과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 3>>

**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 : int progaddr() 3

3.1.1 기능 3

3.1.2 사용 변수 3

3.2 모듈 이름: int loader() 4

3.2.1 기능 4

3.2.2 사용 변수 4

3.3 모듈이름: putbpnode(int val) 4

3.3.1 기능 4

3.3.2 사용변수 4

3.4 모듈이름: bp(int command) 4

3.4.1 기능 4

3.4.2 사용변수 4

3.5 모듈이름: run() 5

3.5.1 기능 5

3.5.2 사용변수 5

4. 전역 변수 정의 5

4.1 int addr 5

4.2 struct extnode 5

4.3 struct bpnode, struct bpnode \*bphead 5

4.4 int finst, proglen 5

4.5 int ra, rx, rl, rpc, rb, rs, rt, rcc, rf 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

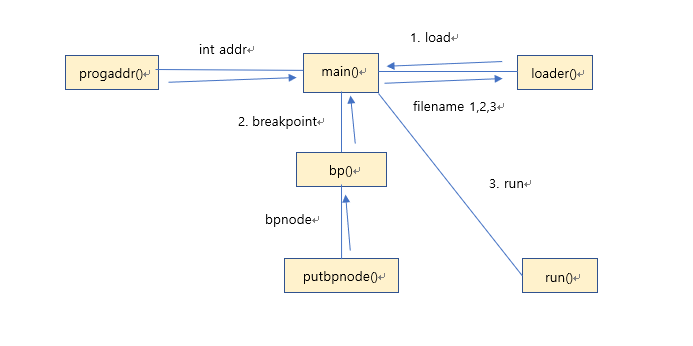


그림 1> 프로그램 흐름도

# 모듈 정의

## 모듈 이름 : int progaddr()

### 기능

사용자가 memory 범위 안(0x00~0xFFFFF)의 원하는 주소를 직접 지정한다.

### 사용 변수

int i – 반복문을 위한 변수

int a – 사용자가 입력한 address를 16진수 숫자로 저장하는 변수

## 모듈 이름: int loader()

### 기능

사용자가 입력한 1~3개의 파일을 지정한 메모리 주소를 시작주소로 하여link, load 한다.

### 사용 변수

char a[100], b[100], c[100], d[100], e[100] – 입력한 파일 이름을 b, c, d에 저장하고, e는 세 개 이상의 파일을 입력하였는지 체크하기 위해 사용한다.

FILE \*fp – 파일을 열고 닫기 위한 변수

int i, j – 반복문을 위한 변수

int filenumb, cslth, csaddr – 각각 입력된 파일의 개수, 현재 처리하는 control section 의 길이, 시작주소를 가지고 있다.

int addrtmp, numtmp, estabnumb – 각각 주소와 관련된 값을 저장, 번호와 관련된 값을 저장, external symbol table 에 현재 들어있는 symbol의 개수를 저장하는 변수이다.

int addr1, addr2, addr3 – control section각각의 시작주소를 저장하는 변수이다.

unsigned int valtmp – 메모리에 들어있는 값을 load, store 할 때 사용하는 변수이다. unsigned 인 이유는 memory 가 unsigned char 형태이기 때문이다. (carry에 대한 부분)

struct extnode estab[300] - external symbol table이다.

int ertab1[100], ertab2[100], ertab3[100] – control section각각의 external reference table이다.

int ch, flag, total – 각각 파일의 H, M, E, T 등등을 구별할 때 사용하는 변수, 0또는 1로 설정되어 loop문을 빠져나올 때 쓰는 변수, load된 프로그램의 총 길이를 저장하는 변수이다.

char linetmp[300] – 파일의 한 줄을 읽을 때 사용하는 변수이다.

char memtmp[3] – 메모리에 저장해야 할 1byte의 숫자를 char형태로 읽을 때 사용한다.

## 모듈이름: putbpnode(int val)

### 기능

입력된 breakpoint를 추가한다.

### 사용변수

struct bpnode \*new, \*ptr – 각각 breakpoint node 를 생성, 집어넣을 위치를 가리키는 변수이다.

## 모듈이름: bp(int command)

### 기능

command 가 1이면 입력된 breakpoint를 putbpnode함수를 통해 추가하고,

command가 2이면 저장된 breakpoint들을 모두 보여주고,

command 가 3이면 저장된 breakpoint들을 모두 삭제한다.

### 사용변수

struct bpnode \*ptr, \*tmp – 각각 bpnode를 가리킬 때, 저장된 node를 수정할 때 사용하는 변수이다.

int i, a – 각각 반복문, 추가할 주소를 char에서 hexadecimal로 저장할 때 사용하는 변수이다.

