

MOV DX,AX

MOV N4,DX

JNC STOP

INC CL

STOP:

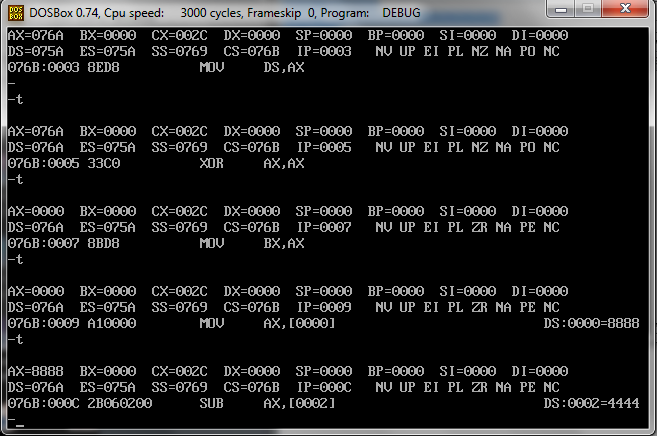
MOV AX,4CH

INT 21H

CODE ENDS

END START

OUTPUT

****

**16 BIT MULTIPLICATION**

**ALGORITHM**

Step I       :    Initialize the data segment.

Step II     :    Get the first number in AX register.

Step III   :    Get the second number in BX register.

Step IV    :    Multiply the two 16 bit numbers.

Step V     :    Display the result.

Step VI    :    Stop

**PROGRAM**

DATA SEGMENT

N1 DW 8888H

N2 DW 4444H

N3 DW ?

DATA ENDS

CODE SEGMENT

ASSUME CS :CODE;DS:DATA

START:

MOV AX,4343

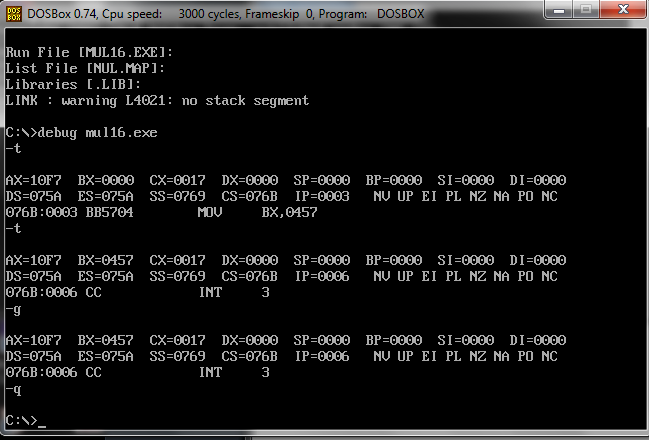
MOV BX,1111

INT 3

CODE ENDS

END START

**OUTPUT**

****

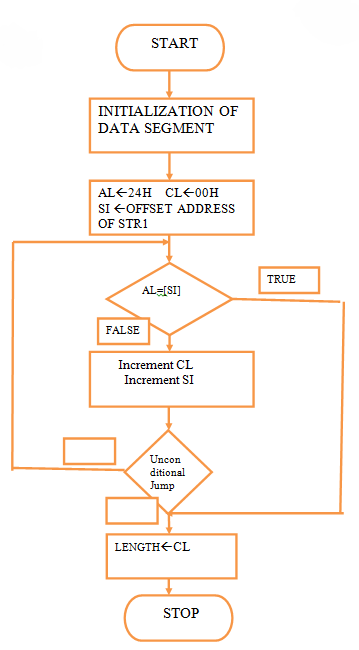
**Pgm.No.19**

**STRING DISPLAY**

**AIM**

Write a program to display a given string

**FLOWCHART**

****

**PROGRAM**

DATA SEGMENT

MSG1 DB "HELLO WORLD$"

DATA ENDS

ASSUME CS:CODE; DS:DATA

CODE SEGMENT

START:

MOV AX,DATA

MOV DS,AX

MOV DX,OFFSET MSG1

MOV AH,09H

INT 21H

MOV AH,4CH

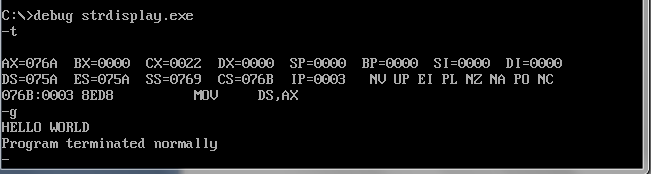
MOV AL,00H

INT 21H

CODE ENDS

END START

**OUTPUT**

****

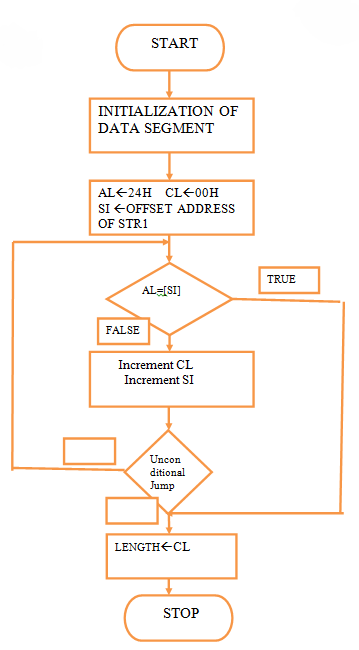
**Pgm.No.20**

**STRING CONCATENATENATION**

**AIM**

Write a program to concatenate two strings

**FLOWCHART**

****

**PROGRAM**

DATA SEGMENT

MSG1 DB "HELLO$"

