**과목명: 시스템프로그래밍**

**1 분반**

**<<Project #3>>**

**서강대학교 [컴퓨터 공학과]**

**[20171653]**

**[신나현]**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. int Linking\_Loader\_Pass1(char \*filename, int num)
      1. 사용 변수
      2. 모듈 기능
   2. int Linking\_Loader\_Pass2(char \*filename, int num)
      1. 사용 변수
      2. 모듈 기능
   3. void Reset\_ESTAB()
      1. 사용 변수
      2. 모듈 기능
   4. int Make\_ESTAB(char \*symbol, int csaddr, int cslth, int num)
      1. 사용 변수
      2. 모듈 기능
   5. est \*Find\_ESTAB(char \*symbol)
      1. 사용 변수
      2. 모듈 기능
   6. void Print\_ESTAB()
      1. 사용 변수
      2. 모듈 기능
   7. void Clear\_BreakPoint()
      1. 사용 변수
      2. 모듈 기능
   8. void Print\_BreakPoint()
      1. 사용 변수
      2. 모듈 기능
   9. void Set\_BreakPoint(int num)
      1. 사용 변수
      2. 모듈 기능
   10. void Run()
       1. 사용 변수
       2. 모듈 기능
   11. void Print\_Register()
       1. 사용 변수
       2. 모듈 기능
   12. hash \*Find\_Opcode\_By\_Num(int op)
       1. 사용 변수
       2. 모듈 기능
   13. void Store\_Reg(int addr, int value)
       1. 사용 변수
       2. 모듈 기능
4. **전역 변수 정의**
   1. int progaddr[5]
   2. int execaddr[5]
   3. int prog\_end
   4. int prog\_len
   5. int bp\_cnt
   6. int bp\_index
   7. int bp[50]
   8. int exec\_addr
5. **코드 설명**
   1. 20171653.h
   2. 20171653.c
6. **프로그램 개요**

프로젝트 1, 2 에서 구현한 셀(shell)에 linking과 loading 기능을 추가하는 프로그램이다. 프로젝트 2 에서 구현된 assemble 명령을 통해서 생성된 object 파일을 link시켜 메모 리에 올리는 일을 수행한다.

1. **프로그램 설명**
   1. 프로그램 흐름도

**텍스트, 화이트보드이(가) 표시된 사진

자동 생성된 설명**

1. **모듈 정의**
   1. **int Linking\_Loader\_Pass1(char \*filename, int num)**
      1. 사용 변수