## 모듈이름: run()

### 기능

link, load된 프로그램을 지정한 시작주소에서부터 실행한다. 임의로 만든 레지스터에 그때의 값들이 저장된다.

### 사용변수

int opcode, on, oi, ox, ob, op, oe, disp, r1, r2, ta, tv, i, flag – 각각 현재 위치의 object code를 읽어 opcode, n, i, x, b, p, e, disp, rl, r2의 값으로 저장한다. ta, tv는 각각 target address, target value이다. i는 반복문을 위해 사용되고, flag는 1로 설정되면 반복문을 빠져나올 때 사용된다.

# 전역 변수 정의

## int addr

사용자가 입력한 프로그램의 주소가 저장된다. (0x00~0xFFFFF) 아무것도 입력하지 않으면 시작주소는 0x00이다.

## struct extnode

char name[7]은 external symbol의 이름, int address는 external symbol의 실제 주소, int length는 external symbol이 program name일 때 현재 control section의 길이를 저장한다.

## struct bpnode, struct bpnode \*bphead

int checkpoint는 사용자가 설정한 breakpoint의 값, struct bpnode \*next는 linked list의 다음 node의 주소이다. bphead는 linked list의 가장 앞 부분을 가리키는 변수이다.

## int finst, proglen

finst는 프로그램이 상대주소0부터 시작하지 않았을 때 시작주소에서 얼만큼 떨어진 곳에서 시작하는지를 저장하고 있는 변수이다.

proglen은 load 된 프로그램의 총 길이를 저장하는 변수이다.

## int ra, rx, rl, rpc, rb, rs, rt, rcc, rf

각각 레지스터 A, X, L, PC, B, S, T, SW, F의 값을 저장하는 변수이다.

# 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();

int progaddr();

int loader();

void putbpnode(int val);

int bp(int command);

int run();

//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, };

unsigned char memorytmp[0x100000];

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 addr = 0;

struct extnode{

char name[7];

int address;

int length;

};

struct bpnode {

int checkpoint;

struct bpnode \*next;

};

struct bpnode \*bphead;

int finst, proglen;

int ra, rx, rl, rpc, rb, rs, rt, rcc, rf;

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;

proglen = 0;

bphead = NULL;

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) || (strncmp(com, "loader", 6) == 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 if((strncmp(com, "progaddr", 8) == 0) && (com[8] == ' ')){

validcheck = progaddr();

}

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

validcheck = loader();

}

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

validcheck = bp(3);

}

else if ((strncmp(com, "bp", 2) == 0) && (com[2] == ' ')) {

validcheck = bp(1);

}

else if ((strncmp(com, "bp", 2) == 0) && (com[2] == '\0')) {

validcheck = bp(2);

}

else if ((strncmp(com, "run", 3) == 0) && (com[3] == '\0')) {

validcheck = run();

}

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\nprogaddr [address]\nloader [filename1] [filename2] [filename3]\nrun\nbp [address|clear]\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;

struct bpnode\* bpptr;

struct bpnode\* bptmp;

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;

}

}

if (bphead != NULL) {

bpptr = bphead;

bptmp = bpptr->next;

while (1) {

bpptr->next = NULL;

free(bpptr);

if (bptmp == NULL) {

break;

}

bpptr = bptmp;

bptmp = bpptr->next;

}

}

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;

}

int progaddr(){

int i, a = 0;

if((com[9] == '0')){

if(com[10] == '\0'){//주소가 0인 경우

addr = 0;

}

else if(com[10] == 'x'){// 주소가 0x로 입력된 경우

if(com[11] == '\0'){

printf("invalid address\n");

return -1;

}

else{

for(i=11; i<strlen(com); i++){//주소 계산

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

a = a\*16 + com[i] - 'a' + 10;

}

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

a = a\*16 + com[i] - '0';

}

else{

printf("invalid address\n");

return -1;

}

}

}

}

else{//0뒤에 x가 아닌 주소가 들어오는 경우

for(i=9; i<strlen(com); i++){

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

a = a\*16 + com[i] - 'a' + 10;

}

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

a = a\*16 + com[i] - '0';

}

else{

printf("invalid address\n");

return -1;

}

}

}

}

else{//주소가 0x없이 주어졌을 경우

for(i=9; i<strlen(com); i++){

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

a = a\*16 + com[i] - 'a' + 10;

}

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

a = a\*16 + com[i] - '0';

}

else{

printf("invalid address\n");

return -1;