MSG2 DB "WORLD$"

DATA ENDS

ASSUME CS:CODE; DS:DATA

CODE SEGMENT

START:

MOV AX,DATA

MOV DS,AX

MOV DX,OFFSET MSG1

MOV AH,09H

INT 21H

MOV DX,OFFSET MSG2

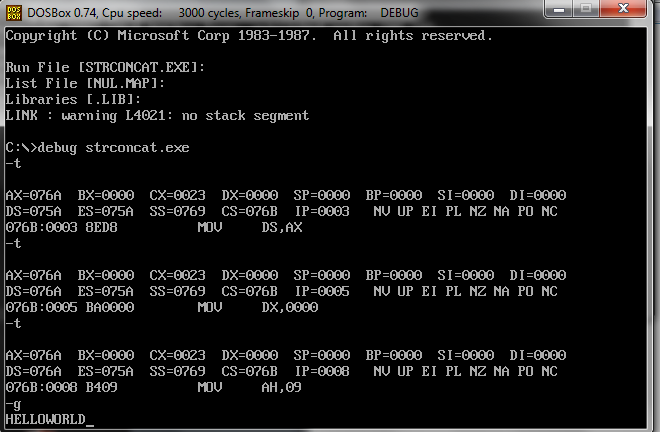
MOV AH,09H

INT 21H

CODE ENDS

END START

**OUTPUT**

****

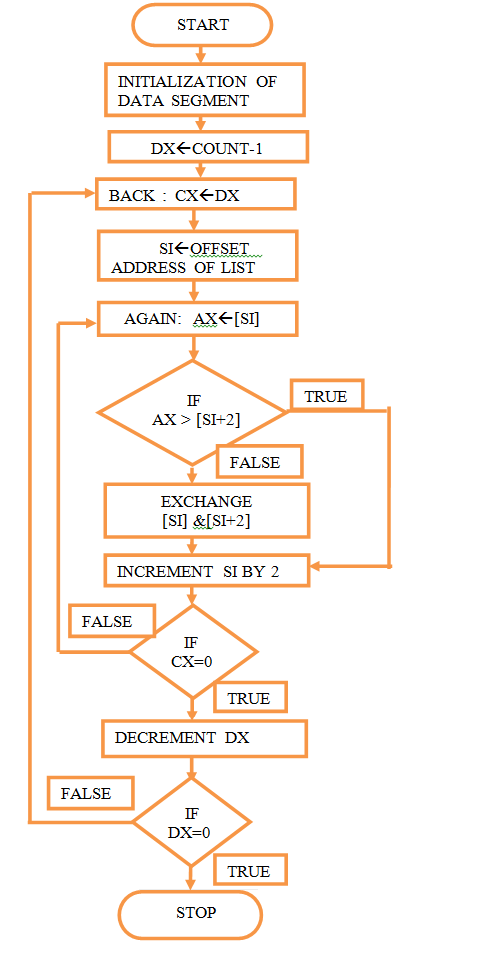
**Pgm.No . 21**

**SORTING**

**AIM**

Write a program to perform sorting

FLOWCHART



**PROGRAM**

DATA SEGMENT

STRING1 DB 99H,12H,56H,45H,36H

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START: MOV AX,DATA

MOV DS,AX

MOV CH,04H

UP2: MOV CL,04H

LEA SI,STRING1

UP1:MOV AL,[SI]

MOV BL,[SI+1]

CMP AL,BL

JNC DOWN

MOV DL,[SI+1]

XCHG [SI],DL

MOV [SI+1],DL

DOWN: INC SI

DEC CL

JNZ UP1

DEC CH

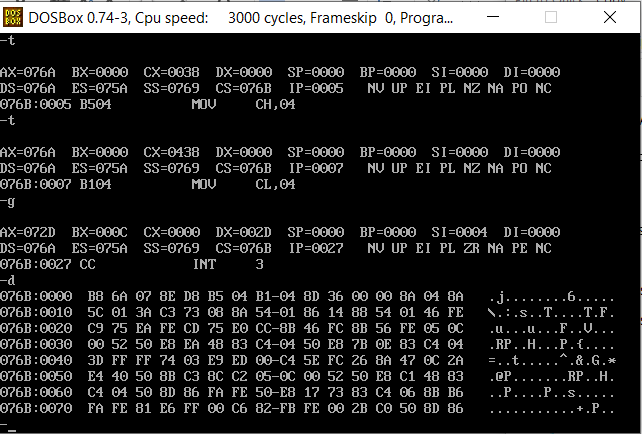
JNZ UP2

INT 3

CODE ENDS

END START

**OUTPUT**

****

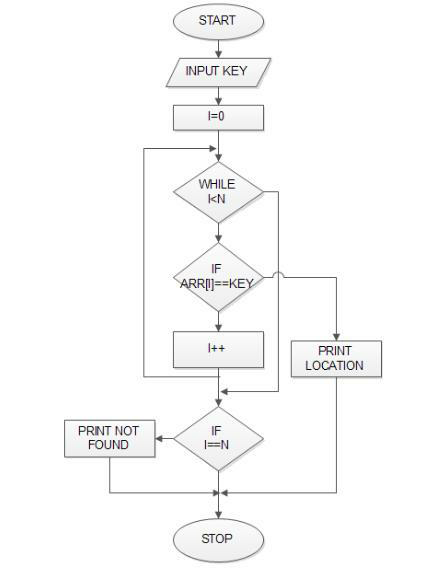
**Pgm.No . 22**

**SEARCHING**

**AIM**

Write a program to perform searching

ALGORITHM



**PROGRAM**

DATA SEGMENT

STRING1 DB 11H,22H,33H,44H,55H

MSG1 DB "FOUND$"

