과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 2>>

**서강대학교 컴퓨터공학과**

**[20151561]**

**[신용하]**

목 차

1. 프로그램 개요 3

2. 프로그램 설명 3

2.1 프로그램 흐름도 3

3. 모듈 정의 3

3.1 모듈 이름 : help() 3

3.1.1 기능 3

3.1.2 사용 변수 3

3.2 모듈 이름: type() 4

3.2.1 기능 4

3.2.2 사용 변수 4

3.3 모듈이름: isreg(char \*a) 4

3.3.1 기능 4

3.3.2 사용변수 4

3.4 모듈이름: assemble() 4

3.4.1 기능 4

3.4.2 사용변수 4

3.5 모듈이름: putsymnode(char\* sym, int add) 5

3.5.1 기능 5

3.5.2 사용변수 5

3.6 모듈이름: symbol() 5

3.6.1 기능 5

3.6.2 사용변수 5

4. 전역 변수 정의 5

4.1 objnametmp[50], lstnametmp[50] 5

4.2 struct symnode, struct symnode\* symtab 5

4.3 struct asmnode 5

5. 코드 6

# 프로그램 개요

1MB의 메모리와 opcode table을 가진 shell environment를 직접 구현한 프로그램이다. 사용자는 shell, memory, opcode에 관련된 여러 command를 입력하여 기능을 수행할 수 있다.

# 프로그램 설명

## 프로그램 흐름도

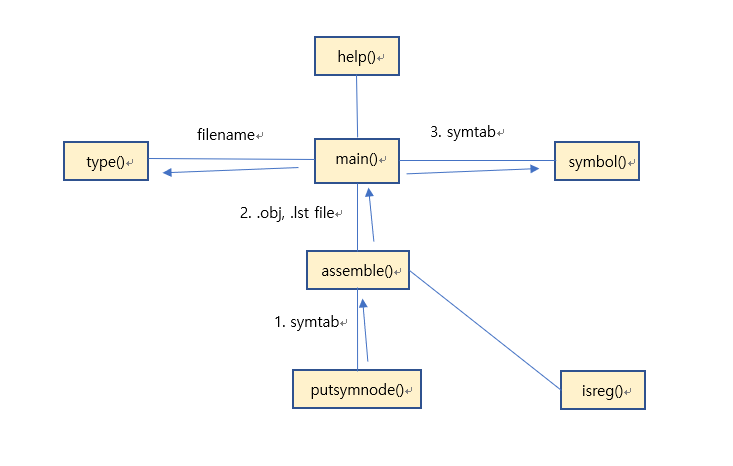


그림 1> 프로그램 흐름도

# 모듈 정의

## 모듈 이름 : help()

### 기능

사용자에게 입력할 수 있는 command들을 보여준다.

assemble filename, type filename, symbol이 추가되었다.

### 사용 변수

없음

## 모듈 이름: type()

### 기능

사용자에게 filename을 입력 받아 그 파일이 존재한다면 파일의 내용을 출력해준다.

### 사용 변수

char name[500] – filename을 저장하는 변수

int i – 반복문을 위한 변수

FILE\* fp – 입력 받은 이름의 파일을 열기 위한 변수

DIR\* dp – 현재 디렉토리의 파일들에 접근하기 위한 변수

char ch – 파일의 내용을 한 글자씩 출력해주는 변수

## 모듈이름: isreg(char\* a)

### 기능

SIC/XE machine의 레지스터 번호를 저장하고 있는 함수. a 가 레지스터라면 그에 맞는 값을 출력해준다. format2에 사용한다.

### 사용변수

없음

## 모듈이름: assemble()

### 기능

사용자에게 입력 받은 어셈블리 파일을 번역하여 object 파일과 list파일로 만들어준다.

### 사용변수

char fname[500] – asm 파일의 이름을 저장하는 변수

char objname[500] – object 파일의 이름을 저장하는 변수

char oneline[500] - asm파일을 한 줄씩 받는 변수

char a[50] – mnemonic 부분을 저장하는 변수

char b[50] – opeerand부분을 저장하는 변수

char c[50] – operand 부분에 comma가 있다면 그 뒤의 operand를 저장하는 변수

char d[50] – symbol 부분을 저장하는 변수

int i , j – 반복문을 위한 변수

int nlcheck – object 파일에서 길이가 0x1E가 넘는지 체크해주는 변수

int nlsl – object 파일에서 T뒤에 오는 address의 주소를 저장하는 변수

int FILE\* fp – asm 파일을 열기 위한 변수

int locctr – 현재 읽어 들인 줄의 주소를 저장하고, pass1에서는 전체 프로그램의 길이를 측정, pass2 에서는 PC-relative addressing 을 사용할 때 쓰인다.

int linecheck – asmfile 의 주소가 아닌 줄의 수를 저장하는 변수

int focheck, commacheck, valuecheck – 각각 현재 줄의 mnemonic의 format, comma의 개수,

operand가 숫자일 경우 이 수를 저장하는 변수.

int startadd – 어셈블리 프로그램의 시작주소를 저장하는 변수

int flag – object 파일에서의 새로운 줄이 시작되는 주소를 읽으면 flag가 1로 세팅된다.

int objcode – objectcode를 저장하는 변수

int base – base relative addressing을 사용하기 위해 base register의 값을 저장한다.

int asmlength – 어셈블리 프로그램의 길이를 저장한다.

FILE\* obj, lst – 각각 object 파일과 list 파일을 열어 작성하기 위한 변수

struct symnode\* new – operand의 symbol을 symbol table에 있는 symbol들에서 찾기 위한 변수

struct hashnode\* ptr - hashtable에서 구하고자 하는 mnemonic의 opcode와 format을 찾기 위한 변수

struct asmnode asmfile[200] – pass1에서 어셈블리 프로그램을 한 줄씩 읽어 처리할 때 이 때의

정보들을 저장하기 위한 변수

## 모듈이름: putsymnode(char\* sym, int add)

### 기능

symbol과 그 때의 address를 받아 symbol table 에 저장한다.

### 사용변수

struct symnode\* new – symbol table 에 접근하여 저장하기 위한 변수

## 모듈이름: symbol

### 기능

symbol table에 저장된 symbol과 address를 symbol name의 내림차순으로 보여준다.