}

}

}

if(a > 0xfffff){

printf("invalid address\n");

return -1;

}

addr = a;

printf("\nProgram starting address set to 0x%x.\n\n", addr);

return 1;

}

int loader(){

char a[100], b[100], c[100], d[100], e[100];

FILE \*fp;

int filenumb, i, j, cslth, csaddr, addrtmp, numtmp, addr1, addr2, addr3, estabnumb = 0;

unsigned int valtmp;

struct extnode estab[300];

int ertab1[100] = {-1, };

int ertab2[100] = {-1, };

int ertab3[100] = {-1, };

int ch, flag, total;

char linetmp[300];

char memtmp[3];

a[0] = '\0';

b[0] = '\0';

c[0] = '\0';

d[0] = '\0';

e[0] = '\0';

sscanf(com, "%s %s %s %s %s", a, b, c, d, e);

if(e[0] != '\0'){

printf("file maximum is 3\n");

proglen = 0;

return -1;

}

if(b[0] != '\0'){

filenumb = 1;

}

if(c[0] != '\0'){

filenumb = 2;

}

if(d[0] != '\0'){

filenumb = 3;

}// 입력받은 파일의 갯수를 확인.

for(i=0; i<filenumb; i++){//pass1

if(i == 0){

fp = fopen(b, "r");

csaddr = addr;

addr1 = csaddr;

}

if(i == 1){

fp = fopen(c, "r");

csaddr = csaddr + cslth;

addr2 = csaddr;

}

if(i == 2){

fp = fopen(d, "r");

csaddr = csaddr + cslth;

addr3 = csaddr;

}//첫번째, 두번째, 세번째 파일의 csaddr을 지정.

if(fp == NULL){

printf("file name error\n");

proglen = 0;

return -1;

}

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

if(ch == 'H'){

fgets(linetmp, 7, fp);

for(j=0; j<6; j++){

if(linetmp[j] == ' '){

linetmp[j] = '\0';

}

}

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

if(strcmp(estab[j].name, linetmp) == 0){

printf("duplicate symbol\n");

fclose(fp);

proglen = 0;

return -1;

}

}

strcpy(estab[estabnumb].name, linetmp);//프로그램의 이름

fgets(linetmp, 7, fp);

addrtmp = 0;

for(j=0; j<6; j++){

if((linetmp[j] >= '0') && (linetmp[j] <= '9')){

addrtmp = addrtmp\*16 + linetmp[j] - '0';

}

else if((linetmp[j] >= 'a') && (linetmp[j] <= 'f')){

addrtmp = addrtmp\*16 + linetmp[j] - 'a' + 10;

}

else if((linetmp[j] >= 'A') && (linetmp[j] <= 'F')){

addrtmp = addrtmp\*16 + linetmp[j] - 'A' + 10;

}

else{

printf("obj file error\n");

fclose(fp);

proglen = 0;

return -1;

}

}

estab[estabnumb].address = csaddr + addrtmp;//프로그램의 시작 주소

fgets(linetmp, 7, fp);

addrtmp = 0;

for(j=0; j<6; j++){

if((linetmp[j] >= '0') && (linetmp[j] <= '9')){

addrtmp = addrtmp\*16 + linetmp[j] - '0';

}

else if((linetmp[j] >= 'a') && (linetmp[j] <= 'f')){

addrtmp = addrtmp\*16 + linetmp[j] - 'a' + 10;

}

else if((linetmp[j] >= 'A') && (linetmp[j] <= 'F')){

addrtmp = addrtmp\*16 + linetmp[j] - 'A' + 10;

}

else{

printf("obj file error\n");

fclose(fp);

proglen = 0;

return -1;

}

}

estab[estabnumb].length = addrtmp;//프로그램의 길이

cslth = addrtmp;

if ((csaddr + cslth) > 0xfffff) {

printf("range over\n");

fclose(fp);

proglen = 0;

return -1;

}

estabnumb++;

}

if(ch == 'D'){

flag = 0;

while (1) {

fgets(linetmp, 7, fp);

for (j = 0; j < 6; j++) {

if (linetmp[j] == ' ') {

linetmp[j] = '\0';

}

if (linetmp[j] == '\n') {

flag = 1;

break;

}

}

if (flag == 1) {//줄의 끝에 도달하면 flag가 1이 됨.

break;

}

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

if (strcmp(estab[j].name, linetmp) == 0) {

printf("duplicate symbol\n");

fclose(fp);

proglen = 0;

return -1;

