**ASSIGNMENT: 1**

**Problem statement:**

Write a program to implement I pass assembler.( For hypothetical instruction set from Dhamdhere)

a. Consider following cases only (Literal processing not expected)

b. Forward references

c. DS and DC statement

d. START, EQU

e. Error handling: symbol used but not defined, invalid instruction/register etc.

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

struct MOTtable

{

char Mnemonic[6];

int Class;

char Opcode[3];

};

struct symboltable

{

char Symbol[8];

int Address;

int Size;

}ST[20];

struct intermediatecode

{

int LC;

int Code1,Type1;

int Code2,Type2;

int Code3,Type3;

}IC[30];

static struct MOTtable MOT [29]={{"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"},{"DREG",4,"04"},

{"EQ",5,"01"},{"LT",5,"02"},{"GT",5,"03"},{"LE",5,"04"},

{"GE",5,"05"},{"NE",5,"06"},{"ANY",5,"07"}};

int nMOT=29; **//Number of entries in MOT**

int LC=0; **//Location counter**

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 cnt=0; **//line number**

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\_opcode(); **//Display opcode table**

void intermediate(); **//Display intermediate code**

char s1[8],s2[8],s3[8],label[8];

void DC(); **//Handle declaration statement DC**

void DS(); **//Handle declaration statement DS**

void START(); **//Handle START directive**

void ORIGIN(); **//handle ORIGIN directive**

void EQU(); **//handle EQU directive**

int tokencount; **//total number of words in a statement**

int main()

{

char file1[40],nextline[80];

int len,i,j,var,index1;

FILE \*ptr1;

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);

cnt++;

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)

{

**//separate opcode and operands**

tokencount=sscanf(nextline,"%s%s%s%s",label,s1,s2,s3);

if(tokencount==1)

{

var=atoi(s1);

continue;

}

**//check the mnemonic**

if(tokencount==2)

{

if(searchST(label)==-1)

insertST(label,LC,0);

}

if(tokencount==3)

{

if(strcmp(s1,"DS")!=0 && strcmp(s1,"DC")!=0 && strcmp(s1,"EQU")!=0)

tokencount=sscanf(nextline,"%s%s%s",s1,s2,s3);

else

{

if(searchST(label)==-1)

insertST(label,LC,0);

}

}

if(tokencount==4)

if(searchST(label)==-1)

insertST(label,LC,0);

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

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("\nError at line %d :%s\n",cnt,label);

continue;

}

switch (MOT[i].Class)

{

case 1: imperative();

break;

case 2: declaration();

break;

case 3: directive();

break;

default: printf("\nError at line %d :%s\n",cnt,label);

continue;

}

}

print\_symbol();

print\_opcode();

intermediate();

return 0;

}

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;

if(tokencount>1)

{

index=searchMOT(s2);

if(index != -1)

{

IC[iIC].Code2=index;

IC[iIC].Type2=MOT[index].Class;

}

else if(index == -1)

{

printf("Error at line %d: %s\n",cnt,s2);

}

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>3)

{

index=searchST(label);

if(index!=-1)

{

ST[index].Address=LC;

ST[index].Size=1;

}

}

LC=LC+1;

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,"EQU")==0)

{

EQU();

return;

}

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

ORIGIN();

}

void ORIGIN()

{

char \*p=NULL;

int add,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;

p=strtok(s2,"+");

index=searchST(p);

add=ST[index].Address;

LC=add+atoi(s3);

iIC++;

}

void EQU()

{

int i1,i2,add, 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;

i1=searchST(s2);

IC[iIC].Code2=i1;

IC[iIC].Type2=7; **//VALUE 7 IS FOR VARIABLES**

add=ST[i1].Address;

i2=searchST(label);

ST[i2].Address=add;

iIC++;

LC=LC+1;

}

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("\n--------------SYMBOL TABLE--------------\n");

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

printf("\n%10s %3d ",ST[i].Symbol,ST[i].Address);

}

void print\_opcode()

{

int i;

printf("\n------------OPCODE TABLE------------");

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

if(MOT[i].Class==1)

printf("\n%6s %2s",MOT[i].Mnemonic,MOT[i].Opcode);

}

/\*

**OUTPUT:**

[Aditya@localhost ~]$ gcc 2.c

/tmp/ccdauMLz.o: In function `main':

2.c:(.text+0x114): warning: the `gets' function is dangerous and should not be used.

[Aditya@localhost ~]$ ./a.out

Enter Source file name:prog.txt

**--------------SYMBOL TABLE--------------**

LOOP 202

NEXT1 206

A 209

B 211

NEXT2 202

**------------OPCODE TABLE------------**

STOP 00

ADD 01

SUB 02

MULT 03

MOVER 04

MOVEM 05

COMP 06

BC 07

DIV 08

READ 09

PRINT 10

**Intermediate Code :**

0) (AD,01) (C,200)

200) (IS,04) (RG,01)

201) (IS,05) (RG,01)

202) (IS,04) (RG,01)

203) (IS,04) (RG,03)

204) (IS,01) (RG,03)

205) (IS,01)

206) (IS,02) (RG,01)

207) (AD,03)

208) (IS,03) (RG,03)

209) (DL,01) (C, 2)

211) (DL,02) (C, 3)

212) (AD,04) (S, 0)

**//INPUT PROGRAM**

START 200

MOVER AREG,='5'

MOVEM AREG,A

LOOP MOVER AREG,A

MOVER CREG,B

ADD CREG,='1'

NEXT1 SUB AREG,='1'

ORIGIN LOOP+6

MULT CREG,B

A DS 2

B DC 3

NEXT2 EQU LOOP

END

**//ERROR IN PROGRAM**

START 200

MOVER AREG,='5'

MOVE AREG,A

LOOP MOVER AREG,A

MOVER CREG,B

AD CREG,='1'

NEXT1 SUB AREG,='1'

ORIGI LOOP+6

MULT CREG,B

A DS 2

B DC 3

NEXT2 EQU LOOP

END

[Aditya@localhost ~]$ gcc 2.c

/tmp/ccS5PnS5.o: In function `main':

2.c:(.text+0x114): warning: the `gets' function is dangerous and should not be used.

[Aditya@localhost ~]$ ./a.out

Enter Source file name:prog.txt

Error at line 3 :MOVE

Error at line 6 :AD

Error at line 7 :AD

Error at line 9 :ORIGI

**--------------SYMBOL TABLE--------------**

LOOP 201

NEXT1 203

A 205

B 207

NEXT2 201

**------------OPCODE TABLE------------**

STOP 00

ADD 01

SUB 02

MULT 03

MOVER 04

MOVEM 05

COMP 06

BC 07

DIV 08

READ 09

PRINT 10

**Intermediate Code :**

0) (AD,01) (C,200)

200) (IS,04) (RG,01)

201) (IS,04) (RG,01)

202) (IS,04) (RG,03)

203) (IS,02) (RG,01)

204) (IS,03) (RG,03)

205) (DL,01) (C, 2)

207) (DL,02) (C, 3)

208) (AD,04) (S, 0)

\*/