Assembly.h

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<ctype.h>

const char regs[4][8]={"AREG","BREG","CREG","DREG"};

const char impers[11][8]= {"STOP","ADD","SUB","MULT","MOVER","MOVEM","COMP","BC","DIV","READ","PRINT"};

const char assd[5][8]= {"START","END","EQU","LTORG","ORIGIN"};

const char cond[6][8]={"LT","LE","EQ","GE","GT","ANY"};

const char dec[2][8]={"DS","DC"};

int imparative(char s[])

{

int i;

for(i=0;i<11;i++)

{

if(strcmp(s,impers[i])==0)

return 1;

}

return 0;

}

int declarative(char s[])

{

int i;

for(i=0;i<2;i++)

{

if(strcmp(s,dec[i])==0)

return 1;

}

return 0;

}

int ass\_dir(char s[])

{

int i;

for(i=0;i<5;i++)

{

if(strcmp(s,assd[i])==0)

return 1;

}

return 0;

}

int cond\_code(char s[])

{

int i;

for(i=0;i<6;i++)

{

if(strcmp(s,cond[i])==0)

return 1;

}

return 0;

}

int reg\_code(char s[])

{

int i;

for(i=0;i<4;i++)

{

if(strcmp(s,regs[i])==0)

return 1;

}

return 0;

}

int isnum(char s[])

{

int i;

for(i=0;s[i]!='\0';i++)

if(!isdigit(s[i]))

return 0;

return 1;

}

#include"assembly.h"

int main(int argc, char \*argv[])

{

if(argc!=2)

{

printf("\nPlease enter the valid arguments!\n");

exit(0);

}

FILE \*fp;

int n,i,x1,x2,x3,x4,x5,l=0;

int f1=0,f2=0;

char instr[50],str[50];

fp=fopen(argv[1],"r");

if(fp==NULL)

{

printf("\nFile could not open!\n");

exit(0);

}

while(fgets(instr,50,fp)!=NULL)

{

n=0;

char \*s=instr;

char \*str1;

char t1[10]={ },t2[10]={ },t3[10]={ },t4[10]={ };

l++;

while((str1=strtok\_r(s,"\n ,\t\0",&s)))

{

n++;

switch(n)

{

case 1: strcpy(t1,str1);

break;

case 2: strcpy(t2,str1);

break;

case 3: strcpy(t3,str1);

break;

case 4: strcpy(t4,str1);

break;

default:printf("\nInvalid Statement!\n");

break;

}

}

switch(n)

{

case 1:

if(strcmp(t1,"STOP")==0 || strcmp(t1,"LTORG")==0 || strcmp(t1,"END")==0)

{

printf("\nLine %d\t(%s)",l,t1);

}

else

{

printf("\n%s\tINVALID MNEMONIC INSTRUCTION",t1);

printf("\nSHOULD BE STOP,END OR LTORG\n");

}

break;

case 4:

printf("\nLine %d \t(%s %s %s,%s)",l,t1,t2,t3,t4);

if(imparative(t1)==1 || declarative(t1)==1 || ass\_dir(t1)==1 || cond\_code(t1)==1 || reg\_code(t1)==1)

{

printf("\n%s\tINVALID SYMBOLIC NAME",t1);

}

x1=imparative(t2);

x2=declarative(t2);

x3=ass\_dir(t2);

x4=cond\_code(t2);

x5=reg\_code(t2);

if(x2==1 || x3==1 || x4==1 || x5==1 || strcmp(t2,"STOP")==0 || strcmp(t2,"READ")==0 || strcmp(t2,"PRINT")==0)

{

printf("\n%s\tINVALID MNEMONIC INSTRUCTION",t2);

}

if(reg\_code(t3)==0)

{

printf("\n%s\tINVALID REGISTER OPERAND",t3);

}

x1=imparative(t4);

x2=declarative(t4);

x3=ass\_dir(t4);

x4=cond\_code(t4);

x5=reg\_code(t4);

if(x1==1 || x2==1 || x3==1 || x4==1 || x5==1)

{

printf("\n%s\tINVALID SYMBOLIC NAME",t4);

}

break;

case 2:

printf("\nLine %d \t(%s %s)",l,t1,t2);

x1=imparative(t1);

x2=declarative(t1);

x3=ass\_dir(t1);

x4=cond\_code(t1);

x5=reg\_code(t1);

if(strcmp(t2,"STOP")==0)

{

if(x1==1 || x2==1 || x3==1 || x4==1 || x5==1 || isnum(t1))

