**EXPNO :2 DESIGN OF SINGLE PASS ASSEMBLER**

**AIM:**

To write a C program to implement the design of single pass assembler.

**ALGORITHM:**

I.code\_area\_address=address of code\_area;

loc\_cntr=0

pooltab\_ptr=1

tittab\_ptr=l

symtab\_ptr=1

POOLTAB[1]=1

2.While next statement is not an END statement

a)clear machine\_code buffer

b)if label is present then

i) this label=symbol in label field

ii)Enter(this\_label,loc \_cntr)in SYMTAB.

iii)symtab\_ptr =symtab\_ptr+1

c)if LTORG statement

i)process literals in LITTAB[POOLTAB [pooltab\_ptr]]......LITTAB[littab\_ptr-1]ie.

assemble the literals in machine\_code\_buffer.

ii)size= size of memory area required for literals.

iii)pooltab\_ptr=pooltab\_ptr+1

iv)POOLTAB [pooltab\_ptr)=littab\_ptr

d)if a START or ORIGIN statement

i) loc\_cntr=value specified in operand field size=0

e)if a declaration statement

i)ifa DC statement then assemble the constant in machine\_code\_buffer

ii)size=size of memory area required by DC/DS

f) if an imperative statement

i) if operand is a literal then

this\_literal=literal in operand field

LITTAB[littab \_ptr]=this\_literal

Littab\_ptr=littab\_ptr+1

ii)if operand is a symbol then

this\_symbol=operand

if this\_symbol is present in the SYMTAB

this\_address=address of the symbol

assemble instruction in machine\_code \_buffer

else

enter <loc\_cntr,this\_symbol>in Tll(table of incomplete instructions)

assemble instruction in machine\_code\_buffer with memory operand empty.

iii) size=size of the instruction

g)if size !=0 then

i)move contents of machine \_code\_buffer to the address code\_area\_address+loc\_cntr

ii)loc\_entr=loc\_cntr+size

3)Processing of END statement

i)perform 2(c)

ii)perform backpatching

iii)write code\_area into output file.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<string.h>

struct menumonic\_table

{

char op\_code[7];

char class1[3];

char info[3];

}nem[20];

int checkreg(char var1[])

{

if(strcmp(var1,"AREG")==0)

return 1;

if(strcmp(var1,"BREG")==0)

return 2;

if(strcmp(var1,"CREG")==0)

return 3;

else

return 0;

}

char symb[6][10],symb1[14][11];

int add[6];

int insert(char var[])

{

static int i=1;

strcpy(symb1[i],var);

i+=1;

return 0;

}

int checksym(char var[])

{

static int i=1;

int j;

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

{

if(strcmp(symb[j],var)==0)

return j;

}

strcpy(symb[i],var);

i++;

return 0;

}

int search(char var2[])

{

int i;

for(i=0;i<19;i++)

{

if(strcmp(var2,nem[i].op\_code)==0)

return i;

}

return 20;

}

main()

{

int i;

strcpy(nem[0].op\_code,"ADD");

strcpy(nem[0].class1,"IS");

strcpy(nem[0].info,"01");

strcpy(nem[1].op\_code,"SUB");

strcpy(nem[1].class1,"IS");

strcpy(nem[1].info,"02");

strcpy(nem[2].op\_code,"MULT");

strcpy(nem[2].class1,"IS");

strcpy(nem[2].info,"03");

strcpy(nem[3].op\_code,"MOVER");

strcpy(nem[3].class1,"IS");

strcpy(nem[3].info,"04");

strcpy(nem[4].op\_code,"MOVEM");

strcpy(nem[4].class1,"IS");

strcpy(nem[4].info,"05");

strcpy(nem[5].op\_code,"COMP");

strcpy(nem[5].class1,"IS");

strcpy(nem[5].info,"06");

strcpy(nem[6].op\_code,"STOP");

strcpy(nem[6].class1,"IS");

strcpy(nem[6].info,"00");

strcpy(nem[7].op\_code,"BC");

strcpy(nem[7].class1,"IS");

strcpy(nem[7].info,"07");

strcpy(nem[8].op\_code,"DIV");

strcpy(nem[8].class1,"IS");

strcpy(nem[8].info,"08");

strcpy(nem[9].op\_code,"READ");

strcpy(nem[9].class1,"IS");

strcpy(nem[9].info,"09");

strcpy(nem[10].op\_code,"PRINT");

strcpy(nem[10].class1,"IS");

strcpy(nem[10].info,"10");

strcpy(nem[11].op\_code,"START");

strcpy(nem[11].class1,"AD");

strcpy(nem[11].info,"01");

strcpy(nem[12].op\_code,"END");

strcpy(nem[12].class1,"AD");

strcpy(nem[12].info,"02");

strcpy(nem[13].op\_code,"LTROG");

strcpy(nem[13].class1,"AD");

strcpy(nem[13].info,"03");

strcpy(nem[14].op\_code,"EQU");

strcpy(nem[14].class1,"AD");

strcpy(nem[14].info,"04");

strcpy(nem[15].op\_code,"ORIGIN");

strcpy(nem[15].class1,"AD");

strcpy(nem[15].info,"05");

strcpy(nem[16].op\_code,"DS");

strcpy(nem[16].class1,"DL");

strcpy(nem[16].info,"02");

strcpy(nem[17].op\_code,"DC");

strcpy(nem[17].class1,"DL");

strcpy(nem[17].info,"01");

strcpy(nem[18].op\_code,"LE");

strcpy(nem[18].class1,"CS");

strcpy(nem[18].info,"02");