int error: 에러가 있는지 여부를 저장

int csaddr: 컨트롤 섹션의 시작 주소 저장

int cslth: 각 섹션의 길이 저장

char str[100]: obj파일을 한줄씩 읽어온 내용을 저장하는 배열

char temp[100]: external symbiol이나 control section의 이름을 저장

char ad[10]: 주소를 배열 저장

int obj\_addr: 배열로 저장된 주소를 16진수로 저장

int len: str에 저장된 내용의 길이를 저장

FILE \*ofp : obj파일을 열어서 읽어오는데 사용

* + 1. 모듈 기능

obj 파일을 열어 linking loader의 pass1을 수행하면서 external symbol table을 만들고 에러가 없으면 0을 반환

* 1. **int Linking\_Loader\_Pass2(char \*filename, int num)**
     1. 사용 변수

char str[100]: obj파일을 한줄씩 읽어오는데 사용

char temp[100]: 내용을 임시로 저장하는데 사용

char temp2[10]: 내용을 임시로 저장하는데 사용

char \*token: strtok하는데 사용

int csaddr: 프로그램 시작 주소 저장

int cslth: 프로그램 길이 저장

int ref\_num: reference number

int memory\_hex: 내용을 16진수로 두글자씩 저장

int specified\_addr: 읽어온 str에서 주소를 저장함

int external\_addr[10]: ref\_num에 해당하는 심볼의 주소 저장

int i, j: 프로그램 수행과정에서 연산에 필요한 값 저장

int len: str에 저장된 문장의 길이 저장

int col, row: 메모리의 행,열 저장

int result: Modification의 경우 다시 계산된 주소값 저장

est\* exist\_sym: Find\_ESTAB 함수를 호출해 ESTAB에 존재하는 symbol이면 반환된 노드를 저장

FILE \*ofp: obj파일을 열어서 읽어오는데 사용

* + 1. 모듈 기능

linking loader의 pass2 수행하고 오류가 발생하지않으면 0을 반환

* 1. **void Reset\_ESTAB()**
     1. 사용 변수

est \*del: 링크드 리스트를 따라가는데 이용

* + 1. 모듈 기능

링크드리스트로 구현된 external symbol table을 reset시켜주는 함수

* 1. **int Make\_ESTAB(char \*symbol, int csaddr, int cslth, int num)**
     1. 사용 변수

est \*new: 새로운 심볼을 저장할 노드

est \*temp: 링크드리스트를 따라가는데 사용될 노드

* + 1. 모듈 기능

Find\_ESTAB 함수를 호출해 이미 존재하는 symbol인지 확인하고 이미 존재하면 1을 반환하고 존재하지않는 symbol인 경우 ESTAB에 추가해줌.

* 1. **est \*Find\_ESTAB(char \*symbol)**
     1. 사용 변수

est \*list: 링크드 리스트를 따라가는데 사용될 노드

est \*search: ESTAB에서 해당하는 symbol을 찾을 경우 그 노드를 저장

* + 1. 모듈 기능

링크드리스트로 구현된 external symbol table에서 symbol이 존재하는지 탐색하여 존재한다면 그 노드를, 존재하지 않는다면 NULL을 반환

* 1. **void Print\_ESTAB()**
     1. 사용 변수

est \*temp: 링크드리스트를 따라가는데 사용될 노드

* + 1. 모듈 기능

링크드리스트로 구현된 external symbol table을 탐색하여 출력해줌

* 1. **void Clear\_BreakPoint()**
     1. 사용 변수
     2. 모듈 기능

BreakPoint를 저장한 배열을 초기화해 sicsim에 존재하는 break point를 모두 삭제

* 1. **void Print\_BreakPoint()**
     1. 사용 변수
     2. 모듈 기능

sicsim에 존재하는 break point를 모두 화면에 출력

* 1. **void Set\_BreakPoint(int num)**
     1. 사용 변수
     2. 모듈 기능

break point를 저장하는 배열에 새로 입력된 수를 저장해 sicsim에 break point를 지정해줌

* 1. **void Run()**
     1. 사용 변수

int opcode: op에서 ni값을 제외한 opcode만을 저장

int op: 메모리에서 읽어온 값으로, ni값이 뒤에 포함된 값

int ni, xbpe, x, b, p, e: format3의 연산에서 사용되는 nixbpe 값 저장

int reg1, reg2: format2 연산에서 사용할 레지스터 저장

int \*point\_reg1, \*point\_reg2: reg1, reg2의 값에 해당하는 레지스터의 주소 값 저장

int disp, addr, loc, value;

char form[5]: 해당하는 opcode의 format을 저장

int col, row: 메모리의 행,열 저장

hash \*find: Find\_Opcode\_By\_Num 함수를 호출하여 opcode 해쉬 테이블에서 찾은 노드 저장

* + 1. 모듈 기능

메모리에 로드된 내용을 읽어와 해당하는 opcode에 맞는 연산을 실행해주면서 레지스터의 값을 변화시키며 프로그램을 실행해주는 함수

* 1. **void Print\_Register()**
     1. 사용 변수
     2. 모듈 기능

프로그램이 실행된 후 레지스터에 저장된 값을 출력해주는 함수

* 1. **hash \*Find\_Opcode\_By\_Num(int op)**
     1. 사용 변수

hash \*temp: 링크드 리스트를 따라가는데 사용되는 함수

* + 1. 모듈 기능

op에 해당하는 opcode값을 갖고있는 노드를 링크드리스트로 구현된 해쉬 테이블에서 탐색해 존재하면 그 노드를, 존재하지 않으면 NULL을 반환해줌

* 1. **void Store\_Reg(int addr, int value)**
     1. 사용 변수

int col, row: 메모리의 행, 열 저장

unsigned char change: 레지스터에 저장된 값을 메모리에 올리는데 사용됨

* + 1. 모듈 기능

mnemonic이 ST@인 경우에 메모리에 @레지스터의 값을 올려주는 함수.