{

printf("\n(%s)\tINVALID SYMBLOIC NAME\n",t1);

}

}

else if(strcmp(t1,"START")==0 || strcmp(t1,"ORIGIN")==0 || strcmp(t1,"END")==0)

{

if(atoi(t2)>999 || atoi(t2)<0)

{

printf("\n(%s)\tINVALID MEMORY ADDRESS\n",t2);

}

else if(strcmp(t1,"START")==0 && isnum(t2)==0)

{

printf("\n(%s)\tINVALID MEMORY ADDRESS\n",t2);

}

else if(strcmp(t1,"ORIGIN")==0 &&(imparative(t2) || declarative(t2) || ass\_dir(t2) || cond\_code(t2) || reg\_code(t2)))

{

printf("\n(%s)\tINVALID SYMBLOIC NAME\n",t2);

}

else if(imparative(t2) || declarative(t2) || ass\_dir(t2) || cond\_code(t2) || reg\_code(t2))

{

printf("\n(%s)\tINVALID MENEMONIC INSTRUCTION\n",t2);

}

}

else if(strcmp(t1,"READ")==0 || strcmp(t1,"PRINT")==0)

{

if(imparative(t2) || declarative(t2) || ass\_dir(t2) || cond\_code(t2) || reg\_code(t2))

{

printf("\n(%s)\tINVALID SYMBLOIC NAME\n",t2);

}

}

else

{

printf("\n(%s)\tINVALID MENEMONIC INSTRUCTION\n",t1);

}

break;

case 3:

printf("\nLine %d\t(%s %s %s)",l,t1,t2,t3);

x1=imparative(t1);

x2=declarative(t1);

x3=ass\_dir(t1);

x4=cond\_code(t1);

x5=reg\_code(t1);

if(strcmp(t2,"EQU")==0 || strcmp(t2,"READ")==0 || strcmp(t2,"PRINT")==0 || declarative(t2))

{

if(x1==1 || x2==1 || x3==1 || x4==1 || x5==1 || isnum(t1))

{

printf("\n(%s)\tINVALID SYMBLOIC NAME\n",t1);

}

else if(strcmp(t2,"DS")==0)

{

if(isnum(t3)==0)

{

printf("\n(%s)\tINVALID DECLARATIVE STATEMENT",t3);

}

}

else if(strcmp(t2,"DC")==0)

{

char cons[10];

if(t3[0]=='\'' && t3[strlen(t3)-1]=='\'')

{

for(i=1;i<strlen(t3)-1;i++)

{

cons[i-1]=t3[i];

}

if(isnum(cons))

{

printf("\n(%s)\tINVALID DECLARATIVE STATEMENT",t3);

}

}

else

printf("\n(%s)\tINVALID DECLARATIVE STATEMENT",t3);

}

else if(strcmp(t2,"EQU")==0)

{

if(atoi(t2)>999 || atoi(t2)<0)

{

printf("\n(%s)\tINVALID MEMORY ADDRESS\n",t3);

}

else if(imparative(t3) || declarative(t3) || ass\_dir(t3) || cond\_code(t3) || reg\_code(t3))

{

printf("\n(%s)\tINVALID SYMBLOIC NAME\n",t3);

}

}

else if(strcmp(t2,"READ")==0 || strcmp(t2,"PRINT")==0)

{

if(imparative(t3) || declarative(t3) || ass\_dir(t3) || cond\_code(t3) || reg\_code(t3) || isnum(t3))

{

printf("\n(%s)\tINVALID SYMBLOIC NAME\n",t3);

}

}

}

else if(x1==1)

{

if(strcmp(t1,"BC")==0)

{

if(cond\_code(t2)==0)

{

printf("\n(%s)\tINVALID CONDITION CODE\n",t2);

}

}

else if(reg\_code(t2)==0)

{

printf("\n(%s)\tINVALID REGISTER OPERAND\n",t2);

}

if(imparative(t3) || declarative(t3) || ass\_dir(t3) || cond\_code(t3) || reg\_code(t3) || isnum(t3))

{

printf("\n(%s)\tINVALID SYMBLOIC NAME\n",t3);

}

}

else

{

printf("\n\tINVALID MENEMONIC INSTRUCTION\n",t1);

}

break;

}

}

//printf("%s \t %s \t %s \t %d\n",t1,t2,t3,n);

printf("\n");

fclose(fp);

}

START 1000

REA A

READ B

MOVER AREG,A

ADD AREG,BREG

MOVEM AREG,C

RINT C

STOP

A DS 1

B DS 1

C DS 1

EN