}

}

strcpy(estab[estabnumb].name, linetmp);//symbol의 이름

fgets(linetmp, 7, fp);

addrtmp = 0;

for (j = 0; j < 6; j++) {

if ((linetmp[j] >= '0') && (linetmp[j] <= '9')) {

addrtmp = addrtmp \* 16 + linetmp[j] - '0';

}

else if ((linetmp[j] >= 'a') && (linetmp[j] <= 'f')) {

addrtmp = addrtmp \* 16 + linetmp[j] - 'a' + 10;

}

else if ((linetmp[j] >= 'A') && (linetmp[j] <= 'F')) {

addrtmp = addrtmp \* 16 + linetmp[j] - 'A' + 10;

}

else {

printf("obj file error\n");

fclose(fp);

proglen = 0;

return -1;

}

}

estab[estabnumb].address = csaddr + addrtmp;//symbol의 주소

estab[estabnumb].length = 0;

estabnumb++;

}

}

if(ch == 'R'){

fgets(linetmp, 300, fp);

}

if(ch == 'T'){

fgets(linetmp, 7, fp);

addrtmp = 0;

for (j = 0; j<6; j++) {

if ((linetmp[j] >= '0') && (linetmp[j] <= '9')) {

addrtmp = addrtmp \* 16 + linetmp[j] - '0';

}

else if ((linetmp[j] >= 'a') && (linetmp[j] <= 'f')) {

addrtmp = addrtmp \* 16 + linetmp[j] - 'a' + 10;

}

else if ((linetmp[j] >= 'A') && (linetmp[j] <= 'F')) {

addrtmp = addrtmp \* 16 + linetmp[j] - 'A' + 10;

}

else {

printf("obj file error\n");

fclose(fp);

proglen = 0;

return -1;

}

}

fgets(memtmp, 3, fp);

j = 0;

while (1) {

fgets(memtmp, 3, fp);

if((memtmp[0] == '\n') || (memtmp[1] == '\n')){

break;

}

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

memory[addrtmp+csaddr + j] = 16 \* (memtmp[0] - '0');

}

else if ((memtmp[0] >= 'a') && (memtmp[0] <= 'f')) {

memory[addrtmp + csaddr + j] = 16 \* (memtmp[0] - 'a' + 10);

}

else if ((memtmp[0] >= 'A') && (memtmp[0] <= 'F')) {

memory[addrtmp + csaddr + j] = 16 \* (memtmp[0] - 'A' + 10);

}

else {

printf("obj file error\n");

fclose(fp);

proglen = 0;

return -1;

}

if ((memtmp[1] >= '0') && (memtmp[1] <= '9')) {

memory[addrtmp + csaddr + j] += (memtmp[1] - '0');

}

else if ((memtmp[1] >= 'a') && (memtmp[1] <= 'f')) {

memory[addrtmp + csaddr + j] += (memtmp[1] - 'a' + 10);

}

else if ((memtmp[1] >= 'A') && (memtmp[1] <= 'F')) {

memory[addrtmp + csaddr + j] += (memtmp[1] - 'A' + 10);

}

else {

printf("obj file error\n");

fclose(fp);

proglen = 0;

return -1;

}

j++;

}

}

if(ch == 'M'){

fgets(linetmp, 300, fp);

}

if(ch == 'E'){

fgets(linetmp, 300, fp);

}

if((ch == '.') || (ch == '\n') || (ch == ' ') || (ch == '\t')){

continue;

}

}

fclose(fp);

}

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

if (i == 0) {

fp = fopen(b, "r");

csaddr = addr1;

}

if (i == 1) {

fp = fopen(c, "r");

csaddr = addr2;

}

if (i == 2) {

fp = fopen(d, "r");

csaddr = addr3;

}

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

if (ch == 'H') {

fgets(linetmp, 300, fp);

}

if (ch == 'D') {

fgets(linetmp, 300, fp);

}

if (ch == 'R') {

flag = 0;

while (1) {

fgets(memtmp, 3, fp);

if ((memtmp[0] == '\n') || (memtmp[1] == '\n')) {

break;

}//line의 끝에 도달

if ((memtmp[0] >= '0') && (memtmp[0] <= '9') && (memtmp[1] >= '0') && (memtmp[1] <= '9')) {//앞에 number 가 붙은 경우

fgets(linetmp, 7, fp);

for (j = 0; j < 6; j++) {

if (linetmp[j] == ' ') {

linetmp[j] = '\0';

}

if (linetmp[j] == '\n') {

flag = 1;

linetmp[j] = '\0';

break;

}

}

if (i == 0) {

ertab1[1] = csaddr;