1. **전역 변수 정의**
   1. **int progaddr[5]**

각각의 object파일의 메모리 시작 주소를 저장해줌

* 1. **int execaddr[5]**

pass2에서 각각의 실행주소를 저장해줌

* 1. **int prog\_end**

모든 오브젝트 파일을 로드한 메모리의 끝 주소를 저장

* 1. **int prog\_len**

프로그램의 길이 저장

* 1. **int bp\_cnt**

break point의 개수 저장

* 1. **int bp\_index**

break point의 index 저장

* 1. **int bp[50]**

break point를 저장해 두는 배열

* 1. **int exec\_addr**

Run할 때 실행 주소를 저장

1. **코드 설명**
   1. **20171653.h**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<dirent.h>

#include<sys/stat.h>

typedef struct ExternalTable{

char symbol[100];

int addr;

int len;

struct ExternalTable \*next;

}est;

typedef struct SymbolTable{

char symbol[100];

int num;

struct SymbolTable \*next;

}symbol;

typedef struct HashTable{

int opcode;

char command[100];

char format[100];

struct HashTable \*next;

}hash;

typedef struct History{

char command[100];

struct History \*next;

}history;

void Get\_Command(); //명령어 입력받는 함수

void Read\_Opcode(); //opcode.txt읽어오는 함수

int Hash\_Function(char \*arr); //hash값 생성

int Print\_Opcode(char \*arr); //mnemonic에 해당하는 opcode 출력

void Opcode\_List(); //opcode hash table 출력

void Print\_All\_Command(); //h[elp]

void Current\_Directory\_File(); //d[ir]

void Quit\_Sicsim(); //q[uit]

void View\_History(); //hi[story]

void Push\_Command(char \*arr); //history를 링크드리스트 형태로 저장

void dump(int start, int end); //du[mp]

void Edit(int addr, int value); //e[dit]

void Fill(int start, int end, int value); //f[ill]

void Reset(); //reset

int Print\_File(char \*arr); //type filename

void Reset\_SymbolTable(); //symbl table reset해주는 함수

int Assemble\_File(char \*arr); //assemble filename

int Pass1(char\*filename); //location counter와 .obj파일의 한줄 길이 결정

int Pass2(char\*filename); //.lst, .obj파일 생성해 내용저장

void Make\_SymbolTable(char \*arr, int addr); //assemble하면서 읽어들인 symbol을 table에 추가

symbol \*Check\_Duplicate(char \*arr); //symbol이 이미 symbol table에 저장된지 확인

hash\* Find\_Opcode(char \*arr); //opcode table에 해당하는 opcode존재여부 확인

void Print\_SymbolTable(); //가장 최근에 assemble된 symbol table출력

int Format4(int opcode, int reg, int addr); //format4의 objcode생성해 반환

int Format3(int opcode, int reg, int addr); //format3의 objcode생성해 반환

int Format2(int opcode, char reg1, char reg2); //format2의 objcode생성해 반환

void PasteSymbol(int sw); //assemble에 실패하는경우에 대비해 symboltable을 복사해두는 함수

int Linking\_Loader\_Pass1(char \*filename, int num); //Load의 Pass1을 실행하는 함수

int Linking\_Loader\_Pass2(char \*filename, int num); //Load의 Pass2를 실행하는 함수

void Reset\_ESTAB(); //external symbol table을 초기화 해주는 함수

int Make\_ESTAB(char \*symbol, int csaddr, int cslth, int num); //external symbol table에 추가해주는 함수

est \*Find\_ESTAB(char \*symbol); //ESTAB에 이미 존재하면 그 노드를 반환하는 함수

void Print\_ESTAB(); //ESTAB을 출력하는 함수

void Clear\_BreakPoint(); //bp clear명령어 수행하는 함수 bp를 모두 지워줌

void Print\_BreakPoint(); //sicsim에 존재하는 bp 모두 출력

void Set\_BreakPoint(int num); //입력된 bp추가

void Run(); //메모리에 load된 프로그램 실행

void Print\_Register(); //실행 결과인 레지스터 값 출력

hash \*Find\_Opcode\_By\_Num(int op); //opcode번호로 해쉬테이블에서 해당하는 노드 찾아서 반환

void Store\_Reg(int addr, int value); //symbol이 STR\_인 경우 저장해줌

* 1. **20171653.c**

#include "20171653.h"

