CS2308 SYSTEM SOFTWARE LAB

Prepared By,

R.Uma,
Lecturer, CSE

CS2308 SYSTEM SOFTWARE LAB

L T P C

0 0 3 2

(Using C)

1. Implement a symbol table with functions to create, insert, modify, search, and display.
2. Implement pass one of a two pass assembler.
3. Implement pass two of a two pass assembler.
4. Implement a single pass assembler.
5. Implement a two pass macro processor
6. Implement a single pass macro processor.
7. Implement an absolute loader.
8. Implement a relocating loader.
9. Implement pass one of a direct-linking loader.
10. Implement pass two of a direct-linking loader.
11. Implement a simple text editor with features like insertion / deletion of a character, word, and sentence.
12. Implement a symbol table with suitable hashing

(For loader exercises, output the snap shot of the main memory as it would be, after the loading has taken place)

TOTAL=45 PERIODS

Requirement for a batch of 30 students

Description of Equipment - Quantity Required

1. Hardware – Pentium PC Desktops - 30 Nos.

2. Software – Turbo C (Freely download) - Multiuser

LIST OF EXPERIMENTS

S.NO	TITLE	PAGE NO
1	Implement a symbol table with functions to create, insert, modify, search, and display.	
2	Implement pass one of a two pass assembler.	
3	Implement pass two of a two pass assembler.	
4	Implement a single pass assembler.	
5	Implement a two pass macro processor	
6	Implement a single pass macro processor.	
7	Implement an absolute loader.	
8	Implement a relocating loader.	
9	Implement pass one of a direct-linking loader.	
10	Implement pass two of a direct-linking loader.	
11	Implement a simple text editor with features like insertion / deletion of a character, word, and sentence.	
12	Implement a symbol table with suitable hashing	

EXPT NO 1

Implement a symbol table with functions to create, insert, modify, search, and display.

AIM :

To write a C program to implement Symbol Table

Algorithm :

1. Start the program for performing insert, display, delete, search and modify option in symbol table
2. Define the structure of the Symbol Table
3. Enter the choice for performing the operations in the symbol Table
4. If the entered choice is 1, search the symbol table for the symbol to be inserted. If the symbol is already present, it displays "Duplicate Symbol". Else, insert the symbol and the corresponding address in the symbol table.
5. If the entered choice is 2, the symbols present in the symbol table are displayed.
6. If the entered choice is 3, the symbol to be deleted is searched in the symbol table. If it is not found in the symbol table it displays "Label Not found". Else, the symbol is deleted.
7. If the entered choice is 5, the symbol to be modified is searched in the symbol table. The label or address or both can be modified.

Source Code program in c implement symbol table

```
# include <stdio.h>
# include <conio.h>
# include <alloc.h>
# include <string.h>
# define null 0
int size=0;
void insert();
void del();
int search(char lab[]);
void modify();
void display();
struct symbtab
{
char label[10];
int addr;
struct symtab *next;
};
struct symbtab *first,*last;
void main()
```

```
{
int op;
int y;
char la[10];
clrscr();
do
{
printf("\nSYMBOL TABLE IMPLEMENTATION\n");
printf("1. INSERT\n");
printf("2. DISPLAY\n");
printf("3. DELETE\n");
printf("4. SEARCH\n");
printf("5. MODIFY\n");
printf("6. END\n");
printf("Enter your option : ");
scanf("%d",&op);
switch(op)
{
case 1:
insert();
display();
break;
case 2:
display();
break;
case 3:
del();
display();
break;
case 4:
printf("Enter the label to be searched : ");
scanf("%s",la);
y=search(la);
if(y==1)
{
printf("The label is already in the symbol Table");
}
else
{
printf("The label is not found in the symbol table");
}
break;
case 5:
modify();
display();
break;
case 6:
break;
}
}
while(op<6);
getch();
}
void insert()
{
int n;
char l[10];
printf("Enter the label : ");
scanf("%s",l);
n=search(l);
if(n==1)
{
printf("The label already exists. Duplicate cant be inserted\n");
}
else
{
struct symbtab *p;
p=malloc(sizeof(struct symbtab));
strcpy(p->label,l);
printf("Enter the address : ");
scanf("%d",&p->addr);
p->next=null;
if(size==0)
{
first=p;
last=p;
}
else
{
last->next=p;
last=p;
}
size++;
}
}
void display()
{
int i;
struct symbtab *p;
p=first;
printf("LABEL\tADDRESS\n");
for(i=0;i<size;i++)
```

```
{
printf("%s\t%d\n",p->label,p->addr);
p=p->next;
}
}
int search(char lab[])
{
int i,flag=0;
struct symbtab *p;
p=first;
for(i=0;i<size;i++)
{
if(strcmp(p->label,lab)==0)
{
flag=1;
}
p=p->next;
}
return flag;
}
void modify()
{
char l[10],nl[10];
int add, choice, i, s;
struct symbtab *p;
p=first;
printf("What do you want to modify?\n");
printf("1. Only the label\n");
printf("2. Only the address of a particular label\n");
printf("3. Both the label and address\n");
printf("Enter your choice : ");
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter the old label\n");
scanf("%s",l);
printf("Enter the new label\n");
scanf("%s",nl);
s=search(l);
if(s==0)
{
printf("NO such label");
}
else
{
```

```
for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0)
{
strcpy(p->label,nl);
}
p=p->next;
}
}
break;
case 2:
printf("Enter the label whose address is to modified\n");
scanf("%s",l);
printf("Enter the new address\n");
scanf("%d",&add);
s=search(l);
if(s==0)
{
printf("NO such label");
}
else
{
for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0)
{
p->addr=add;
}
p=p->next;
}
}
break;
case 3:
printf("Enter the old label : ");
scanf("%s",l);
printf("Enter the new label : ");
scanf("%s",nl);
printf("Enter the new address : ");
scanf("%d",&add);
s=search(l);
if(s==0)
{
printf("NO such label");
}
else
{
```

```
for(i=0;i<size;i++)
{
if(strcmp(p->label,l)==0)
{
strcpy(p->label,nl);
p->addr=add;
}
p=p->next;
}
}
break;
}
}
void del()
{
int a;
char l[10];
struct symbtab *p,*q;
p=first;
printf("Enter the label to be deleted\n");
scanf("%s",l);
a=search(l);
if(a==0)
{
printf("Label not found\n");
}
else
{
if(strcmp(first->label,l)==0)
{
first=first->next;
}
else if(strcmp(last->label,l)==0)
{
q=p->next;
while(strcmp(q->label,l)!=0)
{
p=p->next;
q=q->next;
}
p->next=null;
last=p;
}
else
{
q=p->next;
```

```
while(strcmp(q->label,l)!=0)
{
p=p->next;
q=q->next;
}
p->next=q->next;
}
size--;
}
}
```

RESULT: Thus a symbol table is implemented in C .

EXPT NO 2

Implement pass one of a two pass assembler.

AIM

To implement pass one of a two pass assembler.

ALGORITHM

1. Open the files fp1 and fp4 in read mode and fp2 and fp3 in write mode
   2.Read the source program
   3. If the opcode read in the source program is START, the variable location counter is initialized with the operand value.
   4. Else the location counter is initialized to 0.
   5. The source program is read line by line until the reach of opcode END.
   6. Check whether the opcode read is present in the operation code table.
   7. If the opcode is present, then the location counter is incremented by 3.
   8. If the opcode read is WORD, the location counter is incremented by3.
   9. If the opcode read is RESW, the operand value is multiplied by 3 and then the location
   counter is incremented.
   10. If the opcode read is RESB, the location counter value is incremented by operand value.
   11. If the opcode read is BYTE, the location counter is auto incremented.
   The length of the source program is found using the location counter value.