### 사용변수

struct symnode\* ptr – symbol table의 내용을 하나씩 읽는 변수

char tmp1[50], tmp2[50] – tmp1에 symbol name의 가장 작은 경우를 저장하고 tmp2에 가장 큰 경우를 저장한다.

char a[50] – 가장 큰 경우의 다음으로 작은 경우를 저장하고, 가장 작아질 때까지 이를 반복하는 변수

# 전역 변수 정의

## objnametmp[50], lstnametmp[50]

object 파일과 list 파일의 이름을 저장한다. 이는 pass2에서 error가 발생하면 작성하던 두 파일을

삭제할 때 사용하기 때문이다.

## struct symnode, struct symnode\* symtab

symtab은 symbol table이고, 각각의 symbol은 하나의 symnode에 저장한다. symnode에는 symbol

name과 그 때의 address를 가지고 있다. Hash 형태로 저장한다.

## struct asmnode

asmnode는 어셈블리 파일의 한 줄에 대한 처리 정보들을 가지고 있다. 그 속에는 한 줄의 string, comma의 개수, mnemonic의 format, 이 줄의 location, symbol 이 선언된 여부, objectcode, object 파일에 새로운 줄에 써야하는 지점, format 4에서 direct addressing을 할 경우 modification 을 해야 한다는 것들을 알려준다.

# 코드

//20151561.h

void help();

int dir();

void quit();

void history();

int dump(int start, int end);

int edit(int address, int value);

int fill(int start, int end, int value);

void reset();

int opmn(int start, int end, int a);

void oplist();

int type();

int makenum(int start, int end);

int perform(int numptr, int whatcom);

void putnode();

void puthashnode(int opvalue, int mnvalue, int fovalue);

void putsymnode(char \*sym, int add);

int isreg(char \*a);

int assemble();

int symbol();

//20151561.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

#include "20151561.h"

unsigned char memory[0x100000] = {0, };

char com[500];

char tmp[500];

int dumpptr = 0;

char mntmp[10];

char objnametmp[50];

char lstnametmp[50];

struct node{

char command[500];

struct node \*next;

};

struct node \*head;

struct node \*tail;

struct hashnode{

char mn[10];

int op;

int fo;

struct hashnode \*next;

};

struct hashnode\* hashtable[20];

struct symnode{

char symbol[50];

int address;

struct symnode \*next;

};

struct symnode\* symtab[2];

struct asmnode {

char str[500];

int comma;

int format;

int loc;

int label;

unsigned int objectcode;

int newline;

int jsubflag;

};

int main(){

int i, j, l, mnvalue, validcheck = 0;

char optmp[4];

char fotmp[5];

int fovalue;

char \*pos = NULL;

unsigned int opvalue;

FILE \*fp;

struct symnode \*ptr;

fp = fopen("opcode.txt","r");

head = (struct node\*)malloc(sizeof(struct node));

head->next = NULL;

tail = head;

for(i=0; i<20; i++){

hashtable[i] = NULL;

}

symtab[0] = NULL;

symtab[1] = NULL;

while(1){

opvalue = 0;

mnvalue = 0;

fscanf(fp, "%s%s%s", optmp, mntmp, fotmp);

if(feof(fp)){

break;

}

opvalue = strtoul(optmp, &pos, 16);

for(i=0; i<strlen(mntmp); i++){

mnvalue = mnvalue + mntmp[i];

}

mnvalue = mnvalue%20;

if(strcmp(fotmp, "1") == 0){

fovalue = 1;

}

if(strcmp(fotmp, "2") == 0){

fovalue = 2;

}

if(strcmp(fotmp, "3/4") == 0){

fovalue = 3;

}

puthashnode(opvalue, mnvalue, fovalue);

}

fclose(fp);

while(1){

printf("sicsim> ");

fgets(com, 500, stdin);

l = strlen(com) - 1;

com[l] = '\0';

if(com[0] == '\0'){

printf("Enter the command\n");

continue;

}

strcpy(tmp, com);

for(i=0; i<strlen(com); i++){

if(com[i] == '\t'){

com[i] = ' ';

}

}

while(1){

if(com[0] == ' '){

for(i=0; i<strlen(com); i++){

com[i] = com[i+1];

}

}

else{

break;

}

}

for(i=strlen(com)-1; i>=0; i--){

if(com[i] == ' '){

com[i] = '\0';

}

else{

break;

}

}

i=0;

while(com[i] != '\0'){

if((com[i] == ' ') && ((com[i+1] == ' ') || (com[i+1] == ',') || (com[i-1] == ','))){

for(j=i;j<strlen(com); j++){

com[j] = com[j+1];

}

continue;

}

i++;

}

for(i=0; i<strlen(com); i++){

if((strncmp(com, "assemble", 8) == 0) || (strncmp(com, "type", 4) == 0)){

break;

}

if((com[i]>='A') && (com[i]<='Z')){

com[i] = com[i] + 32;

}

}

if(((strncmp(com, "help", 4) == 0) && (com[4] == '\0')) || ((com[0] == 'h') && (com[1] == '\0'))){

help();

validcheck = 1;

}

else if(((strncmp(com, "dir", 3) == 0) && (com[3] == '\0')) || ((com[0] == 'd') && (com[1] == '\0'))){

validcheck = dir();

}

else if(((strncmp(com, "quit", 4) == 0) && (com[4] == '\0')) || ((com[0] == 'q') && (com[1] == '\0'))){

quit();

}

else if(((strncmp(com, "history", 7) == 0) && (com[7] == '\0')) || ((com[0] == 'h') && (com[1] == 'i') && (com[2] == '\0'))){

putnode();

history();

}

else if(((strncmp(com, "dump", 4) == 0) && (com[4] == '\0')) || ((com[0] == 'd') && (com[1] == 'u') && (com[2] == '\0'))){

if(dumpptr+159 > 0xFFFFF){

validcheck = dump(dumpptr, 0xFFFFF);

}

else{

validcheck = dump(dumpptr, dumpptr+159);