MSG2 DB "NOT FOUND$"

SE DB 33H

DATA ENDS

PRINT MACRO MSG

MOV AH, 09H

LEA DX, MSG

INT 21H

INT 3

ENDM

CODE SEGMENT

ASSUME CS:CODE, DS:DATA

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

PRINT MSG2

JMP END1

FO:

PRINT MSG1

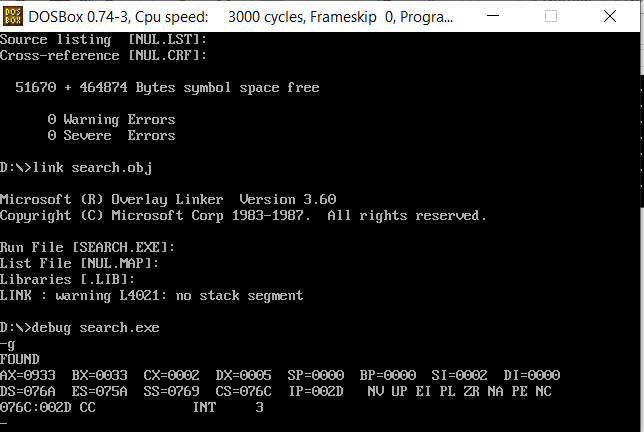
END1:

INT 3

CODE ENDS

END START

**OUTPUT**



**PROGRAMS ON 8086 TRAINER KIT**

**Pgm.No.23**

**ADDITION-16 BIT**

**AIM**

Write a program to perform addition of two 16 bit numbers

**ALGORITHM**

1. Load the lower part of first number in B register
2. Load the lower part of second number in A (accumulator)
3. Add both the numbers and store
4. Load the higher part of first number in B register
5. Load the higher part of second number in A (accumulator)
6. Add both the numbers with carry from the lower bytes (if any) and store at the next location

**PROGRAM**

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

**INPUT**

| ADDRESS | VALUE |
| --- | --- |
| 0500 | B5 |
| 0501 | 7A |
| 0550 | 2A |
| 0551 | E6 |

**OUTPUT**

| ADDRESS | VALUE |
| --- | --- |
| 0600 | DF |
| 0601 | 5F |
| 0602 | 01 |

**SUBTRACTION-16 BIT**

**AIM**

Write a program to perform subtraction of two 16 bit numbers

**ALGORITHM**

1. Load 0000H into CX register (for borrow)
2. Load the data into AX(accumulator) from memory 3000
3. Load the data into BX register from memory 3002
4. Subtract BX with Accumulator AX
5. Jump if no borrow
6. Increment CX by 1
7. Move data from AX(accumulator) to memory 3004
8. Move data from CX register to memory 3006
9. Stop

**PROGRAM**

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

**INPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 0700 | 18 |
| 0701 | 08 |
| 0800 | 40 |
| 0801 | 10 |

**OUTPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 0900 | D8 |
| 0901 | F7 |

**MULTIPLICATION-16 BIT**

**AIM**

Write a program to perform multiplication of two 16 bit numbers

**ALGORITHM**

* First load the data into AX(accumulator) from memory 3000.
* Load the data into BX register from memory 3002.
* Multiply BX with Accumulator AX.
* Move data from AX(accumulator) to memory.
* Move data from DX to AX.
* Move data from AX(accumulator) to memory.
* Stop.

**PROGRAM**

| **ADDRESS** | **MNEMONICS** |
| --- | --- |
| 2000 | MOV AX, [3000] |
| 2004 | MOV BX, [3002] |
| 2008 | MUL BX |
| 200A | MOV [3004], AX |
| 200E | MOV AX, DX |
| 2010 | MOV [3006], AX |
| 2014 | MOV AX, [3000] |
| 2000 | HLT |

**INPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 3003 | 07 |
| 3002 | 08 |
| 3001 | 04 |
| 3000 | 03 |

**OUTPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 3007 | 00 |
| 3006 | 1C |
| 3005 | 35 |
| 3004 | 18 |

**Pgm.No.24**

**SORTING-ASCENDING**

**AIM**

Write a program to perform sorting

**ALGORITHM**

1. Load data from offset 500 to register CL (for count).
2. Travel from starting memory location to last and compare two numbers if first number is greater than second number then swap them.
3. First pass fix the position for last number.
4. Decrease the count by 1.
5. Again travel from starting memory location to (last-1, by help of count) and compare two numbers if first number is greater than second number then swap them.
6. Second pass fix the position for last two numbers.
7. Repeated.