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

if (strcmp(estab[j].name, linetmp) == 0) {

ertab1[(memtmp[0] - '0') \* 10 + (memtmp[1] - '0')] = estab[j].address;

break;

}

if (j == (estabnumb - 1)) {

printf("not in estab\n");

fclose(fp);

proglen = 0;

return -1;

}

}

}

if (i == 1) {

ertab2[1] = csaddr;

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

if (strcmp(estab[j].name, linetmp) == 0) {

ertab2[(memtmp[0] - '0') \* 10 + (memtmp[1] - '0')] = estab[j].address;

break;

}

if (j == (estabnumb - 1)) {

printf("not in estab\n");

fclose(fp);

proglen = 0;

return -1;

}

}

}

if (i == 2) {

ertab3[1] = csaddr;

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

if (strcmp(estab[j].name, linetmp) == 0) {

ertab3[(memtmp[0] - '0') \* 10 + (memtmp[1] - '0')] = estab[j].address;

break;

}

if (j == (estabnumb - 1)) {

printf("not in estab\n");

fclose(fp);

proglen = 0;

return -1;

}

}

}

if (flag == 1) {//줄의 끝에 도달하면 flag가 1이 됨.

break;

}

}

}

}

if (ch == 'T') {

fgets(linetmp, 300, fp);

}

if (ch == 'M') {

fgets(linetmp, 7, fp);

addrtmp = 0;

for (j = 0; j < 6; j++) {

if ((linetmp[j] >= '0') && (linetmp[j] <= '9')) {

addrtmp = addrtmp \* 16 + linetmp[j] - '0';

}

else if ((linetmp[j] >= 'a') && (linetmp[j] <= 'f')) {

addrtmp = addrtmp \* 16 + linetmp[j] - 'a' + 10;

}

else if ((linetmp[j] >= 'A') && (linetmp[j] <= 'F')) {

addrtmp = addrtmp \* 16 + linetmp[j] - 'A' + 10;

}

else {

printf("obj file error\n");

fclose(fp);

proglen = 0;

return -1;

}

}

fgets(memtmp, 3, fp);

numtmp = (memtmp[0] - '0') \* 10 + (memtmp[1] - '0');

valtmp = 0;

if (numtmp % 2 == 0) {//짝수 half byte

for (j = 0; j < (numtmp / 2); j++) {//메모리에 저장된 값을 가져옴

valtmp = valtmp \* 256 + memory[csaddr + addrtmp + j];

}

}

else {//홀수 half byte

valtmp = (memory[csaddr + addrtmp] % 16);

for (j = 0; j < (numtmp / 2); j++) {

valtmp = valtmp \* 256 + memory[csaddr + addrtmp + j + 1];

}

}

fgets(linetmp, 300, fp);

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

if (linetmp[j] == '\n') {

linetmp[j] = '\0';

}

}

if (linetmp[0] == '+') {//flag를 +면 0, -면 1로 설정

flag = 0;

}

else if(linetmp[0] == '-'){

flag = 1;

}

else {

if (filenumb != 1){

printf("obj file error\n");

fclose(fp);

proglen = 0;

return -1;

}

}

if(filenumb != 1){

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

linetmp[j] = linetmp[j + 1];

}

}

if ((linetmp[0] >= '0') && (linetmp[0] <= '9') && (linetmp[1] >= '0') && (linetmp[1] <= '9')) {//reference number 가 주어질 때

if (i == 0) {

if (ertab1[(linetmp[0] - '0') \* 10 + (linetmp[1] - '0')] == -1) {

printf("no reference number\n");

fclose(fp);

proglen = 0;

return -1;

}

if (flag == 0) {

valtmp += ertab1[(linetmp[0] - '0') \* 10 + (linetmp[1] - '0')];

}

else {

valtmp -= ertab1[(linetmp[0] - '0') \* 10 + (linetmp[1] - '0')];

}

}

if (i == 1) {

if (ertab2[(linetmp[0] - '0') \* 10 + (linetmp[1] - '0')] == -1) {

printf("no reference number\n");

fclose(fp);

proglen = 0;

return -1;

}

if (flag == 0) {

valtmp += ertab2[(linetmp[0] - '0') \* 10 + (linetmp[1] - '0')];

}

else {

valtmp -= ertab2[(linetmp[0] - '0') \* 10 + (linetmp[1] - '0')];

}

}

if (i == 2) {

if (ertab3[(linetmp[0] - '0') \* 10 + (linetmp[1] - '0')] == -1) {

printf("no reference number\n");

fclose(fp);

proglen = 0;

return -1;

}

if (flag == 0) {

valtmp += ertab3[(linetmp[0] - '0') \* 10 + (linetmp[1] - '0')];