/\*

typedef struct ExternalTable{

char symbol[100];

int addr;

int len;

struct ExternalTable \*next;

}est;\*/

unsigned char memory[16][65536];

history \*head;

hash \*hashhead[20];

symbol \*symbolhead[26];

symbol \*symbolkeep[26];

est \*estab[10];

int obj\_len[100]={0};

int dumpend=-1;

int dumpstart;

int program\_len;

int base\_addr;

int error\_line;

int progaddr[5];

int execaddr[5];

int prog\_end=0;

int prog\_len =0;

int bp\_cnt = 0;

int bp\_index=0;

int bp[50] = {0};

int exec\_addr=0;

int A =0 ,X = 0, L = 0, PC = 0, B = 0, S = 0, T = 0 ,CC=0;

/\*

int Linking\_Loader\_Pass1(char \*filename, int num);

int Make\_ESTAB(char \*symbol, int csaddr, int cslth, int num);

est \*Find\_ESTAB(char \*symbol);

void Print\_ESTAB();\*/

/\*

main함수\*/

int main(void){

Read\_Opcode();

progaddr[0] = 0;

while(1){

Get\_Command();

}

}

/\*

Print\_All\_Command()

h[elp]

shell에서 실행가능한 모든 명령어 리스트 출력

\*/

void Print\_All\_Command(){

printf("h[elp]\n");

printf("d[ir]\n");

printf("q[uit]\n");

printf("hi[story]\n");

printf("du[mp] [start, end]\n");

printf("e[dit] address, value\n");

printf("f[ill] start, end, value\n");

printf("reset\n");

printf("opcode mnemonic\n");

printf("opcodelist\n");

printf("assemble filename\n");

printf("type filename\n");

printf("symbol\n");

printf("progaddr address\n");

printf("loader object filename1 object filename2 ...\n");

printf("bp address\n");

printf("bp clear\n");

printf("bp\n");

printf("run\n");

}

/\*

Current\_Directory\_File()

d[ir]

현재 디렉터리의 파일 출력

\*/

void Current\_Directory\_File(){

DIR \*directory=opendir("./");

struct dirent \*di;

struct stat statbuf;

if(directory !=NULL){

while((di = readdir(directory))!=NULL){

lstat(di->d\_name,&statbuf);

if(S\_ISREG(statbuf.st\_mode))

printf("%s\*\t", di->d\_name);

else if(S\_ISDIR(statbuf.st\_mode)&&S\_IXUSR)

printf("%s/\t", di->d\_name);

}

printf("\n");

}

closedir(directory);

}

/\*

Quit\_Sicsim()

q[uit]

링크드리스트로 구현된 history와 opcode hash table을 모두 free하여 삭제해준 뒤 sicsim 종료

\*/

void Quit\_Sicsim(){

history \*del;

hash \*hashdel;

while(head != NULL){

del = head;

head = head->next;

free(del);

}

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

while(hashhead[i]!=NULL){

hashdel = hashhead[i];

hashhead[i] = hashhead[i]->next;

free(hashdel);

}

}

Reset\_ESTAB();

exit(0);

}

/\*

View\_History()

hi[story]

링크드르스트로 구현된 history를 head부터 따라가면서 사용한 명령어 순서대로 번호와 함께 보여줌

\*/

void View\_History(){

int num=1;

history \*temp = head;

while(temp!=NULL){

printf("%d %s\n", num, temp->command);

temp = temp->next;

num++;

}

}

/\*

Push\_Command(char \*arr)

command를 history의 링크드리스트의 맨 뒤에 삽입함.