**PROGRAM**

| **ADDRESS** | **MNEMONICS** |
| --- | --- |
| 400 | MOV SI, 500 |
| 403 | MOV CL, [SI] |
| 405 | DEC CL |
| 407 | MOV SI, 500 |
| 40A | MOV CH, [SI] |
| 40C | DEC CH |
| 40E | INC SI |
| 40F | MOV AL, [SI] |
| 411 | INC SI |
| 412 | CMP AL, [SI] |
| 414 | JC 41C |
| 416 | XCHG AL, [SI] |
| 418 | DEC SI |
| 419 | XCHG AL, [SI] |
| 41B | INC SI |
| 41C | DEC CH |
| 41E | JNZ 40F |
| 420 | DEC CL |
| 422 | JNZ 407 |
| 424 | HLT |

**INPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 500 | 04 |
| 501 | F9 |
| 502 | F2 |
| 503 | 39 |
| 504 | 05 |

**OUTPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 501 | 05 |
| 502 | 39 |
| 503 | F2 |
| 504 | F9 |

**CODE CONVERSIONS**

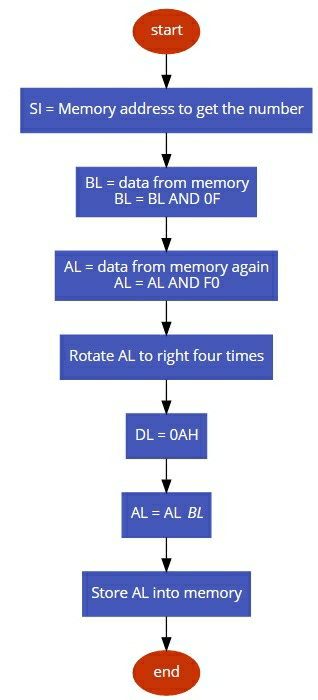
**Pgm.No. 25**

**BCD TO HEXADECIMAL**

**AIM**

Write a program to perform conversion of 8 bit BCD number into hexadecimal number

**FLOWCHART**

****

**PROGRAM**

| **ADDRESS** | **MNEMONICS** |
| --- | --- |
| 400 | MOV SI, 500 |
| 403 | MOV DI, 600 |
| 406 | MOV BL, [SI] |
| 408 | AND BL, 0F |
| 040A | MOV AL, [SI] |
| 040C | AND AL, F0 |
| 040E | MOV CL, 04 |
| 410 | ROR AL, CL |
| 412 | MOV DL, 0A |
| 414 | MUL DL |
| 416 | ADD AL, BL |
| 418 | MOV [DI], AL |
| 041A | HLT |

**INPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 500 | 25 |

**OUTPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 600 | 19 |

**Pgm.No.**

**ASCII TO BCD**

**AIM**

Write a program to perform conversion of ASCII(in hex) value of number to its BCD(decimal) number

ALORITHM

1. Load the value from memory into register AL
2. Then perform and operation on register AL with 0F
3. Move the result value from register AL to memory
4. Terminate the program

| **ASCII (in Hex)** | **30** | **31** | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **BCD** | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 |

**PROGRAM**

| **ADDRESS** | **MNEMONICS** |
| --- | --- |
| 400 | MOV AL,[2050] |
| 403 | AND AL,0F |
| 405 | MOV [3050],AL |
| 408 | HLT |

**INPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 2050 | 39 |

**OUTPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 3050 | 09 |

**BCD TO ASCII**

**AIM**

Write a program to perform conversion of 8 bit BCD number to ASCII code

**ALGORITHM**

* Load contents of memory location 2000 in register AL.
* Copy contents of register AL in register AH.
* Perform AND operation on register AL with 0F.
* Assign 04 to CL Register.
* Shift the contents of AH by executing SHR instruction using CL.
* Perform OR operation on register AX with 3030.

**PROGRAM**

| **ADDRESS** | **MNEMONICS** |
| --- | --- |
| 400 | MOV AL, [2000] |
| 404 | MOV AH, AL |
| 406 | AND AL, 0F |
| 408 | MOV CL, 04 |
| 40A | SHR AH, CL |
| 40C | OR AX, 3030 |
| 40F | MOV [3000], AX |
| 413 | HLT |

**INPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 2000 | 98 |

**OUTPUT**

| **ADDRESS** | **VALUE** |
| --- | --- |
| 3000 | 38 |
| 3001 | 39 |

**8051 TRAINER KIT PROGRAMS**

**Pgm.No. 25**

**BASIC ARITHEMATIC AND LOGICAL OPERATIONS**

**AIM**

Write a program to perform basic arithematic operations on 8051 trainer kit

**ALGORITHM**

* Initialize Ports P0 and P1 as input ports.
* Initialize Ports P2 and P3 as output ports.
* Initialize the R1 register.
* Move the contents from Port 0 to B register.
* Move the contents from Port 1 to A register.
* Add contents in A and B.
* If carry is present increment R1.
* Move contents in R1 to Port 2.
* Move the sum in step 6 to Port 3.