}

else {

valtmp -= ertab3[(linetmp[0] - '0') \* 10 + (linetmp[1] - '0')];

}

}

}

else if ((filenumb == 1) && (linetmp[0] == '\0')) {//control section 에 의해 만들어진 obj 파일이 아닐 경우

valtmp += csaddr;

}

else {//이름으로 주어질때

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

if (strcmp(estab[j].name, linetmp) == 0) {

if (flag == 0) {

valtmp += estab[j].address;

}

else {

valtmp -= estab[j].address;

}

break;

}

if (j == (estabnumb - 1)) {

printf("symbol is not in the estab\n");

fclose(fp);

proglen = 0;

return -1;

}

}

}

if (numtmp % 2 == 0) {//짝수 half byte

for (j = 0; j < (numtmp / 2); j++) {//메모리에 저장

memory[csaddr + addrtmp + (numtmp / 2) - 1 - j] = valtmp % 256;

valtmp = valtmp / 256;

}

}

else {//홀수 half byte

for (j = 0; j < (numtmp / 2); j++) {

memory[csaddr + addrtmp + (numtmp / 2) - j] = valtmp % 256;

valtmp = valtmp / 256;

}

memory[csaddr + addrtmp] -= memory[csaddr + addrtmp] % 16;

memory[csaddr + addrtmp] += valtmp % 16;

}

}

if (ch == 'E') {

fgets(linetmp, 300, fp);

if ((i == 0) && ((linetmp[0] == ' ') || (linetmp[0] == '\t') || (linetmp[0] == '\n'))) {

finst = 0;

}

if ((i == 0) && (linetmp[0] != ' ') && (linetmp[0] != '\t') && (linetmp[0] != '\n')) {

addrtmp = 0;

for (j = 0; j<6; j++) {

if ((linetmp[j] >= '0') && (linetmp[j] <= '9')) {

addrtmp = addrtmp \* 16 + linetmp[j] - '0';

}

else if ((linetmp[j] >= 'a') && (linetmp[j] <= 'f')) {

addrtmp = addrtmp \* 16 + linetmp[j] - 'a' + 10;

}

else if ((linetmp[j] >= 'A') && (linetmp[j] <= 'F')) {

addrtmp = addrtmp \* 16 + linetmp[j] - 'A' + 10;

}

else {

printf("obj file error\n");

fclose(fp);

proglen = 0;

return -1;

}

}

finst = addrtmp;//실행할 처음 instruction의 상대위치

}

}

if ((ch == '.') || (ch == '\n') || (ch == ' ') || (ch == '\t')) {

continue;

}

}

fclose(fp);

}

printf("\ncontrol symbol address length\n");

printf("section name\n");

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

total = 0;

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

if (estab[j].length != 0) {

printf("%-22s%-11X%04X\n", estab[j].name, estab[j].address, estab[j].length);

total += estab[j].length;

}

else {

printf(" %-11s%-11X\n", estab[j].name, estab[j].address);

}

}

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

printf(" total length %04X\n\n", total);

proglen = total;//프로그램의 전체 길이

ra = 0;

rx = 0;

rl = addr + proglen;

rpc = addr + finst;

rb = 0;

rs = 0;

rt = 0;

rf = 0;

rcc = 0;

for (j = 0; j < 0x100000; j++) {

memorytmp[j] = memory[j];

}

return 1;

}

void putbpnode(int val) {

struct bpnode \*new;

struct bpnode \*ptr;

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

new->checkpoint = val;

new->next = NULL;

if (bphead == NULL) {//처음 넣을 때

bphead = new;

printf("\n[ok] create breakpoint %X\n\n", val);

}

else {

ptr = bphead;

while (1) {

if (ptr->checkpoint == val) {//이전에 있던 breakpoint

printf("\nbreakpoint is already exist\n\n");

break;

}

if (ptr->next == NULL) {// 이전에 없던 breakpoint

ptr->next = new;

printf("\n[ok] create breakpoint %X\n\n", val);

break;

}

ptr = ptr->next;

}

}

}

int bp(int command) {

struct bpnode \* ptr;

struct bpnode \* tmp;

int i, a = 0;

if (command == 1) {//bp address command

if ((com[3] == '0')) {

if (com[4] == '\0') {//주소가 0인 경우

a = 0;

}

else if (com[4] == 'x') {// 주소가 0x로 입력된 경우

if (com[5] == '\0') {

printf("invalid address\n");

return -1;

}

else {

for (i = 5; i<strlen(com); i++) {//주소 계산

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

a = a \* 16 + com[i] - 'a' + 10;

