INDEX

List of Experiments

S.NO.	Experiments
1	Introduction of System Programming and its Components.
2	Implementation of single pass assembler.
3	Implementation of pass 2 compiler

1. Introduction of System Programming and its Components.

I. INTRODUCTION

System programming is the activity of programming system software. Systems programming aims to produce software which provide services to other software. It is a low level language but programmer writes it in high level language which is not a computer language. To handle this, Compiler and Interpreter convert high level language into machine language. System programming requires great degree of hardware awareness .It is used for creating Operating System and AAA games etc. The main goal is to achieve efficient use of available resources. System programming allows limited facilities. System programming is a way in which the programmer can interact with the system. Programmer will be able to access the system using System programming. Without system programming there is no another way to interact with system. Therefore it's so important. By using System programming we can produce new software& software platforms which will help to provide services to other software.

II. COMPONENTS OF SYSTEM PROGRAMMING

Components of system programming are:

1) Loader
2) Assembler
3) Compiler
4) Macro
5) Interpreter

Loader

Loader plays an important role in system programming. It palaces the program in the main memory and prepare it for execution. Loading program means reading contents of executable file and loading program in the memory and then carryout tasks and prepare them for running, as the loading is complete the operating system starts the program execution by passing control to program code.

Assembler

An assembler is used to convert assembly language program into machine level language. Assembler provides access to Software and Application developers to manage computer hardware and its components. Assembler act as a compiler of assembly language program.

Compiler

A Compiler is program used to convert high level programming language to low level programming language. Compilers are a type of translator that translates digital devices and computer languages.

Further compiler is divided into two types:-

a. Cross Compiler: A cross compiler is a compiler which is capable of generating an executable code for other platform where it resides on another platform.

b. Source to Source Compiler: A source to source compiler accepts an input written in one programming language and produces output for same written in another language.

Macro

A macro is a single line abbreviation for group of statements or block of code. Macro allows programmer to write the short part of the program. A macro can be expanded into no. of instructions.

Interpreter

In System Programming interpreter is a program which directly executes the program without compiling it to the machine language.

The below program shows the use of all components of system programming:

MACRO

ADDITION &arg1,&arg2,&arg3

ADD A,&arg1

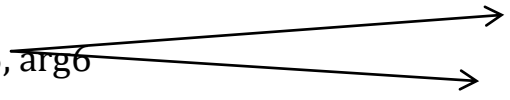
ADD
B,&arg2
ADD
C,&arg3

....

....

MEND

ADDITION arg4,arg5, arg6



ADD A, arg1

ADD B, arg2

ADD C, arg3

…

…

MEND

Macro expansion can be done using macro call. While in macro expansion the sequence of assembly statement are replaced by macro statement. Here MACRO indicates the beginning of the program and ADDITION is the name of macro. Arguments i.e. &arg1,&arg2,&arg3 are used for the expansion of macro. Set of instruction are used that are added with macro name and dummy parameters. Later macro is been called due to which macro expansion takes place. MEND indicates end of the macro program.

Types of system
software are: i)YACC

ii) LEX

i)YACC: YACC stands for "Yet Another Compiler Compiler " it is used to generate code for parser in C. It is an open source program. The output generated is a shift-reduce parser in C. In yacc file user writes its own main function which calls yyparse() and its end in y.tab.c

ii)LEX: Lex is a computer program which generates lexers or scanners. Lexical analyser reads input stream and produces source code as output. Lex and yacc both work internally. Lex source is a table of regular expressions. The table is translated to a program which reads an input stream and then produces source code as output.

2. Implementation of single pass assembler.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
FILE *f1,*f2;
char ch,str[30],str1[10],str2[30],cstr[15];
int i,j,num,q,r;
clrscr();
printf("Enter your assembly instructions\n");
f1=fopen("asin","w");
while(1)
{ ch=getchar();
if(ch=='*')
break;
fputc(ch,f1);
}
fclose(f1);
f1=fopen("asin","r");
f2=fopen("asout","w");
while(1)
{
fgets(str,25,f1);
strncpy(str1,str,3);
```

```
str1[3]='\0';
j=0;
for(i=3;i<strlen(str);i++)
{s
tr2[j]=str[i];
j++;
}s
tr2[j]='\0';
if((strcmp(str1,"lda"))==0)
{
fputs("3a\t",f2);
fputs(str2,f2);
} else if((strcmp(str1,"mov"))==0)
{
fputs("47\n",f2);
} else if((strcmp(str1,"add"))==0)
{
fputs("80\n",f2);
} else if((strcmp(str1,"sub"))==0)
{
fputs("90\n",f2);
} else if((strcmp(str1,"hlt"))==0)
{
fputs("76\n",f2);
break;
} else if((strcmp(str1,"sta"))==0)
{
fputs("32\t",f2);
num=atoi(str2)
q=num/100;
r=num%100;
if(r==0)
fputs("00\t",f2);
else
```

```
fputs(itoa(r,cstr,10),f2);
fputs("\t",f2);
fputs(itoa(q,cstr,10),f2);
fputs("\n",f2);
} else
{
fputs("error\n",f2);
}}
fclose(f1);
fclose(f2);
f2=fopen("asout","r");
printf("\nTHE OBJECT CODE CONTENTS\n");
ch=fgetc(f2);
while(ch!=EOF)
{
putchar(ch);
ch=fgetc(f2);
}
fclose(f2);
getch();
}
```

3.Implementation of pass 2 compiler

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct opc
{
int len;
char mnemonic[10],code[3];
}opcode;
struct opd
{
int address;
char code[10];
}op;
struct source_result
{
int address;
char label[10],instr[10],operand[10];
}res;
struct res
{
int a;
char c[10];
}s;
void main()
{
```

```
FILE *r,*o,*result,*symb;
int i,j,found=0,l;
char s1[10];
clrscr();
r=fopen("re.txt","r");
o=fopen("opcode.txt","r");
result=fopen("output.txt","w");
symb=fopen("symbol.txt","r");
while(!feof(r))
{
fscanf(r,"%d%s%s%s",&res.address,&res.label,&res.instr,&res.operand);
rewind(o);
rewind(symb);
found = 0;
while(!feof(o))
{
fscanf(o,"%d%s%s",&opcode.len,&opcode.mnemonic,&opcode.code);
if(strcmp(res.instr,opcode.mnemonic)==0)
{
op.address=res.address;
strcpy(op.code,opcode.code);
fprintf(result,"%d\t%s",op.address,op.code);
found=1;
break;
}
}
if(found==0)
continue;
while(!feof(symb))
{
fscanf(symb,"%d%s",&s.a,&s.c);
if(strcmp(res.operand,s.c)==0)
{
fprintf(result,"%d\n",s.a);
```

```
break;
}
else if(strcmp(res.operand,"NULL")==0)
{
fprintf(result,"0000");
break;
}
else if(res.operand[0]=='#')
{
strcpy(s1,res.operand);
l=strlen(s1)-1;
for(i=4;i>l;i--)
fprintf(result,"0");
for(j=1;j<=l;j++)
fprintf(result,"%c",s1[j]);
fprintf(result,"\n");
break;
}
}
}
printf("Pass 2 completed!");
fcloseall();
getch();
}
```