**PROGRAM**

**Addition**

| **ADDRESS** | **MNEMONICS** |
| --- | --- |
| 3000 | MOV R0,#20H;set source address 20H to R0 |
| 3002 | MOV A,@R0; take the value from source to register A |
| 3004 | MOV R1,#30H;set destination address 30H to R1 |
| 3006 | MOV R5,A; Move the value from A to R5 |
| 3008 | MOV R4,#00H; Clear register R4 to store carry |
| 300A | INC R0; Point to the next location |
| 300B | MOV A,@R0; take the value from source to register A |
| 300D | ADD A,R5;Add R5 with A and store to register A |
| 300F | JNC SAVE |
| 3010 | INC R4; Increment R4 to get carry |
| 3011 | MOV B,R4;Get carry to register B |
| 3012 | MOV @R1,B; Store the carry first |
| 3013 | INC R1; Increase R1 to point to the next address |
| 3014 | SAVE:   MOV@R1,A;Store the result |
| 3016 | HALT:   SJMP HALT ;Stop the program |

**Output**

| **Address** | **Value** |
| --- | --- |
| 21H | D8H |
|  |  |
| 30H | 00H |
| 31H | 00H |

**Subtraction**

**ALGORITHM**

* 1. set source address 20H to R0
  2. set destination address 30H to R1
  3. MOV R1, #30H
     1. take the value from source to register A
     2. Move the value from A to R5
     3. Move the value from A to R5
     4. Clear register R4 to store borrow
     5. Point to the next location
     6. take the value from source to register A
     7. store second byte
     8. get back the first operand
     9. Subtract R3 from A
     10. If no carry save result
     11. Increment R4 to get borrow
     12. Store the borrow first
     13. Increase R1 to point to the next address
     14. Store the result
     15. Stop the program

**PROGRAM**

| **ADDRESS** | **MNEMONICS** |
| --- | --- |
| 3000 | MOV R0,#20H;set source address 20H to R0 |
| 3002 | MOV R1,#30H;set destination address 30H to R1 |
| 3004 | MOV A,@R0;take the value from source to register A |
| 3006 | MOV R5,A; Move the value from A to R5 |
| 3008 | MOV R4,#00H; Clear register R4 to store borrow |
| 300A | INC R0; Point to the next location |
| 300B | MOV A,@R0; take the value from source to register A |
| 300D | MOV R3,A; store second byte |
| 300F | MOV A,R5;get back the first operand |
| 3010 | SUBB A,R3; Subtract R3 from A |
| 3011 | JNC SAVE |
| 3012 | INC R4; Increment R4 to get borrow |
| 3013 | MOV B,R4;Get borrow to register B |
| 3014 | MOV @R1,B; Store the borrow first |
| 3015 | INC R1; Increase R1 to point to the next address |
| 3016 | SAVE:  MOV@R1,A; Store the result |
| 3017 | HALT:  SJMP HALT ;Stop the program |

## Output

| **Address** | **Value** |
| --- | --- |
| 20H | 73H |
| 21H | BDH |
| 30H | 01H |
| 31H | B6H |

**Multiplication**

ALGORITHM

* 1. set source address 20H to R0
  2. set destination address 30H to R1
  3. take the first operand from source to register A
  4. Point to the next location
  5. take the second operand from source to register B
  6. Multiply A and B
  7. ; Store higher order byte to 30H
  8. Increase R1 to point to the next location
  9. Store lower order byte to 31H
  10. Stop the program

**PROGRAM**

| **ADDRESS** | **MNEMONICS** |
| --- | --- |
| 3000 | MOV R0, #20H;set source address 20H to R0 |
| 3002 | MOV R1, #30H;set destination address 30H to R1 |
| 3004 | MOV A, @R0;take the first operand from source to register A |
| 3006 | INC R0; Point to the next location |
| 3008 | MOV B,@R0;take the second operand from source to register B |
| 300A | MUL AB ;Multiply A and B |
| 300B | MOV @R1, B; Store higher order byte to 30H |
| 300D | INC R1; Increase R1 to point to the next location |
| 300E | MOV @R1, A;Store lower order byte to 31H |
| 3010 | HALT:   SJMP HALT ; Stop the program |

**Output**

| **Address** | **Value** |
| --- | --- |
| 20H | FFH |
| 21H | FFH |
| 30H | FEH |
| 31H | 01H |

**CODE CONVERSION**

**HEX to BCD**

**ALGORITHM**

* 1. Initialize the address of the data
  2. Get the data from an address, which is stored in R0
  3. Store the content of A into R2
  4. Clear the Carry Flag
  5. Subtract 0AH from A
  6. When a carry is present, A is numeric
  7. Add41H for Alphabet
  8. Jump to store the value
  9. Copy R2 to A
  10. Add 30H with A to get ASCII
  11. Point the destination location
  12. Store A content to the memory location pointed by R0
  13. Stop program

PROGRAM

| **ADDRESS** | **MNEMONICS** |
| --- | --- |
| 3000 | MOV R0,#20H; Initialize the address of the data |
| 3002 | MOV A,@R0; Get the data from an address, which is stored in R0 |
| 3004 | MOV R2,A;Store the content of A into R2 |
| 3006 | CLRC; Clear the Carry Flag |
| 3007 | SUBB A,#0AH;Subtract 0AH from A |
| 3009 | JCNUM ; When a carry is present, A is numeric |
| 300A | ADD A,#41H;Add41H for Alphabet |
| 300C | SJMP STORE; Jump to store the value |
| 300F | NUM:    MOV A,R2; Copy R2 to A |
| 3011 | ADD A,#30H; Add 30H with A to get ASCII |
| 3013 | STORE:  MOVR0,#30H; Point the destination location |
| 3015 | MOV @R0,A; Store A content to the memory location pointed by R0 |
| 3017 | HALT:   SJMPHALT |