}

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

a = a \* 16 + com[i] - '0';

}

else {

printf("invalid address\n");

return -1;

}

}

}

}

else {//0뒤에 x가 아닌 주소가 들어오는 경우

for (i = 3; i<strlen(com); i++) {

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

a = a \* 16 + com[i] - 'a' + 10;

}

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

a = a \* 16 + com[i] - '0';

}

else {

printf("invalid address\n");

return -1;

}

}

}

}

else {//주소가 0x없이 주어졌을 경우

for (i = 3; i<strlen(com); i++) {

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

a = a \* 16 + com[i] - 'a' + 10;

}

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

a = a \* 16 + com[i] - '0';

}

else {

printf("invalid address\n");

return -1;

}

}

}

if (a > 0xfffff) {

printf("invalid address\n");

return -1;

}

putbpnode(a);

}

else if (command == 2) {//bp command

if (bphead == NULL) {

printf("\nno breakpoint set.\n\n");

}

else {

printf("\nbreakpoints\n");

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

ptr = bphead;

while (1) {

if (ptr == NULL) {

break;

}

else {

printf("%X\n", ptr->checkpoint);

}

ptr = ptr->next;

}

}

}

else {//bp clear command

if (bphead == NULL) {

printf("\nalready cleared.\n\n");

}

else {

ptr = bphead;

tmp = ptr->next;

while (1) {

ptr->next = NULL;

free(ptr);

if (tmp == NULL) {

break;

}

ptr = tmp;

tmp = ptr->next;

}

bphead = NULL;

printf("\n[ok] clear all breakpoints\n\n");

}

}

return 1;

}

int run() {

int opcode, on, oi, ox, ob, op, oe, disp, r1, r2, i, flag, ta, tv;

struct bpnode \* ptr;

if (proglen == 0) {

printf("not loaded\n");

return -1;

}

while (1) {

opcode = memory[rpc] - (memory[rpc] % 4);

on = (memory[rpc] - opcode) / 2;

oi = (memory[rpc] - opcode) % 2;

if ((on == 0) && (oi == 0)) {//format2

r1 = memory[rpc + 1] / 16;

r2 = memory[rpc + 1] % 16;

if (opcode == 0xa0) {//COMPR

switch (r1) {

case 0: r1 = ra; break;

case 1: r1 = rx; break;

case 2: r1 = rl; break;

case 3: r1 = rb; break;

case 4: r1 = rs; break;

case 5: r1 = rt; break;

case 6: r1 = rf; break;

case 8: r1 = rpc; break;

case 9: r1 = rcc; break;

}

switch (r2) {

case 0: r2 = ra; break;

case 1: r2 = rx; break;

case 2: r2 = rl; break;

case 3: r2 = rb; break;

case 4: r2 = rs; break;

case 5: r2 = rt; break;

case 6: r2 = rf; break;

case 8: r2 = rpc; break;

case 9: r2 = rcc; break;

}

if (rl < r2) {

rcc = 0;

}

else if (r1 == r2) {

rcc = 1;

}

else {

rcc = 2;

}

rpc += 2;

}

else if (opcode == 0xb4) {//CLEAR

switch (r1) {

case 0: ra = 0; break;

case 1: rx = 0; break;

case 2: rl = 0; break;

case 3: rb = 0; break;

case 4: rs = 0; break;

case 5: rt = 0; break;

case 6: rf = 0; break;

case 8: rpc = 0; break;

case 9: rcc = 0; break;

}

rpc += 2;

}

else if (opcode == 0xb8) {//TIXR

rx++;

switch (r1) {

case 0: r1 = ra; break;

case 1: r1 = rx; break;

case 2: r1 = rl; break;

case 3: r1 = rb; break;

case 4: r1 = rs; break;

case 5: r1 = rt; break;

case 6: r1 = rf; break;

case 8: r1 = rpc; break;

case 9: r1 = rcc; break;

}

if (rx < r1) {

rcc = 0;

}

else if (rx == r1) {

rcc = 1;

}

else {

rcc = 2;

}

rpc += 2;

}

else {//not instruction

rpc++;

}

}

else {//format 3, 4

ox = memory[rpc + 1] / 128;

ob = (memory[rpc + 1] - (ox \* 128)) / 64;

op = (memory[rpc + 1] - (ox \* 128) - (ob \* 64)) / 32;

oe = (memory[rpc + 1] - (ox \* 128) - (ob \* 64) - (op \* 32)) / 16;

disp = ((memory[rpc + 1] % 16) \* 0x100) + memory[rpc + 2];

if (oe == 1) {

disp = disp \* 0x100 + memory[rpc + 3];