Source Code program in c pass one of a two pass assembler.

```
# include <stdio.h>
# include <conio.h>
# include <string.h>
void main()
{
char opcode[10],mnemonic[3],operand[10],label[10],code[10];
int locctr,start,length;
FILE *fp1,*fp2,*fp3,*fp4;
clrscr();
fp1=fopen("input.dat","r");
fp2=fopen("symtab.dat","w");
fp3=fopen("out.dat","w");
fp4=fopen("optab.dat","r");
fscanf(fp1,"%s%s%s",label,opcode,operand);
if(strcmp(opcode,"START")==0)
{
start=atoi(operand);
locctr=start;
fprintf(fp3,"\t%s\t%s\t%s\n",label,opcode,operand);
fscanf(fp1,"%s%s%s",label,opcode,operand);
}
else
locctr=0;
while(strcmp(opcode,"END")!=0)
{
fprintf(fp3,"%d\t",locctr);
if(strcmp(label,"**")!=0)
fprintf(fp2,"%s\t%d\n",label,locctr);
rewind(fp4);
fscanf(fp4,"%s",code);
while(strcmp(code,"END")!=0)
{
if(strcmp(opcode,code)==0)
{
locctr+=3;
break;
}
fscanf(fp4,"%s",code);
}
if(strcmp(opcode,"WORD")==0)
locctr+=3;
else if(strcmp(opcode,"RESW")==0)
locctr+=(3*(atoi(operand)));
else if(strcmp(opcode,"RESB")==0)
locctr+=(atoi(operand));
else if(strcmp(opcode,"BYTE")==0)
++locctr;
fprintf(fp3,"%s\t%s\t%s\n",label,opcode,operand);
fscanf(fp1,"%s%s%s",label,opcode,operand);
}
fprintf(fp3,"%d\t%s\t%s\t\%s\n",locctr,label,opcode,operand);
length=locctr-start;
printf("The length of the program is %d",length);
fclose(fp1);
fclose(fp2);
fclose(fp3);
fclose(fp4);
getch();
}
```

INPUT FILES

INPUT.DAT

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

OPTAB.DAT

```
START
LDA
STA
LDCH
STCH
END
```

OUTPUT FILES

OUT.DAT

```
** START 2000
2000 ** LDA FIVE
2003 ** STA ALPHA
2006 ** LDCH CHARZ
2009 ** STCH C1
2012 ALPHA RESW 1
2015 FIVE WORD 5
2018 CHARZ BYTE C’Z’
2019 C1 RESB 1
2020 ** END **
```

SYMTAB.DAT

```
ALPHA 2012
FIVE 2015
CHARZ 2018
C1 2018
```

RESULT:

Thus a C program was written to implement PASS ONE of a two pass assembler

EXPT NO 3

Implement pass two of a two pass assembler.

AIM :

To implement pass two of a two pass assembler.

ALGORITHM :

```
1. Start the program
2. Initialize all the variables
3. Open a file by name
   fp1=fopen("assmlist.dat","w");
   fp2=fopen("symtab.dat","r");
   fp3=fopen("intermediate.dat","r");
   fp4=fopen("optab.dat","r");
4. Read the content of the file
5. If opcode is BYTE
   if(strcmp(opcode,"BYTE")==0)
   Then
   fprintf(fp1,"%d\t%s\t%s\t%s\t",address,label,opcode,operand);
   Else if opcode is WORD
   else if(strcmp(opcode,"WORD")==0)
   then
   fprintf(fp1,"%d\t%s\t%s\t%s\t00000%s\n",address,label,opcode,operand,a);
   Else perform
   else if((strcmp(opcode,"RESB")==0)(strcmp(opcode,"RESW")==0))
   fprintf(fp1,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);
   if it is not math anything
   else
   fprintf(fp1,"%d\t%s\t%s\t%s\t%d0000\n",address,label,opcode,operand,code);
6. Finally terminate the of pass two of pass two assembler
```

Source code program in c pass two of pass two assembler

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
char a[10],ad[10],label[10],opcode[10],operand[10],mnemonic[10],symbol[10];
int i,address,code,add,len,actual_len;
FILE *fp1,*fp2,*fp3,*fp4;
clrscr();
fp1=fopen("assmlist.dat","w");
```

```
fp2=fopen("symtab.dat","r");
fp3=fopen("intermediate.dat","r");
fp4=fopen("optab.dat","r");
fscanf(fp3,"%s%s%s",label,opcode,operand);
if(strcmp(opcode,"START")==0)
{
fprintf(fp1,"\t%s\t%s\t%s\n",label,opcode,operand);
fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);
}
while(strcmp(opcode,"END")!=0)
{
if(strcmp(opcode,"BYTE")==0)
{
fprintf(fp1,"%d\t%s\t%s\t%s\t",address,label,opcode,operand);
len=strlen(operand);
actual_len=len-3;
for(i=2;i<(actual_len+2);i++)
{
itoa(operand[i],ad,16);
fprintf(fp1,"%s",ad);
}
fprintf(fp1,"\n");
}
else if(strcmp(opcode,"WORD")==0)
{
len=strlen(operand);
itoa(atoi(operand),a,10);
fprintf(fp1,"%d\t%s\t%s\t%s\t00000%s\n",address,label,opcode,operand,a);
}
else if((strcmp(opcode,"RESB")==0)(strcmp(opcode,"RESW")==0))
{
fprintf(fp1,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);
}
else
{
rewind(fp4);
fscanf(fp4,"%s%d",mnemonic,&code);
while(strcmp(opcode,mnemonic)!=0)
fscanf(fp4,"%s%d",mnemonic,&code);
if(strcmp(operand,"**")==0)
{
fprintf(fp1,"%d\t%s\t%s\t%s\t%d0000\n",address,label,opcode,operand,code);
}
else
{
rewind(fp2);
fscanf(fp2,"%s%d",symbol,&add);
while(strcmp(operand,symbol)!=0)
{
fscanf(fp2,"%s%d",symbol,&add);
}
fprintf(fp1,"%d\t%s\t%s\t%s\t%d%d\n",address,label,opcode,operand,code,add);
}
}
fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);
}
fprintf(fp1,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);
printf("Finished");
fclose(fp1);
fclose(fp2);
fclose(fp3);
fclose(fp4);
getch();
}
```

INPUT FILES

INTERMEDIATE.DAT

```
** START 2000
2000 ** LDA FIVE
2003 ** STA ALPHA
2006 ** LDCH CHARZ
2009 ** STCH C1
2012 ALPHA RESW 1
2015 FIVE WORD 5
2018 CHARZ BYTE C'EOF'
2019 C1 RESB 1
2020 ** END **
```

OPTAB.DAT