**Output**

| 20H | 0EH |  |
| --- | --- | --- |
| 21H |  |  |

**HEX to ASCII**

**ALGORITHM**

1. move the numbers to be converted
2. compare the number with A
3. If the number is greater than A jump to label2
4. If the number is lesser than A add with 30H
5. STOP
6. END  
     
     
   **PROGRAM**

| **ADDRESS** | **MNEMONICS** |
| --- | --- |
| 3000 | MAIN:  MOV R0,#01H; move the numbers to be converted |
| 3002 | MAIN:  MOV R0,#01H ; move the numbers to be converted |
| 3004 | MOV A,R0 |
| 3006 | CJNE A,#0AH,LABEL1  ; compare the number with A |
| 3007 | LABEL1:JNC LABEL2 ;If the number is greater than A jump to label2 |
| 3009 | ADD A,#30H ; If the number is lesser than A add with 30H |
| 300A | SJMP STOP |
| 3011 | LABEL2:ADD A,#37H   ;If the number is greater than A add with 37H |
| 3013 | STOP: |
| 3015 | END |

**VivaQuestions and Answers**

1.What is a Microprocessor?

Microprocessor is a CPU fabricated on a single chip, program-controlled device, which fetches the instructions from memory, decodes and executes the instructions.

2. What is Instruction Set?

 It is the set of the instructions that the Microprocessor can execute.

3. What is Bandwidth ?

 The number of bits processed by the processor in a single instruction.

4. What is Clock Speed ?

Clock speed is measured in the MHz and it determines that how many instructions a processor can processed. The speed of the microprocessor is measured in the MHz or GHz.

5. What are the features of Intel 8086 ?

Features:

·              Released by Intel in 1978

·              Produced from 1978 to 1990s

·              A 16-bit microprocessor chip.

·              Max. CPU clock rate:5 MHz to 10 MHz

·              Instruction set:  x86-16

·              Package: 40 pin DIP

·              16-bit Arithmetic Logic Unit

·              16-bit data bus  (8088 has 8-bit data bus)

·              20-bit address bus - 220 = 1,048,576 = 1 meg

·              The address refers to a byte in memory.

·               In the 8088, these bytes come in on the 8-bit data bus.  In the 8086, bytes at even addresses come in on the low half of the data bus (bits 0-7) and bytes at odd addresses come in on the upper  half of the data bus (bits 8-15).

·              The 8086 can read a 16-bit word at an even address in one operation and at an odd address in two operations.  The 8088 needs two operations in either case.

·              The least significant byte of a word on an 8086 family microprocessor is at the  lower address.

6. What is Logical Address:?

•         A memory address on the 8086 consists of two numbers, usually written in hexadecimal and separated by a colon, representing the *segment* and the *offset*. This combination of segment and offset is referred to as a *logical address*

•         Logical address=segment: offset

7. What is The Effective Address:

•         In general, memory accesses take the form of the following example:

•         Mov ax, [*baseReg* + *indexReg* + *constant*]

•         This example copies a word sized value into the register AX.

•         Combined, the three parameters in brackets determine what is called the *effective address*, which is simply the offset referenced by the instruction

8. What is Physical Address?

Physical memory address pointed by SEGMENT:OFFSET pair is calculated as:

Physical address = (<Segment Addr> \* 10) + <Offset Addr>

9.What are the flags in 8086?

In 8086 Carry flag, Parity flag, Auxiliary carry flag, Zero flag, Overflow flag, Trace flag, Interrupt flag, Direction flag, and Sign flag.

10.Why crystal is a preferred clock source?

Because of high stability, large Q (Quality Factor) & the frequency that doesn’t drift with aging. Crystal is used as a clock source most of the times.

11.What is Tri-state logic?

Three Logic Levels are used and they are High, Low, High impedance state. The high and low are normal logic levels & high impedance state is electrical open circuit conditions. Tri-state logic has a third line called enable line.

12.What happens when HLT instruction is executed in processor?

The Micro Processor enters into Halt-State and the buses are tri-stated.

13.What is Program counter?

Program counter holds the address of either the first byte of the next instruction to be fetched for execution or the address of the next byte of a multi byte instruction, which has not been completely fetched. In both the cases it gets incremented automatically one by one as the instruction bytes get fetched. Also Program register keeps the address of the next instruction.

14.What is 1st / 2nd / 3rd / 4th generation processor?

The processor made of PMOS / NMOS / HMOS / HCMOS technology is called 1st / 2nd / 3rd / 4th generation processor, and it is made up of 4 / 8 / 16 / 32 bits.

15.Name the processor lines of two major manufacturers?

High-end: Intel - Pentium (II, III, 4), AMD - Athlon. Low-end: Intel - Celeron, AMD - Duron. 64-bit: Intel - Itanium 2, AMD - Opteron.

16.How many bit combinations are there in a byte?

Byte contains 8 combinations of bits.

17.Have you studied buses? What types?

There are three types of buses.

Address bus: This is used to carry the Address to the memory to fetch either Instruction or Data.

Data bus : This is used to carry the Data from the memory.