}

if ((ob == 1) && (op == 0)) {//base relative

ta = disp + rb;

}

else if ((op == 1) && (ob == 0)) {//PC relative

ta = disp + rpc + 3;

}

else if ((op == 0) && (ob == 0)) {//direct addressing

ta = disp;

}

else {//not instruction

rpc++;

continue;

}

if (ox == 1) {//indexed addressing

ta += rx;

}

if ((oe == 0) && (disp >= 0x800)) {//negative number

if (ta >= 0x1000) {

ta = ta - 0x1000;

}

}

if ((on == 1) && (oi == 1)) {//simple addressing

tv = (memory[ta] \* 0x10000) + (memory[ta + 1] \* 0x100) + memory[ta + 2];

}

else if ((on == 1) && (oi == 0)) {//indirect addressing

ta = (memory[ta] \* 0x10000) + (memory[ta + 1] \* 0x100) + memory[ta + 2];

tv = (memory[ta] \* 0x10000) + (memory[ta + 1] \* 0x100) + memory[ta + 2];

}

else if ((oi == 1) && (on == 0)) {//immediate addressing

tv = ta;

}

else {

rpc++;

continue;

}

if ((oe == 1) && (ob == 0) && (op == 0)) {//format4

rpc += 4;

}

else if (oe == 0) {//format3

rpc += 3;

}

else {//not instruction

rpc++;

continue;

}

if (opcode == 0x00) {//LDA

ra = tv;

}

else if (opcode == 0x68) {//LDB

rb = tv;

}

else if (opcode == 0x74) {//LDT

rt = tv;

}

else if (opcode == 0x50) {//LDCH

ra = ra - (ra % 0x100) + memory[ta];

}

else if (opcode == 0x0c) {//STA

memory[ta] = ra / 0x10000;

memory[ta + 1] = (ra % 0x10000) / 0x100;

memory[ta + 2] = ra % 0x100;

}

else if (opcode == 0x10) {//STX

memory[ta] = rx / 0x10000;

memory[ta + 1] = (rx % 0x10000) / 0x100;

memory[ta + 2] = rx % 0x100;

}

else if (opcode == 0x14) {//STL

memory[ta] = rl / 0x10000;

memory[ta + 1] = (rl % 0x10000) / 0x100;

memory[ta + 2] = rl % 0x100;

}

else if (opcode == 0x54) {//STCH

memory[ta] = ra % 0x100;

}

else if (opcode == 0x3c) {//J

rpc = ta;

}

else if (opcode == 0x48) {//JSUB

rl = rpc;

rpc = ta;

}

else if (opcode == 0x38) {//JLT

if (rcc == 0) {

rpc = ta;

}

}

else if (opcode == 0x30) {//JEQ

if (rcc == 1) {

rpc = ta;

}

}

else if (opcode == 0x4c) {//RSUB

rpc = rl;

}

else if (opcode == 0x28) {//COMP

if (ra < tv) {

rcc = 0;

}

else if (ra == tv) {

rcc = 1;

}

else {

rcc = 2;

}

}

else if (opcode == 0xe0) {//TD

rcc = 0;

}

else if (opcode == 0xd8) {//RD

ra = ra - (ra & 0x100);

}

else if (opcode == 0xdc) {//WD

}

else {// not instruction

if (oe == 1) {

rpc = rpc - 3;

}

else {

rpc = rpc - 2;

}

continue;

}

}

if (rpc == (addr + proglen)) {//end program

printf("\nA : %06X X : %06X\nL : %06X PC: %06X\nB : %06X S : %06X\nT : %06X\nEnd program\n\n", ra, rx, rl, rpc, rb, rs, rt);

for (i = 0; i < 0x100000; i++) {

memory[i] = memorytmp[i];

}

ra = 0;

rx = 0;

rl = addr + proglen;

rpc = addr + finst;

rb = 0;

rs = 0;

rt = 0;

rf = 0;

rcc = 0;

break;

}

ptr = bphead;

flag = 0;

while (1) {

if (ptr == NULL) {

break;

}

if (ptr->checkpoint == rpc) {

printf("\nA : %06X X : %06X\nL : %06X PC: %06X\nB : %06X S : %06X\nT : %06X\nStop at checkpoint[%X]\n\n", ra, rx, rl, rpc, rb, rs, rt, ptr->checkpoint);

flag = 1;

break;

}

ptr = ptr->next;

}

if (flag == 1) {

break;

}

}

return 1;

}