}

}

else if(((strncmp(com, "dump", 4) == 0) && (com[4] == ' ')) || ((com[0] == 'd') && (com[1] == 'u') && (com[2] == ' '))){

if(strncmp(com, "dump", 4) == 0){

validcheck = perform(5, 1);

}

else{

validcheck = perform(3, 1);

}

}

else if(((strncmp(com, "edit", 4) == 0) && (com[4] == ' ')) || ((com[0] == 'e') && (com[1] == ' '))){

if(strncmp(com, "edit", 4) == 0){

validcheck = perform(5, 2);

}

else{

validcheck = perform(2, 2);

}

}

else if(((strncmp(com, "fill", 4) == 0) && (com[4] == ' ')) || ((com[0] == 'f') && (com[1] == ' '))){

if(strncmp(com, "fill", 4) == 0){

validcheck = perform(5, 3);

}

else{

validcheck = perform(2, 3);

}

}

else if(strncmp(com, "reset", 5) == 0){

reset();

validcheck = 1;

}

else if((strncmp(com, "opcode", 6) == 0) && (com[6] == ' ')){

validcheck = perform(7, 4);

}

else if((strncmp(com, "opcodelist", 10) == 0) && (com[10] == '\0')){

oplist();

validcheck = 1;

}

else if((strncmp(com, "type", 4) == 0) && (com[4] == ' ')){

validcheck = type();

}

else if((strncmp(com, "assemble", 8) == 0) && (com[8] == ' ')){

if(symtab[0] != NULL){

ptr = symtab[0];

while(ptr != NULL){

symtab[0] = ptr->next;

ptr->next = NULL;

free(ptr);

ptr = symtab[0];

}

symtab[0] = NULL;

}

validcheck = assemble();

if(validcheck == -1){

if(symtab[0] != NULL){

ptr = symtab[0];

while(ptr != NULL){

symtab[0] = ptr->next;

ptr->next = NULL;

free(ptr);

ptr = symtab[0];

}

symtab[0] = NULL;

}

}

if(validcheck == -2){

remove(objnametmp);

remove(lstnametmp);

}

}

else if((strncmp(com, "symbol", 6) == 0) && (com[6] == '\0')){

validcheck = symbol();

}

else{

printf("Invalid command\n");

}

if(validcheck == 1){

putnode();

}

validcheck = 0;

}

return 0;

}

void help(){

printf("h[elp]\nd[ir]\nq[uit]\nhi[story]\ndu[mp] [start, end]\ne[dit] address, value\nf[ill] start, end, value\nreset\nopcode mnemonic\nopcodelist\nassemble filename\ntype filename\nsymbol\n");

}

int dir(){

DIR \*dp;

struct dirent \*dirp;

char filename[500];

struct stat st;

dp = opendir(".");

if(dp == NULL){

printf("opendir error\n");

return 0;

}

while((dirp = readdir(dp)) != NULL){

if((strcmp(dirp->d\_name, ".") == 0) || (strcmp(dirp->d\_name,"..") == 0)){

continue;

}

sprintf(filename, "./%s", dirp->d\_name);

if(stat(filename, &st) == -1){

printf("stat error\n");

return 0;

}

printf("%s", dirp->d\_name);

if(S\_ISDIR(st.st\_mode)){

printf("/ ");

}

else if(st.st\_mode & S\_IXUSR){

printf("\* ");

}

else{

printf(" ");

}

}

closedir(dp);

printf("\n");

return 1;

}

void quit(){

struct node\* a = head;

struct node\* b = head->next;

struct hashnode\* c;

struct hashnode\* d;

struct symnode\* e = symtab[0];

struct symnode\* f;

int i;

while(b != NULL){

a->next = NULL;

free(a);

a = b;

b = a->next;

}

free(a);

for(i=0; i<20; i++){

if(hashtable[i] == NULL){

continue;

}

else{

c = hashtable[i];

d = hashtable[i]->next;

while(d != NULL){

c->next = NULL;

free(c);

c = d;

d = c->next;

}

free(c);

}

}

if(e != NULL){

while(e != NULL){

f = e->next;

e->next = NULL;

free(e);

e = f;

}

}

e = symtab[1];

if(e != NULL){

while(e != NULL){

f = e->next;

e->next = NULL;

free(e);

e = f;

}

}

exit(1);

}

void history(){

int i = 1;

struct node\* ptr = head->next;

while(ptr != NULL){

printf("%d %s\n", i, ptr->command);

ptr = ptr->next;

i++;

}

}

int dump(int start, int end){

int i, j;

if((start < 0) || (end > 0xFFFFF) || (start > 0xFFFFF) || (end < 0)){

return -1;

}

if(start > end){

return -2;

}

printf("%05X ", start - (start%0x10));

for(i=0; i<(start%0x10); i++){

printf(" ");

}

for(i=start; i<=end; i++){

printf("%02X ", memory[i]);

if((i+1)%0x10 == 0){

printf("; ");

for(j=i-0xF; j<=i; j++){

if(j < start){

printf(".");

}

else{

if((memory[j] >= 0x20) && (memory[j] <= 0x7E)){

printf("%c", memory[j]);

}

else{

printf(".");

}

}

}

printf("\n");

if(i != end){

printf("%05X ", i+1);

}

}

}

if(end%0x10 != 15){

for(i=end+1; i%0x10 != 0; i++){

printf(" ");

}

printf("; ");

for(i=end-end%0x10; i<=end; i++){

if(i < start){

printf(".");

}

else{

if((memory[i] >= 0x20) && (memory[i] <= 0x7E)){

printf("%c", memory[i]);

}

else{

printf(".");

}

}

}

for(i=end+1; i%0x10 != 0; i++){

printf(".");

}

printf("\n");

}

dumpptr = end + 1;

if(dumpptr > 0xFFFFF){

dumpptr = 0;

}

return 1;

}

int edit(int address, int value){

if((address < 0) || (address > 0xFFFFF) || (value > 0xFF)){

return -1;

}

memory[address] = value;

return 1;

}

int fill(int start, int end, int value){

int i;

if((start < 0) || (start > 0xFFFFF) || (end < 0) || (end > 0xFFFFF) || (value > 0xFF)){

return -1;

}

if(start > end){

return -2;

}

for(i=start; i<=end; i++){

memory[i] = value;