Control bus : This is used to carry the Control signals like RD/WR, Select etc.

18.What is the Maximum clock frequency in 8086?

5 Mhz is the Maximum clock frequency in 8086.

19.What is meant by Maskable interrupts?

An interrupt that can be turned off by the programmer is known as Maskable interrupt.

20.What is Non-Maskable interrupts?

An interrupt which can be never be turned off (ie. disabled) is known as Non-Maskable interrupt

21.What are the different functional units in 8086?

Bus Interface Unit and Execution unit, are the two different functional units in 8086.

22.What are the various segment registers in 8086?

Code, Data, Stack, Extra Segment registers in 8086.

23.What does EU do?

Execution Unit receives program instruction codes and data from BIU, executes these instructions and store the result in general registers.

24.Which Stack is used in 8086? k is used in 8086?

FIFO (First In First Out) stack is used in 8086.In this type of Stack the first stored information is retrieved first.

25.What are the flags in 8086?

In 8086 Carry flag, Parity flag, Auxiliary carry flag, Zero flag, Overflow flag, Trace flag, Interrupt flag, Direction flag, and Sign flag.

26.What is SIM and RIM instructions?

SIM is Set Interrupt Mask. Used to mask the hardware interrupts.

RIM is Read Interrupt Mask. Used to check whether the interrupt is Masked or not.

27.What are the different types of Addressing Modes?

A:- There are 12 different types of Addressing Modes.They are:-

<1> Immediate:-The Immediate data is a part of instruction, and appears in the form of successive bytes.

<2> Direct:-A 16-bit memory address(offset) is directly specified in the instruction as a part of it.

<3> Register:-Data is stored in a register and it is referred using the particular register (except IP).

<4> Register Indirect:-The address of the memory location which contains data or operand is determined in an indirect way.

<5> Indexed:-offset of the operand is stored in one of the index registers.

<6> Register Relative:-The data is available at an eefective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX,BP,SI and DI in the default (either DS or ES) segment.

<7> Based Indexed:-The effective address of the data is formed,in this addressing mode,by adding content of a base register to the content of an index register.

<8> Relative Based Indexed:- The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any one of the base registers and any one of the index registers,in the default segment.

<9> Intrasegment Direct Mode:-In this mode,the address to which the control is to bve transferred lies in the segment in which the control transfer instruction lies and appears directly in the instruction as an immediate displacement value.

<10> Intrasegment Indirect Mode:-In this mode,the displacement to which the control is to be transferred,is in the same segment in whgich the control transfer instruction lies,but it is passed to the instruction indirectly.

<11> Intersegment Direct:-In this mode,the address to which the control is to be transferred is in a different segment.

<12> Intersegment Indirect:-In this mode,the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly sequentially.

28.What are the General Data Registers & their uses?

A:- The Registers AX,BX,CX,DX are the general Purpose 16-bit registers.AX register as 16-bit accumulator.BX register is used as an offset Storage.CX register is used as default or implied counter.Dx register is used as an implicit operand or destination in case of a few instructions.

29.What are Segment Registers & their uses?

A:-There are 4 Segment Registers Code Segment(CS),Data Segment(DS),Extra Segment(ES) & Stack Segment(SS) registers.CS is used for addressing memory locationin code.DS is used to points the data.ES refers to a segment which is essentially in another data segment.SS is used fopr addressing stack segment of memory.

30.What are Flag registers?

A:-Divided into 2 parts:-Condition code or status flags and machine control flags.

S-Sign Flag:-Is to set when the result of any computation is negative.

Z-Zero Flag:-Is to set if the result of the computation or comparision performed by the previous instruction is zero.

C-Carry Flag:-Is set when there is carry out of MSB in case of addition or a borrow in case of subtraction.

T-Trap Flag:-Is set,the processor enters the single step execution mode.

I-Interrupt Flag:-Is set,the maskable interrupts are recognised by the CPU.

D-Direction Flag:-Is set for autoincrementing or autodecrementing mode in string manipulation instructions.

AC-Auxiliary Carry Flag:-Is set if there is a carry from the lowest nibble during addition or borrow for the lowest nibble.

O-Overflow Flag:-Is setif the result of a signed operation is large enough to be accommodated in a destination register.

32) What are Data Copy/Transfer Instructions?

A:- Mov

Push

Pop

Xchg

In

Out

Xlat

Lea

Lds/Les

Lahf

Sahf

Pushf

Popf

33. What are Machine Control Instructions?

A:- Nop

Hlt

Wait

Lock

34) What are Flag Manipulation Instructions?

A:- Cld

Std

Cli

Sti

35) What are String Instructions?

A:- Rep

MovSB/MovSW

  Cmps

  Scas

  Lods

Stos

37.What is an Interrupts

Def:- An interrupt operation suspends execution of a program so that the system can take special action.The interrupt routine executes and normally returns control to the interrupted procedure, which then resumes execution.BIOS handles Int 00H-1FH, whereas DOS handles INT 20H-3FH.

38. What is an Opcode?

A:-The part of the instruction that specifies the operation to be performed is called the Operation code or Op code.

39.What is an Operand?

A:-The data on which the operation is to be performed is called as an Operand.

40.Explain the difference between a JMP and CALL instruction?

A:-A JMP instruction permantely changes the program counter.

A CALL instruction leaves information on the stack so that the original program execution sequence can be resumed.