\*/

void Push\_Command(char \*arr){

history \*save = NULL;

history \*temp;

save = (history\*)malloc(sizeof(history));

strcpy(save->command, arr);

save->next=NULL;

if(head == NULL){

head = save;

}

else{

temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = save;

}

}

/\*

dump(int start, int end)

du[mp] start, end

start번지부터 end번지까지의 메모리의 내용을 <출력하는 메모리의 주소를 5자리 16진수 형태로, 메모리의 내용을 16진수 형태로, 메모리의 내용에 대응하는 ASCII code형태>로 출력해줌

start번지와 end번지가 있는 줄의 각각 첫번지~start-1번지와 end+1번지~끝번지는 16진수로 출력해주지않고 ASCII코드부분역시 '.' 으로 출력해주므로 case를 나눠 출력해줌

\*/

void dump(int start, int end){

int startaddr=start/16;

int endaddr=end/16;

for(int i= startaddr; i<=endaddr; i++){

printf("%05X ", i\*16);

if(startaddr == endaddr){ //start와 end 번지가 한줄에 있는경우

for(int j=0; j<start%16; j++){

printf(" ");

}

for(int j=start%16; j<=end%16; j++){

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

}

for(int j=end%16+1; j<16; j++){

printf(" ");

}

printf("; ");

for(int j=0; j<start%16; j++){

printf(".");

}

for(int j=start%16; j<=end%16; j++){

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

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

}

else{

printf(".");

}

}

for(int j=end%16+1; j<16; j++){

printf(".");

}

}

else if(i== startaddr){ //start번지가 있는 줄

for(int j=0; j<start%16; j++){

printf(" ");

}

for(int j=start%16; j<16; j++){

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

}

printf("; ");

for(int j=0; j<start%16; j++){

printf(".");

}

for(int j=start%16; j<16; j++){

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

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

}

else{

printf(".");

}

}

}

else if(i== endaddr){ //end번지가 있는 줄

for(int j=0; j<=end%16; j++){

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

}

for(int j=end%16+1; j<16; j++){

printf(" ");

}

printf("; ");

for(int j=0; j<=end%16; j++){

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

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

}

else{

printf(".");

}

}

for(int j=end%16+1; j<16; j++){

printf(".");

}

}

else{ //중간

for(int j=0; j<16; j++){

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

}

printf("; ");

for(int j=0; j<16; j++){

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

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

}

else{

printf(".");

}

}

}

printf("\n");

}

}

/\*

Edit(int addr, int value)

e[dit]

메모리에서 해당 address의 값을 value로 바꿔 줌\*/

void Edit(int addr, int value){

memory[addr%16][addr/16]=value;

}

/\*

Fill(int start, int end, int value)

f[ill]

for문에서 Edit함수를 호출하여 start에서 end번지까지 value로 바꿔 줌\*/

void Fill(int start, int end, int value){

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

Edit(i, value);

}

}

/\*

Reset()

reset

for문에서 Edit함수를 호출하여 메모리를 전부 0으로 바꿔 줌 \*/

void Reset(){

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

Edit(i, 0);

}

}

/\*

Read\_Opcode()

opcode.txt 읽어오는 함수

fopen으로 opcode.txt를 열어와서 파일의 끝까지 읽어온다.

읽어온 내용을 opcode hash table에 링크드리스트 형태로 저장해준다.

\*/

void Read\_Opcode(){

int n;

int opcode;

char mnemonic[100];

char num[100];

hash \*new = NULL;

hash \*temp;

FILE \*ofp = fopen("opcode.txt", "r");

if(ofp == NULL){

printf("opcode.txt ERROR!"); //opcode파일이 없으면 ERROR출력

return;

}

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

hashhead[i] = NULL;

}

while(fscanf(ofp, "%X%s%s", &opcode, mnemonic, num)!=EOF){

new = (hash\*)malloc(sizeof(hash));

new->opcode = opcode;

strcpy(new->command, mnemonic);

strcpy(new->format, num);

n=Hash\_Function(mnemonic); //저장할 hash값을 return받아온다.

if(hashhead[n] == NULL){

hashhead[n] = new;

}

else{

temp = hashhead[n];

while(temp->next != NULL){

temp = temp->next;

}

temp->next = new;