```
LDA 33
STA 44
LDCH 53
STCH 57
END *
```

SYMTAB.DAT

```
ALPHA 2012
FIVE 2015
CHARZ 2018
C1 2019
```

OUTPUT FILE :
ASSMLIST.DAT

```
** START 2000
2000 ** LDA FIVE 332015
2003 ** STA ALPHA 442012
2006 ** LDCH CHARZ 532018
2009 ** STCH C1 572019
2012 ALPHA RESW 1
2015 FIVE WORD 5 000005
2018 CHARZ BYTE C'EOF' 454f46
2019 C1 RESB 1
2020 ** END **
```

RESULT:

Thus pass two of a two pass assembler was implemented in C.

EXPT NO 4

Implement a single pass assembler.

AIM

*To i*mplement a single pass assembler.

ALGORITHM

Source code program in c-single pass assembler

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
struct st
{
char op[25];
int nb;
char code[3];
}s;
void main()
{
int i=0,lc,h=0;
char g[10],c[10];
FILE *fp1,*fp2,*fp3;
clrscr();
fp1=fopen("pgm.dat","r");
fp3=fopen("oneoutput.dat","w");
fprintf(fp3,"lc\topcode\tobjcode\n");
lc=0x2000;
while(!feof(fp1))
{
fgets(g,35,fp1);
while(g[i]!='\n')
{
if(g[i]>='A' && g[i]<='Z')
{
c[h]=g[i];
h++;
}
i++;
}
c[h]='\0';
fp2=fopen("opcode.dat","r");
while(!feof(fp2))
{
fscanf(fp2,"%s%d%s",s.op,&s.nb,s.code);
if(strcmp(c,s.op)==0)
{
fprintf(fp3,"%x\t%s",lc);
switch(s.nb)
{
case 1:
fprintf(fp3,"\t%s\n",s.code);
break;
case 2:
fprintf(fp3,"\t%s\t%c%c",s.code,g[i-2],g[i-1]);
fprintf(fp3,"\n");
break;
case 3:
fprintf(fp3,"\t%s\t%c%c\t%c%c",s.code,g[i-2],g[i-1],g[i-4],g[i-3]);
fprintf(fp3,"\n");
break;
}
lc=lc+s.nb;
break;
}
}
fclose(fp2);
i=0;
h=0;
}}
```

INPUT FILES:

opcode.dat

```
MOVAB   1   78
ADDB    1   81
MVIA    2   24
STA     3   32
HLT     1   76
```

pgm.dat

```
MVI   A,06
ADD   B
STA   2500
HLT
```

RESULT:

Thus a single pass assembler was implemented in C.

EXPT NO 5

Implement a two pass macro processor

AIM:

To Implement a two pass macro processor

ALGORITHM:

```
begin {macro processor}
    EXPANDING := FALSE
    while OPCODE  ≠ 'END' do
       begin
          GETLINE
          PROCESSLINE
       end {while}
end {macro processor}


procedure PROCESSLINE
    begin
       search NAMTAB for OPCODE
       if found then
          EXPAND
       else if OPCODE = 'MACRO' then
          DEFINE
       else write source line to expanded file
    end {PROCESSLINE}
```

```
procedure EXPAND
    begin
        EXPANDING  := TRUE
        get first line of macro definition {prototype} from DEFTAB
        set up arguments from macro invocation in ARGTAB
        write macro invocation to expanded file as a comment
        while not end of macro definition do
            begin
                GETLINE
                PROCESSLINE
            end {while}
        EXPANDING := FALSE
    end {EXPAND}


procedure GETLINE
    begin
        if EXPANDING then
            begin
                get next line of macro definition from DEFTAB
                substitute arguments from ARGTAB for positional notation
            end {if}
        else
            read next line from input file
    end {GETLINE}
```

Figure 4.5 (*cont'd*)

Source code program in c- *two pass macro processor*

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<dos.h>
//------------------------------------------Header Files
struct mnt
{
char name[20];
int mdtindex;
int SkipFlag;
}mnt[10];
//------------------------
struct mdt
{
int srno;
char inst[20];
}mdt[10];
//------------------------
struct ala
{
int index;
char arg[20];
}ala[10];
//------------------------
int mntc,mdtc,alac;
//----------------------------------------------------
```

```
char *replace_str(char *str, char *orig, char *rep)
{
  static char buffer[4096];
  char *p;

  if(!(p = strstr(str, orig)))  // Is 'orig' even in 'str'?
    {
    return str;
    }

  strncpy(buffer, str, p-str); // Copy characters from 'str' start to
'orig' st$
  buffer[p-str] = '\0';
  sprintf(buffer+(p-str),"%s%s",rep,p+strlen(orig));

  if(!(p = strstr(buffer, orig)))  // Is 'orig' even in 'buffer'?
    return buffer;
  else
     replace_str(buffer,orig,rep);
}
//----------------------------------------------------------------
void dispALA()
{
int i;
  printf("\nIndex\tArgument");
  printf("\n------------------");
for(i=0;i<alac;i++)
  printf("\n  %d      %s",ala[i].index,ala[i].arg);
}
//-----------------------------------------------------------------
void dispMDT()
{
int i;
  printf("\nIndex\tArgument");
  printf("\n------------------");
for(i=0;i<mdtc;i++)
  printf("\n  %d      %s",mdt[i].srno,mdt[i].inst);
}
//-----------------------------------------------------------------
void pass1()
{
char ch,ch1;
int flag=1;
int i=0,j=0,k=0,l=0;
char token[10],token1[10];
char temp[5],temp1[5];
char tline[80],line[80];

FILE *src;
FILE *dest;
clrscr();
strcpy(temp,"#");

src = fopen("ip.c","r");
dest = fopen("op.c","w");
```

```
do{
    do{                   //----> To Seperate Out Token
      ch = fgetc(src);    //----> ch : Current Pointer
      token[i] = ch;
      i++;
      }while(ch!=EOF && !isspace(ch));
      token[i-1] = '\0';  //----> Mask Current ch
      //-------------------------------------------
      //-------------------------------------------
      //-------------------------------------------
      if(!strcmp(token,"macro")) //--> Check For "macro"
      {
            do{                  //--> Macro Name
              ch1 = fgetc(src);
              token1[j] = ch1;
              j++;
              }while(ch1!=EOF && !isspace(ch1));
              token1[j-1] = '\0';
              //------------------------------------
              strcpy(mnt[mntc].name,token1);
              mnt[mntc].mdtindex = mdtc;
              mntc++;
              //--------------------------------------
              //If no Parameter i.e ALA is Empty
              //--------------------------------------
              if(ch1=='\n')
              {
              mnt[mntc-1].SkipFlag = 1;
              goto Mid;
              }
              //--------------------------------------
              //Create ALA
              //--------------------------------------
              do{                         //ALA Content
                ch1 = fgetc(src);
                if(ch1=='&')
                    flag=1;
                if(ch1==','||ch1=='\n')
                    {
                    token1[k]='\0';
                    flag=0;
                    ala[alac].index = alac;
                    strcpy(ala[alac].arg,token1);
                    alac++;
                    k=0;
                    }
                if(flag==1)
                  token1[k++] = ch1;
                }while(ch1!=EOF && ch1!='\n');
              //------------------------------------------
         Mid:
              flag=1;
              j=0;
              do
              {
              line[0]='\0';
```

```
                    do{                          //MDT
                       ch1 = fgetc(src);
                       line[j++]=ch1;
                       }
                       while(ch1!=EOF && ch1!='\n');
                       line[j-1]='\0';

                         for(l = 0 ;l<alac;l++)
                                 {
                                 sprintf(temp1,"%d",l); //0
                                 strcat(temp,temp1);    //#0
              sprintf(tline,"%s",replace_str(line,ala[l].arg,temp));
                                  //  if(strcmp(tline,""))
                                         strcpy(line,tline);
                                 strcpy(temp,"#");
                                 }
                       strcpy(mdt[mdtc].inst,line);
                       mdt[mdtc].srno = mdtc;
                       mdtc++;
                       j=0;
                 }while(strcmp(line,"mend"));
              ch = ' ';
           }//end if
           else
               fprintf(dest,"%s",token);
       //-----------------------------------------------------------------
       if(ch=='\n')
          fprintf(dest,"\n");
       if(ch==' ')
          fprintf(dest," ");
       //------------------------------------------------------------------
      if(isspace(ch))
          {
          token[0]='\0';
          i=0;
          }
       else
          {
          token[0]= ch;
          token[1]= '\0';
          i=1;
          }
   }while(ch!=EOF); //Outer od
   fclose(src);
   fclose(dest);
 }
 //-----------------------------------------------------------------
 void disp()
 {
 int i;
 char ch;
 FILE *src;
 src = fopen("op.c","r");
 do{
   ch = fgetc(src);
   printf("%c",ch);
```