FILE \*f1,\*f2,\*f3;

int j=0,k=0,lc,sy,no,sym=0,reg,num,mac[20][10],add1[13];

char var[256],var1[4][10];

char delim[] = " ";

static int m=1,n=1;

f1=fopen("input.txt","r");

f2=fopen("output.txt","w");

f3=fopen("symtab.txt","w");

while(k<18)

{

fgets(var,sizeof(var),f1);

i=0;

char \*ptr = strtok(var, delim);

while(ptr != NULL)

{

strcpy(var1[i],ptr);

ptr = strtok(NULL, delim);

i++;

}

j=i;

strtok(var1+0,"\n");

no=search(var1+0);

if(strcmp(var1+0,"START")==0)

{

lc=atoi(var1+1);

}

if(strcmp(nem[no].info,"09")==0)

{

insert(var1+1);

add1[n]=lc;

n++;

mac[m][0]=lc;

mac[m][1]=atoi(nem[no].info);

mac[m][2]=0;

lc+=1;

m+=1;

}

if(strcmp(nem[no].info,"04")==0||strcmp(nem[no].info,"05")==0||strcmp(nem[no].info,"06")==0)

{

insert(var1+2);

add1[n]=lc;

n++;

reg=checkreg(var1+1);

mac[m][0]=lc;

mac[m][1]=atoi(nem[no].info);

mac[m][2]=reg;

//fprintf(f2,"%s %s %d s %d\n",nem[no].class1,nem[no].info,reg,sy);

lc+=1;

m+=1;

}

if(no==20)

{

int ad;

no=search(var1+1);

sy=checksym(var1+0);

add[1]=lc;

reg=checkreg(var1+2);

if(strcmp(nem[no].class1,"DL")==0)

{

ad=checksym(var1+0);

add[ad]=lc;

mac[m][0]=lc;

mac[m][1]=11;

mac[m][2]=1;

mac[m][3]=111;

if(strcmp(nem[no].info,"01")==0)

{

mac[m][1]=00;

mac[m][2]=0;

mac[m][3]=001;

}

m+=1;

}

else

{

insert(var1+3);

add1[n]=lc;

n++;

mac[m][0]=lc;

mac[m][1]=atoi(nem[no].info);

mac[m][2]=reg;

m+=1;

}

lc+=1;

}

if(strcmp(var1+0,"STOP")==0)

{

mac[m][0]=lc;

mac[m][1]=00;

mac[m][2]=0;

mac[m][3]=000;

lc+=1;

m+=1;

}

if(strcmp(nem[no].info,"01")==0&&strcmp(nem[no].class1,"IS")==0)

{

insert(var1+2);

add1[n]=lc;

n++;

reg=checkreg(var1+1);

mac[m][0]=lc;

mac[m][1]=atoi(nem[no].info);

mac[m][2]=reg;

//printf("(%s,%s) (%d) (s,%d)\n",nem[no].class1,nem[no].info,reg,sy);

//fprintf(f2,"%s %s %d s %d\n",nem[no].class1,nem[no].info,reg,sy);

lc+=1;

m+=1;

}

if(strcmp(nem[no].info,"07")==0)

{

num=search(var1+1);

mac[m][0]=lc;

mac[m][1]=atoi(nem[no].info);

mac[m][2]=atoi(nem[num].info);

mac[m][3]=add[1];

lc+=1;

m+=1;

}

if(strcmp(nem[no].info,"10")==0)

{

insert(var1+1);

add1[n]=lc;

n++;

reg=checkreg(var1+1);

mac[m][0]=lc;

mac[m][1]=atoi(nem[no].info);

mac[m][2]=reg;

lc+=1;

m++;

}

k++;

}

for(i=1;i<=11;i++)

{

int l=\*(add1+i)%100;

for(j=1;j<=5;j++)

{

strtok(symb[j],"\n");

strtok(symb1[i],"\n");

if(strcmp(symb1[i],symb[j])==0)

{

mac[l][3]=add[j];

}

}

}

for(i=1;i<=16;i++)

{

printf("%d %d %d %d\n",mac[i][0],mac[i][1],mac[i][2],mac[i][3]);

}

}

**INPUT:**

**input.txt:**

START 101

READ N

MOVER BREG ONE

MOVEM BREG TERM

AGAIN MULT BREG TERM

MOVER CREG TERM

ADD CREG ONE

MOVEM CREG TERM

COMP CREG N

BC LE AGAIN

MOVEM BREG RESULT

PRINT RESULT

STOP

N DS 1

RESULT DS 1

ONE DC 1

TERM DS 1

END

**OUTPUT:**

**symtab.txt:**

N 113

ONE 115

TERM 116

AGAIN 104

RESULT 114

**output.txt:**

101 9 0 113

102 4 2 115

103 5 2 116

104 3 2 116

105 4 3 116

106 1 3 115

107 5 3 116

108 6 3 113

109 7 2 104

110 5 2 114

111 10 0 114

112 0 0 0

113 11 1 111

114 11 1 111

115 0 0 1

116 11 1 111

**RESULT:**

Thus,design of single pass assembler is executed and output is verified and displayed successfully.