}

return 1;

}

void reset(char \*name){

int i;

for(i=0; i<=0xFFFFF; i++){

memory[i] = 0;

}

}

int opmn(int start, int end, int a){

struct hashnode\* ptr;

int i;

char tmp1[10];

char tmp2[10];

ptr = hashtable[a];

if((end-start+1) > 9){

return -1;

}

for(i=start; i<=end; i++){

tmp1[i-start] = com[i] - 'a' + 'A';

}

tmp1[end-start+1] = com[i];

while(ptr != NULL){

strcpy(tmp2, ptr->mn);

if(strcmp(tmp1, tmp2) == 0){

printf("opcode is %02X.\n", ptr->op);

return 1;

}

ptr = ptr->next;

}

return -1;

}

void oplist(){

int i, j;

struct hashnode \*ptr;

for(i=0; i<20; i++){

printf("%d :", i);

ptr = hashtable[i];

j = 0;

while(ptr != NULL){

if(j == 0){

printf(" [%s, %02X]", ptr->mn, ptr->op);

}

else{

printf(" -> [%s, %02X]", ptr->mn, ptr->op);

}

j++;

ptr = ptr->next;

}

printf("\n");

}

}

int type(){

char name[500];

int i;

FILE \*fp;

DIR \*dp;

char ch;

for(i=5; i<500; i++){

name[i-5] = com[i];

if(com[i] == '\0'){

break;

}

}

fp = fopen(name, "r");

if(fp == NULL){

printf("File does not exist\n");

return -1;

}

dp = opendir(name);

if(dp != NULL){

printf("This is directory\n");

closedir(dp);

return -1;

}

while((ch=fgetc(fp)) != EOF){

printf("%c", ch);

}

closedir(dp);

fclose(fp);

return 1;

}

int makenum(int start, int end){

int i, value = 0;

if(end-start == 1){

if((com[start] == '0') && (com[end] == 'x')){

return -1;

}

}

if((end-start > 1) && ((com[start] == '0') && (com[start+1] == 'x'))){

for(i=start+2; i<=end; i++){

if((com[i] >= 'a') && (com[i] <= 'f')){

value = (value\*0x10) + com[i] - 'a' + 10;

}

else if((com[i] >= '0') && (com[i] <= '9')){

value = (value\*0x10) + com[i] - '0';

}

else{

return -1;

}

}

}

else{

for(i=start; i<=end; i++){

if((com[i] >= 'a') && (com[i] <= 'f')){

value = (value\*0x10) + com[i] - 'a' + 10;

}

else if((com[i] >= '0') && (com[i] <= '9')){

value = (value\*0x10) + com[i] - '0';

}

else{

return -1;

}

}

}

return value;

}

int perform(int numptr, int whatcom){

int i, a, b, c, comcheck, commaptr1, commaptr2, commacount = 0;

int comlen = strlen(com);

if(com[numptr] == ','){

printf("Invalid command\n");

return 0;

}

if(com[comlen-1] == ','){

printf("Invalid command\n");

return 0;

}

for(i=numptr; i<comlen; i++){

if(com[i] == ','){

commacount++;

}

if(com[i] == ' '){

printf("Invalid command\n");

return 0;

}

}

if(commacount == 1){

for(i=numptr; i<comlen; i++){

if(com[i] == ','){

commaptr1 = i;

}

}

}

if(commacount == 2){

for(i=numptr; i<comlen; i++){

if(com[i] == ','){

commaptr1 = i;

break;

}

}

if(com[commaptr1 + 1] == ','){

printf("Invalid command\n");

return 0;

}

else{

for(i=commaptr1+1; i<comlen; i++){

if(com[i] == ','){

commaptr2 = i;

}

}

}

}

if(commacount > 2){

printf("Invalid command\n");

return 0;

}

if(commacount == 0){

if(whatcom == 4){

a = 0;

for(i=numptr; i<comlen; i++){

a = a + com[i] - 'a' + 'A';

}

a = a%20;

}

else{

a = makenum(numptr, comlen-1);

if(a == -1){

printf("Invalid input number\n");

return 0;

}

}

}

if(commacount == 1){

a = makenum(numptr, commaptr1-1);

if(a == -1){

printf("Invalid input number\n");

return 0;

}

b = makenum(commaptr1+1, comlen-1);

if(b == -1){

printf("Invalid input number\n");

return 0;

}

}

if(commacount == 2){

a = makenum(numptr, commaptr1-1);

if(a == -1){

printf("Invalid input number\n");

return 0;

}

b = makenum(commaptr1+1, commaptr2-1);

if(b == -1){

printf("Invalid input number\n");

return 0;

}

c = makenum(commaptr2+1, comlen-1);

if(c == -1){

printf("Invalid input number\n");

return 0;

}

}

if(whatcom == 1){

if(commacount == 0){

if(a+159 > 0xFFFFF){

comcheck = dump(a, 0xFFFFF);

if(comcheck == -1){

printf("Not within the boundary\n");

return 0;

}

}

else{

comcheck = dump(a, a+159);

}

}

else if(commacount == 1){

comcheck = dump(a, b);

if(comcheck == -1){

printf("Not within the boundary\n");

return 0;

}

if(comcheck == -2){

printf("Start is larger than end\n");

return 0;

}

}

else{

printf("Invalid command\n");

return 0;

}

}

else if(whatcom == 2){

if(commacount == 1){

comcheck = edit(a, b);

if(comcheck == -1){

printf("Invalid address or value\n");

return 0;

}

}

else{

printf("Invalid command\n");

return 0;

}

}

else if(whatcom == 3){

if(commacount == 2){

comcheck = fill(a, b, c);

if(comcheck == -1){

printf("Invalid address or value\n");

return 0;

}

if(comcheck == -2){

printf("Start is larger than end\n");

return 0;

}

}

else{

printf("Invalid command\n");

return 0;

}

}

else{

comcheck = opmn(numptr, comlen-1, a);

if(comcheck == -1){

printf("There is no corresponding mnemonic\n");

return 0;

}

}

return 1;

}

void putnode(){

struct node \*new;

new = (struct node\*)malloc(sizeof(struct node));

strcpy(new->command, tmp);

new->next = NULL;

tail->next = new;

tail = tail->next;