```
  }while(ch!=EOF);
}
//------------------------------------------------------------------
void pass2()
{
char ch;
int counter=0;
int start_index;
char ch1,ch2,pch; //comment validation
int len,flag=0,sflag=0;
int i = 0,j=0,k=0,l=0;
char token[10];
char token1[10];
char temp[5],temp1[5];
char tline[80];
char line[80];
char repl[10];

FILE *src;
alac = 0;
strcpy(temp,"#");

src = fopen("op.c","r");

do
{
   do{                //For Seperate Token
     ch = fgetc(src);
     token[i] = ch;
     i++;
     }while(ch!=EOF && !isspace(ch));
     token[i-1] = '\0';
     //--------------------------------------------
     for(j=0;j<mntc;j++)
     if(!strcmp(token,mnt[j].name)) //if token="mac1"
     {
     sflag = 1;
     start_index = mnt[j].mdtindex;
     if(mnt[j].SkipFlag==1)
          goto Next;
           do{                         //Modify ALA Content
              ch1 = fgetc(src);
              flag=1;
              if(ch1==','||ch1=='\n')
                    {
                    token1[k]='\0';
                    flag=0;
                    ala[alac].index = alac;
                    strcpy(ala[alac].arg,token1);
                    alac++;
                    k=0;
                    counter++;
                    }
              if(flag==1)
                  token1[k++] = ch1;
              }while(ch1!=EOF && ch1!='\n');
```

```
                //----------------------------------------
        Next:
          k = counter;
          do
          {
          strcpy(line,mdt[start_index].inst);
             if(!strcmp(line,"mend"))
                  break;
          for(l=0;l<k;l++)
          {
          strcpy(temp,"#");
          sprintf(temp1,"%d",l);   //temp1 = "0"
          strcat(temp,temp1);      //temp = "#" before now temp = "#0"
          strcpy(repl,ala[l].arg);  //repl = 10
               sprintf(tline,"%s",replace_str(line,temp,repl));
                     if(strcmp(tline,""))
                          strcpy(line,tline);
          }
          printf("\n%s",line);
          start_index++;
          }while(strcmp(line,"mend"));
          printf("\n");
          sflag = 1;
       }//end if

       if(sflag==0)
       {
       printf("%s",token);
       }

       if(ch=='\n')
          printf("\n");
       if(ch==' ')
          printf(" ");

       if(isspace(ch))
          {
          token[0]='\0';
          i=0;
          }
         else
          {
          token[0]= ch;
          token[1]= '\0';
          i=1;
          }
          sflag = 0;
          counter = 0;
          alac = 0;
          token1[0] = '\0';
          k = 0;
      }while(ch!=EOF);

fclose(src);

}
```

```
//-----------------------------------
void main()
{
clrscr();
pass1();
printf("\n---------------------------");
printf("\nInput Program : ");
printf("\n---------------------------\n");
disp();
printf("\n---------------------------");
printf("\nContent of ALA : ");
printf("\n---------------------------\n");
dispALA();
printf("\n---------------------------");
printf("\nContent of MDT : ");
printf("\n---------------------------\n");
dispMDT();
printf("\n----------------------------\n");
pass2();

getch();
}
```

RESULT:

Thus a two pass macroprocessor was implemented in C

EXPT NO 6

Implement a single pass macro processor.

AIM

*To i*mplement a single pass macro processor.

ALGORITHM

```
1.Start the macro processor program
2. Include the necessary header files and variable
3. Open the three files
f1=macin.dat with read privilege
f2=macout.dat with write privilege
f3= deftab.dat with write privilege
4. Get the variable form f1 file macin.dat for label,opcode,operand
5. Read the variable until the opcode is not is equal to zero
Then check if the opcode is equal to Macro if Macro
Then
6. Copy macroname=label
7. Get the variable label ,opcode ,operand
8. In these if condition perform the while loop until opcode is not equal to MEND
9. Copy the variable
d[lines].lab=label
d[lines].opc=opcode
d[lines].oper=operand
and increase lines++;
close while loop and if condtion
else if opcode is equal to macro name
10. Perform the for loop from 0 to length
fprint for d[i].lab,d[i].opc,d[i].oper
else if it is not match
fprintf(f2,"%s\t%s\t%s\n",label,opcode,operand);
11. Finally terminate the program
```

Source code in c program for perform Simple macro processor

```
# include <stdio.h>
# include <conio.h>
# include <string.h>
# include <stdlib.h>
struct deftab
{
```

```
char lab[10];
char opc[10];
char oper[10];
}d[10];
void main()
{
char label[10],opcode[10],operand[10],newlabel[10],newoperand[10];
char macroname[10];
int i,lines;
FILE *f1,*f2,*f3;
clrscr();
f1=fopen("macin.dat","r");
f2=fopen("macout.dat","w");
f3=fopen("deftab.dat","w");
fscanf(f1,"%s%s%s",label,opcode,operand);
while(strcmp(opcode,"END")!=0)
{
if(strcmp(opcode,"MACRO")==0)
{
strcpy(macroname,label);
fscanf(f1,"%s%s%s",label,opcode,operand);
lines=0;
while(strcmp(opcode,"MEND")!=0)
{
fprintf(f3,"%s\t%s\t%s\n",label,opcode,operand);
strcpy(d[lines].lab,label);
strcpy(d[lines].opc,opcode);
strcpy(d[lines].oper,operand);
fscanf(f1,"%s%s%s",label,opcode,operand);
lines++;
}
}
else if(strcmp(opcode,macroname)==0)
{
printf("Lines = %d\n",lines);
for(i=0;i<lines;i++)
{
fprintf(f2,"%s\t%s\t%s\n",d[i].lab,d[i].opc,d[i].oper);
printf("DLAB = %s\nDOPC = %s\nDOPER = %s\n",d[i].lab,d[i].opc,d[i].oper);
}
}
else
fprintf(f2,"%s\t%s\t%s\n",label,opcode,operand);
fscanf(f1,"%s%s%s",label,opcode,operand);
}
fprintf(f2,"%s\t%s\t%s\n",label,opcode,operand);
```

```
fclose(f1);
fclose(f2);
fclose(f3);
printf("FINISHED");
getch();
}
```

INPUT FILE :

```
MACIN.DAT
CALC START 1000
SUM MACRO **
** LDA #5
** ADD #10
** STA 2000
** MEND **
** LDA LENGTH
** COMP ZERO
** JEQ LOOP
** SUM **
LENGTH WORD 5
ZERO WORD 0
LOOP SUM **
** END **
```

OUTPUT FILES :

```
MACOUT.DAT
CALC START 1000
** LDA LENGTH
** COMP ZERO
** JEQ LOOP
** LDA #5
** ADD #10
** STA 2000
LENGTH WORD 5
ZERO WORD 0
** LDA #5
** ADD #10
** STA 2000
** END **
```

```
DEFTAB.DAT
** LDA #5
** ADD #10
** STA 2000
```

RESULT:

Thus a single pass macroprocessor was implemented in C.

EXPT NO 7

Implement an absolute loader.

AIM:

To implement an absolute loader.

ALGORITHM:

1. Start the program
2. Assign the required variable
3. Open the files

fp1=fopen("input.dat","r");
fp2=fopen("output.dat","w");

4. Read the content
5. Using while loop perform the loop until character is not equal to E

while(strcmp(input,"E")!=0)
Then compare the character is equal to H
If H then
fscanf(fp1,"%d",&start);
Like that get length, and input
Else if the character is T
Then
Then perform the frprintf in fp1 for input file for ,
input[0],inuput[1] for address
input[2],inuput[3] for address+1
input[4],inuput[5] for address+2
Else if it is not H or T
Then perform the frprintf in fp2 for output file for ,
input[0],inuput[1] for address
input[2],inuput[3] for address+1
input[4],inuput[5] for address+2

