Practical 7

Aim: Write a SAL (Simple Assembly Language) program in text file and generate SYMTAB and LITTAB.

Software Required:

Turbo C or Borland C

Knowledge Required:

Algorithm to construct PASS 1 of a two pass Assembler.

THEORY/LOGIC:

This is a Pass 1 of a two pass Assemble. It is used to separate the Symbols, mnemonic opcodes and operand fields. It builds Symbol Table and perform LC processing and construct intermediate code of a given assemble language program.

Data Structures:

Data	Information
OPTAB	A table of mnemonic opcodes and related information.
SYMTAB	Symbol table
LITTAB	A table of literals used in program.

Program:

```
#include<stdio.h>

#include<string.h>

#include<conio.h>

#include<stdlib.h>

#include<ctype.h>

struct MOTtable

{

char Mnemonic[7];

int Class;

char Opcode[3];

};
```

```
struct symboltable
{
char Symbol[8];
int Address;
int Size;
}ST[20];
struct literaltable
{
int Literal;
int Address;
}LT[10];
int PT[20];
struct intermediatecode
{
int LC;
int Code1,Type1;
int Code2,Type2;
int Code3,Type3;
}IC[30];
static struct MOTtable MOT[28]={{"STOP",1,"00"},{"ADD",1,"01"},{"SUB",1,"02"},
{"MULT",1,"03"},{"MOVER",1,"04"},{"MOVEM",1,"05"},
{"COMP",1,"06"},{"BC",1,"07"},{"DIV",1,"08"},
{"READ",1,"09"},{"PRINT",1,"10"},
{"START",3,"01"},{"END",3,"02"},{"ORIGIN",3,"03"},
{"EQU",3,"04"},{"LTORG",3,"05"},
{"DS",2,"01"},{"DC",2,"02"},
{"AREG",4,"01"},{"BREG",4,"02"},{"CREG",4,"03"},
{"EQ",5,"01"},{"LT",5,"02"},{"GT",5,"03"},{"LE",5,"04"},
{"GE",5,"05"},{"NE",5,"06"},{"ANY",5,"07"}};
```

```
int nMOT=28; //Number of entries in MOT
int LC=0; //Location counter
int iPT; //Index of next entry in Pool Table
int iLT=0; //Index of next entry in Literal Table
int iST=0; //Index of next entry in Symbol Table
int iIC=0; //Index of next entry in intermediate code table
int searchST(char symbol[])
{
int i;
for(i=0;i<iST;i++)
if(strcmp(ST[i].Symbol,symbol)==0)
return(i);
return(-1);
}
int searchMOT(char symbol[])
{
int i;
for(i=0;i<nMOT;i++)
if(strcmp(MOT[i].Mnemonic,symbol)==0)
return(i);
return(-1);
}
int insertST( char symbol[],int address,int size)
{
strcpy(ST[iST].Symbol,symbol);
ST[iST].Address=address;
ST[iST].Size=size;
iST++;
return(iST-1);
```

```
}
void imperative(); //Handle an executable statement
void declaration(); //Handle a declaration statement
void directive(); //Handle an assembler directive
void print_symbol(); //Display symbol table
void print_pool(); //Display pool table
void print_literal(); //Display literal table
void print_opcode(); //Display opcode table
void intermediate(); //Display intermediate code
char s1[8],s2[8],s3[8],label[8];
void LTORG(); //Handle LTORG directive
void DC(); //Handle declaration statement DC
void DS(); //Handle declaration statement DS
void START(); //Handle START directive
int tokencount; //total number of words in a statement
int main()
{
char file1[40],nextline[80];
int len,i,j,temp,errortype;
FILE *ptr1;
// clrscr();
printf("\nenter Source file name:");
gets(file1);
ptr1=fopen(file1,"r");
while(!feof(ptr1))
{
//Read a line of assembly program and remove special characters
i=0;
nextline[i]=fgetc(ptr1);
```

```
while(nextline[i]!='\n'&& nextline[i]!=EOF )
{
if(!isalnum(nextline[i]))
nextline[i]=' ';
else
nextline[i]=toupper(nextline[i]);
i++;
nextline[i]=fgetc(ptr1);
}
nextline[i]='\0';
//if the nextline is an END statement
sscanf(nextline,"%s",s1); //read from the nextline in s1
if(strcmp(s1,"END")==0)
break;
//if the nextline contains a label
if(searchMOT(s1)==-1)
{
if(searchST(s1)==-1)
insertST(s1,LC,0);
//separate opcode and operands
tokencount=sscanf(nextline,"%s%s%s%s",label,s1,s2,s3);
tokencount--;
}
else
//separate opcode and operands
tokencount=sscanf(nextline,"%s%s%s",s1,s2,s3);
if(tokencount==0)//blank line
continue;//goto the beginning of the loop
i=searchMOT(s1);
```

```
if(i==-1)
{