}

void puthashnode(int opvalue, int mnvalue, int fovalue){

struct hashnode \*new;

new = (struct hashnode\*)malloc(sizeof(struct hashnode));

strcpy(new->mn, mntmp);

new->op = opvalue;

new->fo = fovalue;

if(hashtable[mnvalue] == NULL){

new->next = NULL;

hashtable[mnvalue] = new;

}

else{

new->next = hashtable[mnvalue];

hashtable[mnvalue] = new;

}

}

int isreg(char \*a){

if(strcmp(a, "A") == 0){

return 0;

}

if(strcmp(a, "X") == 0){

return 1;

}

if(strcmp(a, "L") == 0){

return 2;

}

if(strcmp(a, "B") == 0){

return 3;

}

if(strcmp(a, "S") == 0){

return 4;

}

if(strcmp(a, "T") == 0){

return 5;

}

if(strcmp(a, "F") == 0){

return 6;

}

if(strcmp(a, "PC") == 0){

return 8;

}

if(strcmp(a, "SW") == 0){

return 9;

}

return -1;

}

int assemble(){

char fname[500];

char objname[500];

char oneline[500];

char a[50];

char b[50];

char c[50];

char d[50];

int i = 9;

int nlcheck = 0;

int nlsl = 0;

FILE \*fp;

int locctr = 0;

int linecheck = 0;

int focheck, commacheck, valuecheck, startadd, flag, objcode, j;

int base = -1;

int asmlength = 0;

FILE \*obj;

FILE \*lst;

struct symnode \*new;

struct hashnode \*ptr;

struct asmnode asmfile[200];

while(1){

fname[i-9] = com[i];

if(com[i] == '\0'){

break;

}

i++;

}

fp = fopen(fname, "r");

if(fp == NULL){

printf("File does not exist\n");

return -1;

}

for(i=0; i<strlen(fname); i++){

if(fname[i] == '.'){

fname[i] = '\0';

}

}

strcpy(objname, fname);

strcat(fname, ".lst\0");

strcat(objname, ".obj\0");

while(1){

linecheck++;

fgets(oneline, 500, fp);

for(i=0; i<strlen(oneline); i++){

if(oneline[i] == '\n'){

oneline[i] = '\0';

}

}

strcpy(asmfile[linecheck-1].str, oneline);

asmfile[linecheck-1].newline = 0;

a[0] = '\0';

b[0] = '\0';

c[0] = '\0';

sscanf(oneline, " %s %s %s", a, b, c);

if((a[0] == '.') || a == NULL){

asmfile[linecheck-1].comma = -1;

continue;

}

else{

if(strcmp(a, "START") == 0){

if(b == NULL){

locctr = 0;

asmfile[linecheck-1].comma = 7;

asmfile[linecheck-1].loc = 0;

asmfile[linecheck-1].label = 0;

startadd = locctr;

break;

}

for(i=0; i<strlen(b); i++){

if((b[i] >= '0') && (b[i] <= '9')){

locctr = locctr\*10 + b[i] - '0';

}

else{

fclose(fp);

printf("%d line pass1 error\n", 5\*linecheck);

return -1;

}

}

asmfile[linecheck-1].comma = 7;

asmfile[linecheck-1].loc = locctr;

asmfile[linecheck-1].label = 0;

startadd = locctr;

break;

}

else if(strcmp(b, "START") == 0){

if(c == NULL){

locctr = 0;

}

else{

for(i=0; i<strlen(c); i++){

if((c[i] >= '0') && (c[i] <= '9')){

locctr = locctr\*10 + c[i] - '0';

}

else{

fclose(fp);

printf("%d line pass1 error\n", 5\*linecheck);

return -1;

}

}

}

for(i=0; i<strlen(a); i++){

if((a[i]< 'A') || (a[i] > 'Z')){

fclose(fp);

printf("%d line pass1 error\n", 5\*linecheck);

return -1;

}

}

asmfile[linecheck-1].comma = 7;

asmfile[linecheck-1].loc = locctr;

asmfile[linecheck-1].label = 1;

putsymnode(a, locctr);

startadd = locctr;

break;

}

else{

fclose(fp);

printf("%d line error\n", 5\*linecheck);

return -1;

}

}

}//START

flag = 0;

while(1){

linecheck++;

commacheck = 0;

fgets(oneline, 500, fp);

if(feof(fp) != 0){

fclose(fp);

printf("%d line error\n", 5\*linecheck);

return -1;

}

for(i=0; i<strlen(oneline); i++){

if(oneline[i] == '\n'){

oneline[i] = '\0';

}

if(oneline[i] == ','){

commacheck++;

}

}

asmfile[linecheck-1].newline = 0;

asmfile[linecheck-1].comma = commacheck;

strcpy(asmfile[linecheck-1].str, oneline);

focheck = 0;

asmfile[linecheck-1].format = 0;

a[0] = '\0';

b[0] = '\0';

c[0] = '\0';

sscanf(oneline, " %s %s %s", a, b, c);

if(a[0] == '.'){

asmfile[linecheck-1].comma = -1;

continue;

}

if(commacheck > 1){

fclose(fp);

printf("%d line pass1 error\n", linecheck\*5);

return -1;

}

if(strcmp(a, "BASE") == 0){

asmfile[linecheck-1].comma = 6;

continue;

}

if(flag == 0){

nlsl = linecheck-1;

flag = 1;

}

if(strcmp(a, "END") == 0){

if(nlcheck != 0){

asmfile[nlsl].newline = nlcheck;

}

asmlength = locctr - startadd;

asmfile[linecheck-1].comma = 8;

break;

}

if(a[0] == '+'){//a == format4

for(i=0; i<strlen(a); i++){

a[i] = a[i+1];

}

asmfile[linecheck-1].format = 4;

}

for(i=0; i<strlen(a); i++){

focheck += a[i];

}

focheck = focheck % 20;

ptr = hashtable[focheck];

while(ptr != NULL){

if(strcmp(ptr->mn, a) == 0){

if((asmfile[linecheck-1].format == 4) && (ptr->fo != 3)){

fclose(fp);

printf("%d line pass1 error\n", linecheck\*5);

return -1;