```
fprintf(fp2,"%d\t%c%c\n",address,input[0],input[1]);
fprintf(fp2,"%d\t%c%c\n",(address+1),input[2],input[3]);
fprintf(fp2,"%d\t%c%c\n",(address+2),input[4],input[5]);
address+=3;
fscanf(fp1,"%s",input);
```

6. Finally terminate the program

Source code program in c performing Absoluter Loader

```
# include <stdio.h>
# include <conio.h>
# include <string.h>
void main()
{
char input[10];
int start,length,address;
FILE *fp1,*fp2;
clrscr();
fp1=fopen("input.dat","r");
fp2=fopen("output.dat","w");
fscanf(fp1,"%s",input);
while(strcmp(input,"E")!=0)
{
if(strcmp(input,"H")==0)
{
fscanf(fp1,"%d",&start);
fscanf(fp1,"%d",&length);
fscanf(fp1,"%s",input);
}
else if(strcmp(input,"T")==0)
{
fscanf(fp1,"%d",&address);
fscanf(fp1,"%s",input);
fprintf(fp2,"%d\t%c%c\n",address,input[0],input[1]);
fprintf(fp2,"%d\t%c%c\n",(address+1),input[2],input[3]);
fprintf(fp2,"%d\t%c%c\n",(address+2),input[4],input[5]);
address+=3;
fscanf(fp1,"%s",input);
}
else
{
fprintf(fp2,"%d\t%c%c\n",address,input[0],input[1]);
fprintf(fp2,"%d\t%c%c\n",(address+1),input[2],input[3]);
fprintf(fp2,"%d\t%c%c\n",(address+2),input[4],input[5]);
address+=3;
fscanf(fp1,"%s",input);
}
}
fclose(fp1);
fclose(fp2);
printf("FINISHED");
getch();
}
```

```
INPUT FILE
INPUT.DAT
H 1000 232
T 1000 142033 483039 102036
T 2000 298300 230000 282030 302015
E
```

OUTPUT FILE

```
OUTPUT.DAT
1000 14
1001 20
1002 33
1003 48
1004 30
1005 39
1006 10
1007 20
1008 36
2000 29
2001 83
2002 00
2003 23
2004 00
2005 00
2006 28
2007 20
2008 30
2009 30
2010 20
2011 15
```

RESULT:

Thus an absolute loader was implemented in C.

EXPT NO 8

Implement a relocating loader.

AIM

To implement a relocating loader.

ALGORITHM

```
1. Start the program
2. Include the necessary header file and variable
3. Open the two file for
fp1= relinput.dat and give read
fp2= reloutput.dat and give write
4. Read the content
5. Using while loop perform the loop until character is not equal to E
while(strcmp(input,"E")!=0)
If the character is H
Get the variable add, length, and input
Else if the character is T
Get the variable address and bitmask
And perform the for loop for starting zero to up to len
Get the opcode ,addr and assign relocbit to bitmask
If relocabit is zero
Then
actualadd=addr;
else
Add the addr and star value
6. Finally terminate the program
```

Source code program in c for Relocation Loader

```
# include <stdio.h>
# include <conio.h>
# include <string.h>
# include <stdlib.h>
void main()
{
char add[6],length[10],input[10],binary[12],bitmask[12],relocbit;
int start,inp,len,i,address,opcode,addr,actualadd;
FILE *fp1,*fp2;
clrscr();
printf("Enter the actual starting address : ");
scanf("%d",&start);
fp1=fopen("relinput.dat","r");
fp2=fopen("reloutput.dat","w");
fscanf(fp1,"%s",input);
while(strcmp(input,"E")!=0)
{
if(strcmp(input,"H")==0)
{
fscanf(fp1,"%s",add);
fscanf(fp1,"%s",length);
fscanf(fp1,"%s",input);
}
if(strcmp(input,"T")==0)
{
fscanf(fp1,"%d",&address);
fscanf(fp1,"%s",bitmask);
address+=start;
len=strlen(bitmask);
for(i=0;i<len;i++)
{
fscanf(fp1,"%d",&opcode);
fscanf(fp1,"%d",&addr);
relocbit=bitmask[i];
if(relocbit=='0')
actualadd=addr;
else
actualadd=addr+start;
fprintf(fp2,"%d\t%d%d\n",address,opcode,actualadd);
address+=3;
}
fscanf(fp1,"%s",input);
}
}
fclose(fp1);
fclose(fp2);
printf("FINISHED");
getch();
}
```

INPUT : RELINPUT.DAT

```
H 1000 200
T 1000 11001 14 1033 48 1039 90 1776 92 1765 57 1765
```

```
T 2011 11110 23 1838 43 1979 89 1060 66 1849 99 1477
E 1000
```

```
OUTPUT :
Enter the actual starting address :4000

RELOUTPUT.DAT
4000 144033
4003 484039
4006 901776
4009 921765
4012 574765
5011 234838
5014 434979
5017 894060
5020 664849
5023 991477
```

RESULT:

Thus a relocating loader was implemented in C.

EXPT NO 9

Implement pass one of a direct-linking loader.

AIM

To implement pass one of a direct-linking loader.

OBJECTIVES

1. Start the program for linking loader
2. Assign the necessary variable and the header variable
3. Open the two file
4. In fp1 for link input file for read privilege
5. And fp2 for load map file for write privilege
6. Read the character until the input is not equal to END
7. And the check the character in if condition input is H then
8. Get the name from fp1
9. Copy the name between csname=name
10. And extsym=**
11. Else if the character is D
12. Then get the input variable
13. In these if condition perform the while loop the read character until the input is not equal to R
    The copy csnmae=**
14. And extsym=input
15. Then add address =add+csaddr
16. And length is equal to zero
17. And perform the operation (see the Source code)
18. Finally terminate the program

Source code in c program performing passes one linking loader

```
# include <stdio.h>
# include <conio.h>
# include <string.h>
# define MAX 20
struct estab
{
char csname[10];
char extsym[10];
```

```
int address;
int length;
}es[MAX];
void main()
{
char input[10],name[10],symbol[10];
int count=0,progaddr,csaddr,add,len;
FILE *fp1,*fp2;
clrscr();
fp1=fopen("linkinput.dat","r");
fp2=fopen("loadmap.dat","w");
printf("Enter the location where the program has to be loaded : ");
scanf("%d",&progaddr);
csaddr=progaddr;
fprintf(fp2,"CS_NAME\tEXT_SYM_NAME\tADDRESS\tLENGTH\n");
fprintf(fp2,"--------------------------------------\n");
fscanf(fp1,"%s",input);
while(strcmp(input,"END")!=0)
{
if(strcmp(input,"H")==0)
{
fscanf(fp1,"%s",name);
strcpy(es[count].csname,name);
strcpy(es[count].extsym,"**");
fscanf(fp1,"%d",&add);
es[count].address=add+csaddr;
fscanf(fp1,"%d",&len);
es[count].length=len;
fprintf(fp2,"%s\t%s\t\t%d\t
%d\n",es[count].csname,es[count].extsym,es[count].address,es[count].length);
count++;
}
else if(strcmp(input,"D")==0)
{
fscanf(fp1,"%s",input);
while(strcmp(input,"R")!=0)
{
strcpy(es[count].csname,"**");
strcpy(es[count].extsym,input);
fscanf(fp1,"%d",&add);
// printf("CSADDR = %d",csaddr);
es[count].address=add+csaddr;
es[count].length=0;
fprintf(fp2,"%s\t%s\t\t%d\t
%d\n",es[count].csname,es[count].extsym,es[count].address,es[count].length);
count++;
fscanf(fp1,"%s",input);
}
csaddr=csaddr+len;
}
else if(strcmp(input,"T")==0)
{
while(strcmp(input,"E")!=0)
fscanf(fp1,"%s",input);
}
fscanf(fp1,"%s",input);
}
fprintf(fp2,"---------------------------------------\n");
fclose(fp1);
fclose(fp2);
printf("FINISHED\n");
getch();
}
```

INPUT FILE :