printf("\nWrong Opcode .... %s",s1);
continue;
}
switch (MOT[i].Class)
{
case 1: imperative();break;
case 2: declaration();break;
case 3: directive();break;
default: printf("\Wrong opcode ...%s",s1);
break;
}
}
LTORG();
print_symbol();
getch();
print_pool();
getch();
print_literal();
getch();
print_opcode();
getch();
intermediate();
getch();
}
void imperative()
{
int index;
```

```
index=searchMOT(s1);
IC[iIC].Type1=IC[iIC].Type2=IC[iIC].Type3=0; //intialize
IC[iIC].LC=LC;
IC[iIC].Code1=index;
IC[iIC].Type1=MOT[index].Class;
LC=LC+1;
if(tokencount>1)
{
index=searchMOT(s2);
if(index != -1)
{
IC[iIC].Code2=index;
IC[iIC].Type2=MOT[index].Class;
}
else
{ //It is a variable
index=searchST(s2);
if(index==-1)
index=insertST(s2,0,0);
IC[iIC].Code2=index;
IC[iIC].Type2=7; //VALUE 7 IS FOR VARIABLES
}
}
if(tokencount>2)
{
if(isdigit(*s3))
{
LT[iLT].Literal=atoi(s3);
IC[iIC].Code3=iLT;
```

```
IC[iIC].Type3=8; //VALUE 8 IS FOR LITERAL TYPE
iLT++;
}
else
{
index=searchST(s3);
if(index==-1)
index=insertST(s3,0,0);
IC[iIC].Code3=index;
IC[iIC].Type3=7; //VALUE 7 IS FOR VARIABLES
}
}
iIC++;
}
void declaration()
{
if(strcmp(s1,"DC")==0)
{
DC();
return;
}
if(strcmp(s1,"DS")==0)
DS();
}
void directive()
{
if(strcmp(s1,"START")==0)
{
START();
```

```
return;
}
if(strcmp(s1,"LTORG")==0)
LTORG();
}
void LTORG()
{
int i,index;
for(i=PT[iPT];i<iLT;i++)
{
LT[i].Address=LC;
index=searchMOT("DC");
IC[iIC].Type1=IC[iIC].Type2=IC[iIC].Type3=0; //intialize
IC[iIC].LC=LC;
IC[iIC].Code1=index;
IC[iIC].Type1=MOT[index].Class;
IC[iIC].Type2=6; //6 IS TYPE FOR CONSTANTS
IC[iIC].Code2=LT[i].Literal;
LC=LC+1;
iIC++;
}
iPT++;
PT[iPT]=iLT;
}
void DC()
{
int index;
index=searchMOT(s1);
IC[iIC].Type1=IC[iIC].Type2=IC[iIC].Type3=0; //intialize
```

```
IC[iIC].LC=LC;
IC[iIC].Code1=index;
IC[iIC].Type1=MOT[index].Class;
IC[iIC].Type2=6; //6 IS TYPE FOR CONSTANTS
IC[iIC].Code2=atoi(s2);
index=searchST(label);
if(index==-1)
index=insertST(label,0,0);
ST[index].Address=LC;
ST[index].Size=1;
LC=LC+1;
iIC++;
}
void DS()
{
int index;
index=searchMOT(s1);
IC[iIC].Type1=IC[iIC].Type2=IC[iIC].Type3=0; //intialize
IC[iIC].LC=LC;
IC[iIC].Code1=index;
IC[iIC].Type1=MOT[index].Class;
IC[iIC].Type2=6; //6 IS TYPE FOR CONSTANTS
IC[iIC].Code2=atoi(s2);
index=searchST(label);
if(index==-1)
index=insertST(label,0,0);
ST[index].Address=LC;
ST[index].Size=atoi(s2);
LC=LC+atoi(s2);
```

```
iIC++;
}
void START()
{
int index;
index=searchMOT(s1);
IC[iIC].Type1=IC[iIC].Type2=IC[iIC].Type3=0; //intialize
IC[iIC].LC=LC;
IC[iIC].Code1=index;
IC[iIC].Type1=MOT[index].Class;
IC[iIC].Type2=6; //6 IS TYPE FOR CONSTANTS
IC[iIC].Code2=atoi(s2);
LC=atoi(s2);
iIC++;
}
void intermediate()
{ int i;
char decode[9][3]={" ","IS","DL","AD","RG","CC","C","S","L"};
printf("\n\nIntermediate Code :");
for(i=0;i<iIC;i++)
{
printf("\n%3d) (%s,%2s)",IC[i].LC,decode[IC[i].Type1]
,MOT[IC[i].Code1].Opcode);
if(IC[i].Type2!=0)
{
if(IC[i].Type2<6)
printf(" (%s,%2s)",decode[IC[i].Type2]
,MOT[IC[i].Code2].Opcode);
else
```

```
printf(" (%s,%2d)",decode[IC[i].Type2]
,IC[i].Code2);
}
if(IC[i].Type3!=0)
printf(" (%s,%2d)",decode[IC[i].Type3]
,IC[i].Code3);
}
}
void print_symbol()
{
int i;
printf("\*******symbol table ***********\n");
for(i=0;i<iST;i++)
printf("\n%10s %3d %3d",ST[i].Symbol,ST[i].Address
,ST[i].Size);
}
void print_pool()
{
int i;
printf("\npool table *********\n");
for(i=0;i<iPT;i++)
printf("\n%d",PT[i]);
}
void print_literal()
{
int i;
printf("\nliteral table*******\n");
for(i=0;i<iLT;i++)
printf("\n%5d\t%5d",LT[i].Literal,LT[i].Address);
```

```
}
void print_opcode()
{
int i;
printf("\nopcode table **********");
for(i=0;i<nMOT;i++)
if(MOT[i].Class==1)
printf("\n%6s %2s",MOT[i].Mnemonic,MOT[i].Opcode);
}
```

Output:

Screen shot of out

Note: Same format will be applicable to all other experiments. 1st page title 2nd page experiment.

Imp. Note: (Remove all red text lines)