}

else{

asmfile[linecheck-1].label = 0;

if(asmfile[linecheck-1].format != 4){

asmfile[linecheck-1].format = ptr->fo;

}

asmfile[linecheck-1].loc = locctr;

locctr += asmfile[linecheck-1].format;

break;

}

}

ptr = ptr->next;

}//a가 mnemonic인 것을 찾음. 아니라면 format이 0일 것.

if(asmfile[linecheck-1].format == 1){

asmfile[linecheck-1].objectcode = ptr->op;

}

if(asmfile[linecheck-1].format == 2){

asmfile[linecheck-1].objectcode = ptr->op\*0x100;

}

if(asmfile[linecheck-1].format == 3){

asmfile[linecheck-1].objectcode = ptr->op\*0x10000;

}

if(asmfile[linecheck-1].format == 4){

asmfile[linecheck-1].objectcode = ptr->op\*0x1000000;

}

//여기부터는 a가 mnemonic이 아닐 경우임.

if(asmfile[linecheck-1].format == 0){//a == symbol

focheck = 0;

for(i=0; i<strlen(a); i++){

if((a[i] < 'A') || (a[i] > 'Z')){

fclose(fp);

printf("%d line error\n", 5\*linecheck);

return -1;

}

}

if(strlen(a) > 6){

fclose(fp);

printf("%d line error\n", 5\*linecheck);

return -1;

}

if(b[0] == '+'){

for(i=0; i<strlen(b); i++){

b[i] = b[i+1];

}

asmfile[linecheck-1].format = 4;

}

for(i=0; i<strlen(b); i++){

focheck += b[i];

}

focheck = focheck % 20;

ptr = hashtable[focheck];

while(ptr != NULL){

if(strcmp(ptr->mn, b) == 0){

if((asmfile[linecheck-1].format == 4) && (ptr->fo != 3)){

fclose(fp);

printf("%d line pass1 error\n", linecheck\*5);

return -1;

}

else{

if(asmfile[linecheck-1].format != 4){

asmfile[linecheck-1].format = ptr->fo;

}

break;

}

}

ptr = ptr->next;

}

if(asmfile[linecheck-1].format == 1){

asmfile[linecheck-1].objectcode = ptr->op;

}

if(asmfile[linecheck-1].format == 2){

asmfile[linecheck-1].objectcode = ptr->op\*0x100;

}

if(asmfile[linecheck-1].format == 3){

asmfile[linecheck-1].objectcode = ptr->op\*0x10000;

}

if(asmfile[linecheck-1].format == 4){

asmfile[linecheck-1].objectcode = ptr->op\*0x1000000;

}

if(asmfile[linecheck-1].format == 0){

if(c == NULL){

fclose(fp);

printf("%d line pass1 error\n", linecheck\*5);

return -1;

}

valuecheck = 0;

if(strcmp(b, "BYTE") == 0){

if(strlen(c) > 3){

i = strlen(c);

if((c[0] == 'C') && (c[1] == 39) && (c[i-1] == 39)){

valuecheck = strlen(c) - 3;

}

else if((c[0] == 'X') && (c[1] == 39) && (c[i-1] == 39)){

valuecheck = strlen(c) - 3;

valuecheck = valuecheck%2 + valuecheck/2;

}

else{

fclose(fp);

printf("%d line pass1 error\n", linecheck\*5);

return -1;

}

asmfile[linecheck-1].format = valuecheck;

asmfile[linecheck-1].comma = 2;

}

else{

fclose(fp);

printf("%d line pass1 error\n", linecheck\*5);

return -1;

}

}

else if(strcmp(b, "WORD") == 0){

for(i=0; i<strlen(c); i++){

if((c[i] >= '0') && (c[i] <= '9')){

valuecheck = valuecheck\*10 + c[i] - '0';

}

else{

fclose(fp);

printf("%d line pass1 error\n", linecheck\*5);

return -1;

}

}

asmfile[linecheck-1].objectcode = valuecheck;

asmfile[linecheck-1].format = 3;

asmfile[linecheck-1].comma = 3;

}

else if(strcmp(b, "RESB") == 0){

for(i=0; i<strlen(c); i++){

if((c[i] >= '0') && (c[i] <= '9')){

valuecheck = valuecheck\*10 + c[i] - '0';

}

else{

fclose(fp);

printf("%d line pass1 error\n", linecheck\*5);

return -1;

}

}

asmfile[linecheck-1].format = valuecheck;

asmfile[linecheck-1].comma = 4;

}

else if(strcmp(b, "RESW") == 0){

for(i=0; i<strlen(c); i++){

if((c[i] >= '0') && (c[i] <= '9')){

valuecheck = valuecheck\*10 + c[i] - '0';

}

else{

fclose(fp);

printf("%d line pass1 error\n", linecheck\*5);

return -1;

}

}

asmfile[linecheck-1].format = valuecheck\*3;

asmfile[linecheck-1].comma = 5;

}

else{

fclose(fp);

printf("%d line pass1 error\n", linecheck\*5);

return -1;

}

}

new = symtab[0];

while(new != NULL){

if(strcmp(new->symbol, a) == 0){

fclose(fp);

printf("%d line pass1 error\n", linecheck\*5);

return -1;

}

new = new->next;

}

putsymnode(a, locctr);

asmfile[linecheck-1].label = 1;

asmfile[linecheck-1].loc = locctr;

locctr += asmfile[linecheck-1].format;

}

if((asmfile[linecheck-1].comma == 4) || (asmfile[linecheck-1].comma == 5)){

if(nlsl != 0){

asmfile[nlsl].newline = nlcheck;

nlcheck = 0;

nlsl = 0;

}

}

else{

if(nlcheck == 0){

nlsl = linecheck-1;

nlcheck = asmfile[linecheck-1].format;

}

else{

if(nlcheck + asmfile[linecheck-1].format > 0x1E){

asmfile[nlsl].newline = nlcheck;

nlsl = linecheck-1;

nlcheck = asmfile[linecheck-1].format;

}

else{

nlcheck += asmfile[linecheck-1].format;