```
LINKINPUT.DAT
H PROGA 0000 1000
D LISTA 0040 ENDA 0054
R LISTB ENDB LISTC ENDC
T 0020 141033 465555 678909 568787 345678
T 0054 000014 789087 776555 876666 456666
M 0054 06 +LISTC
E 0000

H PROGB 0000 2000
D LISTB 0040 ENDB 0054
R LISTA ENDA LISTC ENDC
T 0020 141033 465555 678909 568787 345678
T 0054 000000 789087 776555 876666 456666
M 0054 06 +ENDA
M 0054 06 -LISTA
M 0054 06 +LISTC
E 0000

H PROGC 0000 3000
D LISTC 0040 ENDC 0054
R LISTA ENDA LISTC ENDB
T 0020 141033 465555 678909 568787 345678
T 0054 000020 789087 776555 876666 456666
M 0054 06 +ENDA
```

```
M 0054 06 -LISTA
M 0054 06 +PROGC
E 0000
END
```

OUTPUT:

```
Enter the location where the program has to be loaded : 5000
LOADMAP.DAT
CS_NAME EXT_SYM_NAME ADDRESS LENGTH
-----------------------------------------------
PROGA ** 5000 1000
** LISTA 5040 0
** ENDA 5054 0
PROGB ** 6000 2000
** LISTB 6040 0
** ENDB 6054 0
PROGC ** 8000 3000
** LISTC 8040 0
** ENDC 8054 0
```

RESULT:

Thus pass one of a direct linking loader was implemented in C.

EXPT NO 10

Implement pass two of a direct-linking loader.

AIM

To implement pass two of a direct-linking loader.

RESULT:

EXPT NO 11

Implement a simple text editor with features like insertion / deletion of a character, word, and sentence.

AIM

To implement a simple text editor with features like insertion / deletion of a character, word, and sentence.

ALGORITHM

Source code in c Program for Text Editor

```
# include <stdio.h>
# include <conio.h>
# include <ctype.h>
# include <dos.h>
# include <iostream.h>
# include <fstream.h>
char filename[15];
char buff[1000];
int curx,cury,count;
void cur_pos()
{
```

```
curx=wherex();
cury=wherey();
textcolor(12);
textbackground(9);
gotoxy(35,1);
cout<<"\n";
cout<<"####################################################\n";
cout<<"\n";
cout<<"\t\tTEXT EDITOR\n";
cout<<"####################################################\n";
cout<<"\n Type your text and then press ESC key\n";
gotoxy(100,500);
cprintf("%2d%2d",cury,curx);
gotoxy(curx,cury);
}

void main()
{
char ch,c;
ofstream outfile;
ifstream infile;
clrscr();
cur_pos();
curx=wherex();
cury=wherey();
while(c!=27)
{
c=getch();
switch(c)
{
case 80: //down arrow
gotoxy(curx,cury+1);
break;
case 77: //right side
gotoxy(curx+1,cury);
break;
case 72: //up arrow
gotoxy(curx,cury-1);
break;
case 75: //left side
gotoxy(curx-1,cury);
break;
case 32: //space
printf(" ");
buff[count++]=' ';
break;
case 13: //new line
gotoxy(1,cury+1);
buff[count++]='\n';
break;
default:
textcolor(13);
if((c>=65 && c<=122) (c>48 && c<57))
cprintf("%c",c);
break;
}
buff[count++]=c;
cur_pos();
}
cprintf("\n\nDo you want to save? (y/n)");
scanf("%c",&c);
if((c=='y')(c=='Y'))
{
cprintf("\n\nEnter the file name with extension in 8 characters only");
scanf("%s",filename);
outfile.open(filename,ios::out);
outfile<<buff;
outfile.close();
cout<<"\nDo you want to open? (y/n) \n";
ch=getch();
if((ch=='y')(ch=='Y'))
{
cprintf("\n\nEnter the file name to open");
scanf("%s",filename);
infile.open(filename,ios::in);
infile.get(buff,count,'*');
gotoxy(90,1);
printf("%s",buff);
getch();
infile.close();
}
}
}
```

RESULT:

Thus a Text editor was implemented in C

EXPT NO 12

Implement a symbol table with suitable hashing

AIM

To implement a symbol table with suitable hashing

ALGORITHM

Source code in c Program for Symbol table

```
# include <stdio.h>
# include <conio.h>
# include <ctype.h>
# include <dos.h>
# include <iostream.h>
# include <fstream.h>
void main()
{
int i,j,k,n,add=0,size;
```

```
char c,in[20],temp[20],val[20],var[20];
FILE*fp;
clrscr();
fp=fopen("input.c","r");
c=getc(fp);
printf('\n THE INPUT FILE IS:")
while(!feof(fp))
{
printf("%c",c);
c=getc(fp);
}
fclose(fp);
printf("\n**************symbol table**************");
printf("\n DAta type\tName\tNo.of Elements\tSize\tAddress");
printf("************************************");
while(!feof(fp))
{
if(isalpha(c))
{
i=0;
do
{
temp[i==]=c
c=getc(fp);
}while(isalpha(c));
temp[i]='\0';
if(strcmp("int",temp)==0)
{
strcpy(val," ");
strcpy(in," ");
strcpy(var," ");
```