41.What is meant by Polling?

A:- Polling or device Polling is a process which idenfies the device that has interrupted the microprocessor.

42.What is meant by Interrupt?

A:-Interrupt is an external signal that causes a microprocessor to jump to a specific subroutine.

43.What is an Instruction?

A:-An instruction is a binary pattern enetered through an input device to command the microprocessor to perform that specific function.

44.What is Microcontroller and Microcomputer?

A:- Microcontroller is a device that includes microprocessor:memory and I/O signal lines on a single chip,fabricated using VLSI technology.

Microcomputer is a computer that is designed using microprocessor as its CPU.It includes microprocessor,memory and I/O.

45.What is Assembler?

A:-The assembler translates the assembly language program text which is given as input to the assembler to their binary equivalents known as object code.

The time required to translate the assembly code to object code is called access time.The assembler checks for syntax errors&displays them before giving the object code.

49.What is the use of HLDA?

A:-HLDA is the acknowledgment signal for HOLD. It indicates whether the HOLD signal is received or not.

HOLD and HLDA are used as the control signals for DMA operations.

50. Explain about "LEA"?

A:-LEA(Load Effective Address) is used for initializing a register with an offset address.

A common use for LEA is to intialize an offset in BX, DI or SI for indexing an address in memory.

An equivalent operation to LEA is MOV with the OFFSET operator, which generates slightly shorter machine code.

51. Difference between "Shift" and "Rotate".

A:-Shift and Rotate commands are used to convert a number to another form where some bits are shifted or rotated.

A rotate instruction is a closed loop instruction.That is,the data moved out at one end is put back in at the other end.

The shift instruction loses the data that is moved out of the last bit locations.

Basic difference between shift and rotate is shift command makes "fall of " bits at the end of the register.

Where rotate command makes "wrap around" at the end of the register.

53.Difference between JMP and JNC?

A:-JMP is Unconditional Branch.

JNC is Conditional Branch.

55.What are the 4 Segments?

A:-Code Segment Register {CS}

Data Segment Register {DS}

Extra Segment Register {ES}

Stack Segment Register{SS}

56.What is the main use of ready pin?

A:-READY is used by the microprocessor to check whether a peripheral is ready to accept or transfer data.

A peripheral may be a LCD display or analog to digital converter or any other.

These peripherals are connected to microprocessor using the READY pin.

If READY is high then the periphery is ready for data transfer. If not the microprocessor waits until READY goes high.

57.Explain about Direction Flag?

A:-This is used by string manipulation instructions. If this flag bit is 0 , the string is processed beginning from the lowest to the highest address,i.e.,.Autoincrement mode. Otherwise,the string is processed from the highest towards the lowest address,i.e.,.Autodecrementing mode.

58. What is the function of microprocessor in a system?

The microprocessor is the master in the system, which controls all the activity of the system. It issues address and control signals and fetches the instruction and data from memory. Then it executes the instruction to take appropriate action.

59. What are the modes in which 8086 can operate?

The 8086 can operate in two modes and they are minimum (or uniprocessor) mode and maximum ( or multiprocessor) mode.

60. What is the data and address size in 8086?

The 8086 can operate on either 8-bit or 16-bit data. The 8086 uses 20 bit address to access memory and 16-bit address to access 1/0 devices.

61. How clock signal is generated in 8086? What is the maximum internal clock frequency of 8086?

The 8086 does not have on-chip clock generation circuit. Hence the clock generator chip, 8284 is connected to the CLK pin of8086. The clock signal supplied by 8284 is divided by three for internal use. The maximum internal clock frequency of8086 is 5MHz.

62. What is pipelined architecture?

In pipelined architecture the processor will have number of functional units and the execution time of functional units are overlapped. Each functional unit works independently most of the time.

63. What are the functional units available in 8086 architecture?

The bus interface unit and execution unit are the two functional units available in 8086 architecture.

64. List the segment registers of 8086.

The segment registers of 8086 are Code segment, Data segment, Stack segment and Extra segment registers.

65. Define machine cycle.

Machine cycle is defined as the time required to complete one operation of accessing memory, I/O, or acknowledging an external request. This cycle may consist of three to six T-states.

66. Define T-State.

T-State is defined as one subdivision of the operation performed in one clock period. These subdivisions are internal states synchronized with the system clock, and each T-State is precisely equal to one clock period.

67. List the components of microprocessor (single board microcomputer) based system

The microprocessor based system consist of microprocessor as CPU, semiconductor memories like EPROM and RAM, input device, output device and interfacing devices.

68. Why interfacing is needed for 1/0 devices?

Generally I/O devices are slow devices. Therefore the speed of I/O devices does not match with the speed of microprocessor. And so an interface is provided between system bus and I/O devices.

69. What is the difference between CPU bus and system bus?

The CPU bus has multiplexed lines but the system bus has separate lines for each signal. (The multiplexed CPU lines are demultiplexed by the CPU interface circuit to form system bus).

70.What is a port?

The port is a buffered I/O, which is used to hold the data transmitted from the microprocessor to I/O device or vice-versa.

71.Give some examples of port devices used in 8085 microprocessor based system?

The various INTEL I/O port devices used in 8085 microprocessor based system are 8212, 8155, 8156, 8255, 8355 and 8755.