}

}

}

}//pass1

fclose(fp);

lst = fopen(fname, "w+");

obj = fopen(objname, "w+");

strcpy(objnametmp, objname);

strcpy(lstnametmp, fname);

for(i=0; i<linecheck; i++){//pass2

asmfile[i].jsubflag = 0;

objcode = 0;

a[0] = '\0';

b[0] = '\0';

c[0] = '\0';

d[0] = '\0';

if(asmfile[i].comma == -1){//comment

fprintf(lst, "\t%s\n", asmfile[i].str);

continue;

}

else if(asmfile[i].comma == 2){//BYTE

sscanf(asmfile[i].str, "%s %s %s", a, b, c);

if(asmfile[i].newline != 0){

fprintf(obj, "\nT%06X%02X", asmfile[i].loc, asmfile[i].newline);

}

fprintf(lst, "%04X\t%s\t", asmfile[i].loc, asmfile[i].str);

if(c[0] == 'X'){

for(j=2; j<(strlen(c)-1); j++){

fprintf(obj, "%c", c[j]);

fprintf(lst, "%c", c[j]);

}

}

if(c[0] == 'C'){

for(j=2; j<(strlen(c)-1); j++){

fprintf(obj, "%X%X", c[j]/16, c[j]%16);

fprintf(lst, "%X%X", c[j]/16, c[j]%16);

}

}

fprintf(lst, "\n");

}

else if(asmfile[i].comma == 3){//WORD

objcode = asmfile[i].objectcode;

}

else if(asmfile[i].comma == 4){//RESB

fprintf(lst, "%04X\t%s\n", asmfile[i].loc, asmfile[i].str);

continue;

}

else if(asmfile[i].comma == 5){//RESW

fprintf(lst, "%04X\t%s\n", asmfile[i].loc, asmfile[i].str);

continue;

}

else if(asmfile[i].comma == 6){//BASE

sscanf(asmfile[i].str, "%s %s", a, b);

new = symtab[0];

while(new != NULL){

if(strcmp(b, new->symbol) == 0){

objcode = new->address;

break;

}

new = new->next;

}

if(objcode == 0){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

else{

base = objcode;

}

fprintf(lst, "\t%s\n", asmfile[i].str);

}

else if(asmfile[i].comma == 7){//start

locctr = asmfile[i].loc;

sscanf(asmfile[i].str, "%s", a);

fprintf(obj, "H%s", a);

for(j=0; j<(6-strlen(a)); j++){

fprintf(obj, " ");

}

fprintf(obj, "%06X%06X", asmfile[i].loc, asmlength);

}

else if(asmfile[i].comma == 8){//end

for(j=0; j<linecheck; j++){

if(asmfile[j].jsubflag == 1){

if(asmfile[j].format == 4){

fprintf(obj, "\nM%06X05", asmfile[j].loc+1-locctr);

}

else{

fprintf(obj, "\nM%06X03", asmfile[j].loc+1-locctr);

}

}

}

fprintf(lst, "\t%s\n", asmfile[i].str);

fprintf(obj, "\nE%06X\n", locctr);

}

else{//format 1~4

if(asmfile[i].comma == 1){

if(asmfile[i].label == 1){

sscanf(asmfile[i].str, "%s %s %[^,] %\*[,]%s", d, a, b, c);

}

else{

sscanf(asmfile[i].str, "%s %[^,] %\*[,]%s", a, b, c);

}

}

else{

if(asmfile[i].label == 1){

sscanf(asmfile[i].str, "%s %s %s", d, a, b);

}

else{

sscanf(asmfile[i].str, "%s %s", a, b);

}

}

if(asmfile[i].format == 1){

objcode = asmfile[i].objectcode;

}

else if(asmfile[i].format == 2){

if(asmfile[i].comma == 1){

if(isreg(b) == -1){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

else{

if(isreg(c) == -1){

if(atoi(c) == 0){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

if((strcmp(a, "SHIFTL") != 0) && (strcmp(a, "SHIFTR") != 0)){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

if(atoi(c) > 16){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

asmfile[i].objectcode += (atoi(c)-1);

}

else{

asmfile[i].objectcode += isreg(c);

}

}

asmfile[i].objectcode += (isreg(b)\*16);

}

else{

if(isreg(b) == -1){

if(strcmp(a, "SVC") == 0){

if(atoi(b) == 0){

if(strcmp(b, "0") != 0){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

}

else{

asmfile[i].objectcode += (atoi(b)\*16);

}

}

else{

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

}

else{

asmfile[i].objectcode += (isreg(b)\*16);

}

}

objcode = asmfile[i].objectcode;

}

else{//format 3, 4

if((strcmp(a, "RSUB") != 0) && (strcmp(a, "+RSUB") != 0)){

if(atoi(b) != 0){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

if((b[0] == '#') || (b[0] == '@')){

if(b[0] == '#'){

if(asmfile[i].format == 3){

asmfile[i].objectcode += 0x10000;

}

else{

asmfile[i].objectcode += 0x1000000;

}

}

else{

if(asmfile[i].format == 3){

asmfile[i].objectcode += 0x20000;

}

else{

asmfile[i].objectcode += 0x2000000;

}

}

for(j=0; j<strlen(b); j++){

b[j] = b[j+1];

}

if(strcmp(b, "0") != 0){//숫자인지 심볼인지

j=0;

new = symtab[0];

while(new != NULL){

if(strcmp(new->symbol, b) == 0){

j = new->address;

break;

}

new = new->next;

}

if(j == 0){//숫자

if(atoi(b) == 0){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

if(asmfile[i].format == 4){

if(atoi(b) > 0xFFFFF){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

}

if(asmfile[i].format == 3){

if(atoi(b) > 0xFFF){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

}

asmfile[i].objectcode += atoi(b);

}

else{//심볼

if(asmfile[i].format == 4){

asmfile[i].objectcode += j;

asmfile[i].jsubflag = 1;

}

else{

if(((j-asmfile[i].loc-3) < -2048) || ((j-asmfile[i].loc-3) > 2047)){

if(base == -1){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

else{

if(((j-base) > 4095) || ((j-base) < 0)){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

else{//base

asmfile[i].objectcode += 0x4000;

asmfile[i].objectcode += (j-base);