```
while(c!=';')
{
strcpy(val," ");
c=getc(fp);
i=0;
do
{
var[i++]=c;
c=getc(fp);
}while(isalpha(c));
var[i]='\0';
if(c=='[')
{
i=0;
c=getc(fp);
strcpy(in," ")
do
{
in[i++]=c;
c=getc(fp);
k=atoi(in);
size=size*k;
}
else
{
strcpy(in,'1');
size=2;
}
if(c=='=')
c=getc(fp);
i=0;
do
{
val[i++]=c;
c=getc(fp);
}while(c!=','&&c!=';')
val[i]='\0';
}
printf("\n%s\t\t%s\t%s\t\t%s\t%dBytes\t%d",temp,var,val,in,size,add);
add=add+size;
}
}
}
else
c=getc(fp);
}
getch();
}
```

RESULT:

Thus the symbol table was generated using C.

VIVA VOCE QUESTIONS WITH ANSWERS

INTRODUCTION

1. Define system software.
It consists of variety of programs that supports the operation of the computer. This
software makes it possible for the user to focus on the other problems to be solved with out
needing to know how the machine works internally.
Eg: operating system, assembler, loader.
2. Give some applications of operating system.
- to make the computer easier to use
- to manage the resources in computer
- process management
- data and memory management
- to provide security to the user.

Operating system acts as an interface between the user and the system
Eg:windows,linux,unix,dos
3. Define compiler and interpreter.
Compiler is a set of program which converts the whole high level language program
to machine language program.
Interpreter is a set of programs which converts high level language program to machine
language program line by line.
4. Define loader.
Loader is a set of program that loads the machine language translated by the translator

into the main memory and makes it ready for execution.
5. What is the need of MAR register?
MAR (memory address register) is used to store the address of the memory from which
the data is to be read or to which the data is to be written.
6. Draw SS instruction format.
opcode L B1 D1 B2 D2
0 7 8 15 16 19 20 31 32 35 36 47
It is a 6 byte instruction used to move L+I bytes data fro the storage location1 to the
storage location2.
Storage location1 = D1+[B1]
Storage location2 = D2+[B2]
20
Eg: MOV 60,400(3),500(4)
7. Give any two differences between base relative addressing and program counter
relative addressing used in SIC/XE.
8. Define indirect addressing
In the case of immediate addressing the operand field gives the memory location. The
word from the given address is fetched and it gives the address of the operand.
Eg:ADD R5, [600]
Here the second operand is given in indirect addressing mode. First the word in
memory location 600 is fetched and which will give the address of the operand.
9. Define immediate addressing.
In this addressing mode the operand value is given directly. There is no need to refer
memory. The immediate addressing is indicated by the prefix '#'.
Eg: ADD #5
In this instruction one operand is in accumulator and the second operand is an
immediate value the value 5 is directly added with the accumulator content and the result is
stored in accumulator.
10. List out any two CISC and RISC machine.
CISC –Power PC, Cray T3E
RISC – VAX,Pentium Pro architecture
11. Following is a memory configuration:
Address Value Register R
1 5 5
5 7
6 5
What is the result of the following statement?
Base relative addressing PC relative addressing

Here the Target address is calculated
using the formula
Target address = Displacement + [B]
B-base register
Here the target address is calculated using the
formula
Target address = Displacement + [PC]
PC-program counter
Displacement lies between 0 to 4095 Displacement lies between –2048 to
2047
21
ADD 6(immediate) to R (indirect)
Here 6 is the immediate data and the next value is indirect data. ie, the
register contains
the address of the operand. Here the address of the operand is 5 and its
corresponding value
is 7.
6 + [R] = 6+ [5] = 6+ 7 =13
12. Following is a memory configuration:
Address Value Register R
4 9 6
5 7
6 2
What is the result of the following statement?
SUB 4(direct) to R (direct)
Here one operand is in the address location 4(direct addressing) and the
next operand
is in the register (register direct).
The resultant value is 9 –6 =3.
**13. What is the name of A and L register in SIC machine and also
specify its use.**
A-accumulator
Used for arithmetic operation. i.e., in the case of arithmetic operations
one operand is
in the accumulator, and other operand may be an immediate value,
register operand or
memory content. The operation given in the instruction is performed and
the result is stored
in the accumulator register.
L-linkage register
It is used to store the return address in the case of jump to subroutine
(JSUB)
instructions.
**14. What are the instruction formats used in SIC/XE architecture?
Give any one
format.**
Format 1 (1 byte), Format 2 (2 bytes), Format 3 (3 bytes) & Format 4(4
bytes)

Format 2:
8 4 4
OPCODE R1 R2
15. Consider the instructions in SIC/ XE programming
10 1000 LENGTH RESW 4
22
20 ----- NEW WORD 3
What is the value assign to the symbol NEW?
In the line 10 the address is 1000 and the instruction is RESW 4.It reserves 4
word (3 x 4 =12) areas for the symbol LENGTH. hence 12 is added to the LOCCTR.
Thus the value of the symbol NEW is 1000+12 =100C.
16. What is the difference between the instructions LDA # 3 and LDA THREE?
In the first instruction immediate addressing is used. Here the value 3 is directly
loaded into the accumulator register.
In the second instruction the memory reference is used. Here the address (address
assigned for the symbol THREE) is loaded into the accumulator register.
17. Differentiate trailing numeric and leading separate numeric.
The numeric format is used to represent numeric values with one digit per byte. In the
numeric format if the sign appears in the last byte it is known as the trailing numeric. If the
sign appears in a separate byte preceding the first digit then it is called as leading separate
numeric.
18. What are the addressing modes used in VAX architecture?
Register direct; register deferred, auto increment and decrement, program counter
relative, base relative, index register mode and indirect addressing are the various addressing
modes in VAX architecture.
19. How do you calculate the actual address in the case of register indirect with
immediate index mode?
Here the target address is calculated using the formula
T.A =(register) + displacement.
20. Write the sequence of instructions to perform the operation BETA = ALPHA + 1
using SIC instructions.
LDA ALPHA
ADD ONE
STA BETA
…. ….
ALPHA RESW 1

BETA RESW 1
ONE RESW 1
21. Write the sequence of instructions to perform the operation BETA = ALPHA+5
using SIC/XE instructions.
23
LDA ALPHA
ADD #1
STA BETA
…. ….
ALPHA RESW 1
BETA RESW 1
22. What is the use of TD instruction in SIC architecture?
The test device (TD) instruction tests whether the addressed device is ready to send or
receive a byte of data. The condition code is set to indicate the result of this test. Setting of <
means the device is ready to send or receive, and = means the device is not ready.

ASSEMBLERS

1. 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.
2. 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.
3. 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.
4. 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.
24
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.

5. 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.

6. 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.

7. What are the symbol defining statements generally used in assemblers?

‘EQU’-it allows the programmer to define symbols and specify their values
directly. The general format is
Symbol **EQU** value

‘ORG’-it is used to indirectly assign values to symbols. When this statement is
encountered the assembler resets its location counter to the specified value. The
general format is
ORG value

In the above two statements value is a constant or an expression involving constants
and previously defined symbols.

8. Define relocatable program.

An object program that contains the information necessary to perform required
modification in the object code depends on the starting location of the program during load

time is known as relocatable program.

9. Differentiate absolute expression and relative expression.

If the result of the expression is an absolute value (constant) then it is known as
absolute expression.
Eg: BUFEND – BUFFER
25
If the result of the expression is relative to the beginning of the program then it is
known as relative expression. label on instructions and data areas and references to the
location counter values are relative terms.
Eg: BUFEND + BUFFER

10. 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.

11. 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.

12. 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.

13. 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 .

14. What is the use of base register table in AIX assembler?

A base register table is used to remember which of the general purpose registers are
currently available as base registers and also the base addresses they contain.
26
.USING statement causes entry to the table and .DROP statement removes the
corresponding table entry.

15. 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.

16. Define modification record and give its format.

This record contains the information about the modification in the object code during
program relocation. the general format is
Col 1 M
Col 2-7 Starting location of the address field to be modified relative to the beginning
of the program
Col 8-9 length of the address field to be modified in half bytes.

**17. 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

18. 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.
27
Eg: program blocks, Literals.

19. How the register to register instructions are translated in assembler?

In the case of register to register instructions the operand field contains the register
name. During the translation first the object code is converted into its corresponding machine
language equivalent with the help of OPTAB. Then the SYMTAB is searched for the
numeric equivalent of register and that value is inserted into the operand field.
Eg: 125 1036 RDREC CLEAR X B410
B4-macine equivalent of the opcode CLEAR
10-numeric equivalent of the register X.

20. 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.

21. Define control section.

A control section is a part of the program that maintains its identity after assembly;
each control section can be loaded and relocated independently of the others.
Control sections are most often used for subroutines. The major benefit of using
control sections is to increase flexibility.

22. What is the difference between the assembler directive EXTREF and EXTDEF.

EXTDEF names external symbols that are defined in a particular control section
and may be used in other sections.
EXTREF names external symbols that are referred in a particular control section and
defined in another control section.

23. Give the general format of define record.

This record gives information about external symbols that are defined in a particular
control section. The format is
Col 1 D
Col 2-7 name of external symbol defined in this control section

Col 8-13 relative address of the symbol with in this control section
Col 14-73 name and relative address for other external symbols.

24. Give the use of assembler directive CSECT and USE

28

CSECT - used to divide the program into many control sections
USE – used to divide the program in to many blocks called program blocks

25. 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.

LOADERS AND LINKERS

1. 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.

2. Define absolute loader.

The loader, which is used only for loading, is known as absolute loader.
e.g. Bootstrap loader

3. What is meant by bootstrap loader?

This is a special type of absolute loader which loads the first program to be run by the
computer. (usually an operating system)

4. What are relative (relocative) loaders?

Loaders that allow for program relocation are called relocating (relocative) loaders.

5. What is the use of modification record?

Modification record is used for program relocation. Each modification record
specifies the starting address and the length of the field whose value is to be altered and also
describes the modification to be performed.

6. What are the 2 different techniques used for relocation?

29

Modification record method and relocation bit method.

7. Define Relocation bit method.

If the relocation bit corresponding to a word of object code is set to 1, the program's

starting address is to be added to this word when the program is relocated. Bit value 0
indicates no modification is required.

8. Define bit mask.

The relocation bits are gathered together following the length indicator in each text
record and which is called as bit mask. For e.g. the bit mask FFC(111111111100) specifies
that the first 10 words of object code are to be modified during relocation.

9. What is the need of ESTAB?

It is used to store the name and address of the each external symbol. It also indicates
in which control section the symbol is defined.

10. What is the use of the variable PROGADDR?

It gives the beginning address in memory where the linked program is to be loaded.
The starting address is obtained from the operating system.

11. Write the two passes of a linking loader.

Pass1: assigns address to all external symbols
Pass2: it performs actual loading, relocation and linking.

12. Define automatic library search.

In many linking loaders the subroutines called by the program being loaded are
automatically fetched from the library, linked with the main program and loaded. This feature
is referred to as automatic library search.

13. List the loader options INCLUDE &DELETE.

The general format of INCLUDE is
INCLUDE program_name (library name)
This command direct the loader to read the designated object program from a library and treat
it as the primary loader input.
The general format of DELETE command is
DELETE Csect-name
It instructs the loader to delete the named control sections from the sets of programs loaded.

14. Give the functions of the linking loader.

The linking loader performs the process of linking and relocation. It includes the
operation of automatic library search and the linked programs are directly loaded into the
memory.

15. Give the difference between linking loader and linkage editors.

Linking loader Linkage editor

The relocation and linking is performed
each time the program is loaded

It produces a linked version of a program
and which is written in a file for later
execution
Here the loading can be accomplished in a
single pass
Two passes are required
16. Define dynamic linking.
If the subroutine is loaded and linked to the program during its first call
(run time),
then it is called as dynamic loading or dynamic linking.
17. Write the advantage of dynamic linking.
It has the ability to load the routine only when they are needed.
The dynamic linking avoids the loading of entire library for each
execution.
18. What is meant by static executable and dynamic executable?
In static executable, all external symbols are bound and ready to run. In
dynamic
executables some symbols are bound at run time.
19. What is shared and private data?
The data divided among processing element is called shared data. If the
data is not
shared among processing elements then it is called private data.
20. Write the absolute loader algorithm.