}

}

}

else{//pc relative

asmfile[i].objectcode += 0x2000;

asmfile[i].objectcode += (j-asmfile[i].loc-3);

if((j-asmfile[i].loc-3) < 0){

asmfile[i].objectcode += 0x1000;

}

}

}

}

}

}

else{//심볼인지

j=0;

new = symtab[0];

while(new != NULL){

if(strcmp(new->symbol, b) == 0){

j = new->address;

break;

}

new = new->next;

}

if(j==0){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

if(asmfile[i].format == 4){

asmfile[i].jsubflag = 1;

asmfile[i].objectcode += j;

asmfile[i].objectcode += 0x3000000;

}

else{

if(((j-asmfile[i].loc-3) < -2048) || ((j-asmfile[i].loc-3) > 2047)){

if(base == -1){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

else{

if(((j-base) >4095) || ((j-base) < 0)){

fclose(lst);

fclose(obj);

printf("%d line pass2 error\n", (i+1)\*5);

return -2;

}

else{

asmfile[i].objectcode += 0x4000;

asmfile[i].objectcode += (j-base);

}

}

}

else{

asmfile[i].objectcode += 0x2000;

asmfile[i].objectcode += (j-asmfile[i].loc-3);

if((j-asmfile[i].loc-3) < 0){

asmfile[i].objectcode += 0x1000;

}

}

asmfile[i].objectcode += 0x30000;

}

}

if(asmfile[i].format == 4){

asmfile[i].objectcode += 0x100000;

}

if(strcmp(c, "X") == 0){

if(asmfile[i].format == 4){

asmfile[i].objectcode += 0x800000;

}

else{

asmfile[i].objectcode += 0x8000;

}

}

}

else{//RSUB

if(a[0] == '+'){

asmfile[i].objectcode += 0x3000000;

}

else{

asmfile[i].objectcode += 0x30000;

}

}

objcode = asmfile[i].objectcode;

}

}

if((asmfile[i].comma != 8) && (asmfile[i].comma != 2) && (asmfile[i].comma != 6)){

if(asmfile[i].label == 0){

fprintf(lst, "%04X\t%s\t", asmfile[i].loc, asmfile[i].str);

}

else{

fprintf(lst, "%04X\t%s\t", asmfile[i].loc, asmfile[i].str);

}

if(asmfile[i].comma == 7){

fprintf(lst, "\n");

}

else{

if(asmfile[i].format == 1){

fprintf(lst, "%02X\n", objcode);

}

else if(asmfile[i].format == 2){

fprintf(lst, "%04X\n", objcode);

}

else if(asmfile[i].format == 3){

fprintf(lst, "%06X\n", objcode);

}

else{

fprintf(lst, "%08X\n", objcode);

}

}

}

if((asmfile[i].comma != 3) && (asmfile[i].comma != 1) && (asmfile[i].comma != 0)){

continue;

}

if(asmfile[i].newline != 0){

if(asmfile[i].format == 4){

fprintf(obj, "\nT%06X%02X%08X", asmfile[i].loc, asmfile[i].newline, objcode);

}

else if(asmfile[i].format == 3){

fprintf(obj, "\nT%06X%02X%06X", asmfile[i].loc, asmfile[i].newline, objcode);

}

else if(asmfile[i].format == 2){

fprintf(obj, "\nT%06X%02X%04X", asmfile[i].loc, asmfile[i].newline, objcode);

}

else{

fprintf(obj, "\nT%06X%02X%02X", asmfile[i].loc, asmfile[i].newline, objcode);

}

}

else{

if(asmfile[i].format == 4){

fprintf(obj, "%08X", objcode);

}

else if(asmfile[i].format == 3){

fprintf(obj, "%06X", objcode);

}

else if(asmfile[i].format == 2){

fprintf(obj, "%04X", objcode);

}

else{

fprintf(obj, "%02X", objcode);

}

}

}

fclose(lst);

fclose(obj);

return 1;

}

void putsymnode(char \*sym, int add){

struct symnode \*new;

new = (struct symnode\*)malloc(sizeof(struct symnode));

strcpy(new->symbol, sym);

new->address = add;

if (strlen(sym) <= 6) {

if (symtab[0] == NULL) {

new->next = NULL;

symtab[0] = new;

}

else {

new->next = symtab[0];

symtab[0] = new;

}

}

else {

if (symtab[1] == NULL) {

new->next = NULL;

symtab[1] = new;

}

else {

new->next = symtab[1];

symtab[1] = new;

}

}

}

int symbol(){

struct symnode\* ptr = symtab[0];

char tmp1[50];

char tmp2[50];

char a[50];

if(symtab[0] == NULL){

printf("Nothing in the table\n");

return -1;

}

strcpy(tmp1, symtab[0]->symbol);

strcpy(tmp2, symtab[0]->symbol);

while(ptr != NULL){

if(strcmp(ptr->symbol, tmp1) > 0){

strcpy(tmp1, ptr->symbol);

}

ptr = ptr->next;

}

ptr = symtab[0];

while(ptr != NULL){

if(strcmp(ptr->symbol, tmp2) < 0){

strcpy(tmp2, ptr->symbol);

}

ptr = ptr->next;

}

if(strcmp(tmp1, tmp2) == 0){

printf("\t%s\t%04X\n", symtab[0]->symbol, symtab[0]->address);

}

else{

ptr = symtab[0];

while(ptr != NULL){

if(strcmp(ptr->symbol, tmp1) == 0){

break;

}

ptr = ptr->next;

}

printf("\t%s\t%04X\n", ptr->symbol, ptr->address);

while(1){

strcpy(a, tmp2);

ptr = symtab[0];

while(ptr != NULL){

if((strcmp(tmp1, ptr->symbol) > 0) && (strcmp(a, ptr->symbol) < 0)){

strcpy(a, ptr->symbol);

}

ptr = ptr->next;

}

ptr = symtab[0];

while(ptr != NULL){

if(strcmp(a, ptr->symbol) == 0){

break;

}

ptr = ptr->next;

}

printf("\t%s\t%04X\n", ptr->symbol, ptr->address);

if(strcmp(a, tmp2) == 0){

break;

}

else{

strcpy(tmp1, a);

}

}

}

return 1;

}