```
Begin
Read Header record
Verify program name and length
Read first text record
While record type != 'E' do
Begin
Moved object code to specified location in memory
Read next object program record
End
Jump to address specified in End record
```

MACRO PROCESSORS

1. Define macro processor.
Macro processor is system software that replaces each macroinstruction
with the
corresponding group of source language statements. This is also called as
expanding of
macros.
2. What do macro expansion statements mean?
These statements give the name of the macroinstruction being invoked
and the
arguments to be used in expanding the macros. These statements are
also known as macro

call.
3. 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
4. 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.
5. 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.
6. 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.
7. What are the statements used for conditional macro expansion?
IF-ELSE-ENDIF statement
WHILE-ENDW statement
8. What is meant by positional parameters?
32
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.
9. Consider the macro definition
#define DISPLAY (EXPR) Printf (“EXPR = %d\n”, EXPR)
Expand the macro instruction DISPLAY (ANS)
Ans.: Printf (“EXPR = %d\n”, ANS)
10. 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.

11. How the macro is processed using two passes?
Pass1: processing of definitions
Pass 2:actual-macro expansion.
12. 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.
13. 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.
14. 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.
15. 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.
**16. What are the important factors considered while designing general purpose macro
processors?**
comments

grouping of statements
tokens
syntax used for macro definitions
17. What is the symbol used to generate unique labels?
$ symbol is used in macro definition to generate unique symbols. Each macro
expansion the $ symbol is replaced by $XX, where XX is the alpha numeric character.
18. 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.
19. 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

20. How to design the pass structure of a macro assembler?

To design the structure of macro-assembler, the functions of macro preprocessor and
the conventional assembler are merged. After merging, the functions are structured into
passes of the macro assembler.

TEXT EDITORS

1. Define interactive editor?

An interactive editor is a computer program that allows a user to create and revise a
target document. The term document includes objects such as computer programs, text,
equations, tables, diagrams, line art, and photographs any thing that one might find on a
printed page.

2. What are the tasks performed in the editing process?

4 tasks

Select the part of the target document to be viewed and manipulated.
Determine how to format this view on-line and how to display it.
Specify and execute operations that modify the target document.
Update the view appropriately.

34

3. What are the three categories of editor's devices?

Text device/ String devices
Button device/Choice devices
Locator device

4. What is the function performed in editing phase?

In the actual editing phase, the target document is created or altered with a set of
operations such as insert, delete, replace, move and copy.

5. Define Locator device?

Locator devices are two-dimensional analog-to-digital converters that position a
cursor symbol on the screen by observing the user's movement of the device. The most
common such devices for editing applications are the mouse and the data tablet.

6. What is the function performed in voice input device?

Voice-input devices, which translate spoken words to their textual equivalents, may
prove to be the text input devices of the future. Voice recognizers are currently available for
command input on some systems.

7. What are called tokens?
The lexical analyzer tracks the source program one character at a time by making the
source program into sequence of atomic units is called tokens.
8. Name some of typical tokens.
Identifiers, keywords, constants, operators and punctuation symbols such as commas
and parentheses are typical tokens.
9. What is meant by lexeme?
The character that forms a token is said to be a lexeme.
10. Mention the main disadvantage of interpreter.
The main disadvantage of interpreter is that the execution time of interpreted program
is slower than that of a corresponding compiled object program.
11. What is meant by code optimization?
The code optimization is designed to improve the intermediate code, which helps the
object program to run faster and takes less space.
12. What is error handler?
The error handler is used to check if there is an error in the program. If any error, it
should warn the programmer by instructions to proceed from phase to phase.
35
13. Name some of text editors.

- line editors
- stream editors
- screen editors
- word processors
- structure editors

14. What for debug monitors are used?
Debug monitors are used in obtaining information for localization of errors.
15. Mention the features of word processors.

- moving text from one place to another
- merging of text
- searching
- word replacement

16. What are the phases in performing editing process?

- Traveling phase
- Filtering phase
- Formatting phase
- Editing phase

17. Define traveling phase.
The phase specifies the region of interest. Traveling is achieved using operations
such as next screenful, bottom, find pattern.

18. Filtering phase.
The selection of what is to be viewed and manipulated in given by filtering.
19. Editing phase
In this phase, the target document is altered with the set of operations such as insert,
delete, replace, move and copy.
20. Define user interface?
User interface is one, which allows the user to communicate with the system in order
to perform certain tasks. User interface is generally designed in a computer to make it easier
to use.
21. Define input device?
36
Input device is an electromechanical device, which accepts data from the outside
world and translates them into a form, which the computer can interpret.
22.Define output devices
Output devices the user to view the elements being edited and the results of the editing
operations.
23. What are the methods in Interaction language of a text editor?

- Typing –oriented or text command oriented method
- Function key interfaces
- menu oriented method

24. Define interactive debugging systems.
An interactive debugging system provides programmers with facilities that aid in the
testing and debugging of programs.

- Debugging functions and capabilities
- Relationship with other parts of the system
- User interface criteria.

25. Define editor structure.
The command language processor accepts input from the users input devices and
analyzes the tokens and syntactic structure of the commands.
26. Give the components of editor structure
4 components

- Editing component
- Traveling component
- Viewing component
- Display component

27. What are the basic types of computing environments used in editor's functions?
Editor's function in three basic types of computing environments

i. Time sharing ii. Stand-alone iii. Distributed