}

//printf("%X %s %s\n", new->opcode, new->command, new->number);

}

fclose(ofp); //파일을 닫음

}

/\*

Hash\_Function(char \*arr)

hash 값 결정해서 return해줌

\*/

int Hash\_Function(char \*arr){

int sum=0;

for(int i=0; i<strlen(arr); i++){

sum+=(int)arr[i];

}

return sum%20;

}

/\*

Print\_Opcode(char \*arr)

opcode mnemonic

명령어에 해당하는 opcode를 출력하기 위해 링크드리스트로 구현된 hash table을 탐색한다

유효한 mnemonic인 경우 그에 해당하는 opcode를 출력하고 1을 return하고

유효하지않은 mnemonic인 경우 0을 return 한다

\*/

int Print\_Opcode(char \*arr){

char mnemonic[100];

char \*token;

int num;

hash \*temp;

token = strtok(arr, " ");

token = strtok(NULL," ");

strcpy(mnemonic, token);

num = Hash\_Function(mnemonic);

temp = hashhead[num];

while(strcmp(temp->command, mnemonic) != 0){

temp = temp->next;

if(temp == NULL){

return 0;

}

}

printf("opcode is %X\n", temp->opcode);

return 1;

}

/\*

hash\* Find\_Opcode(char \*arr)

arr이 opcode table에 존재하면 그 opcode가 있는 노드를 반환하고 존재하지않으면 0을 반환

\*/

hash \*Find\_Opcode(char \*arr){

hash \*temp;

int num;

num = Hash\_Function(arr);

temp = hashhead[num];

while(strcmp(temp->command, arr) != 0){

temp = temp->next;

if(temp == NULL){

return NULL;

}

}

return temp;

// strcpy(form, temp->format);

// return (form[0]-48);

}

/\*

Opcode\_List()

opcodelist

링크드리스트로 구현된 opcode hash table을 차례로 탐색하면서 그 내용 출력해준다\*/

void Opcode\_List(){

hash \*temp;

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

printf("%d : ", i);

temp = hashhead[i];

if(temp != NULL){

while(temp->next!=NULL){

printf("[%s,%X] -> ", temp->command, temp->opcode);

temp = temp->next;

}

printf("[%s,%X]", temp->command, temp->opcode);

}

printf("\n");

}

}

/\*

int Print\_File(char \*arr)

type filename

현재 디렉터리에 해당 파일이 존재한다면 읽어서 화면에 출력하고 1을 반환

해당파일이 존재하지 않으면 0을 반환

\*/

int Print\_File(char \*arr){

char filename[100];

char \*token;

FILE \*ofp=NULL;

char get;

token = strtok(arr, " ");

token = strtok(NULL, " ");

strcpy(filename, token);

ofp = fopen(filename, "r");

if(ofp == NULL) //해당 파일이 존재하지 않음

{

printf("FILE DOES NOT EXIST!\n");

return 0;

}

//while(fread(&get, sizeof(char), 1, ofp)){

while(fscanf(ofp, "%c", &get)!=EOF){

printf("%c", get); //파일의 내용출력

}

return 1;

}

/\*

void Reset\_SymbolTable()

symbol table을 모두 초기화해주는 함수

\*/

void Reset\_SymbolTable(){

symbol \*symboldel;

for(int i=0; i<26; i++){

while(symbolhead[i]!=NULL){

symboldel = symbolhead[i];

symbolhead[i] = symbolhead[i]->next;

free(symboldel);

}

}

}

/\*

void PasteSymbol(int sw)

assemble 도중 오류가 있는경우 그전에 제대로 assemble된 경우의 symbol table 출력을 위해 복사해두는 함수

sw가 1인 경우엔 symbolkeep에 symbolhead를 복사

sw가 0인 경우엔 symbolhead에 symbolkeep을 복사\*/

void PasteSymbol(int sw){

symbol \*temp;

symbol \*new;

symbol \*paste;

if(sw ==0){ //assemble 중 오류

Reset\_SymbolTable();

for(int i=0; i<26; i++){

temp = symbolkeep[i];

paste = symbolhead[i];

while(temp!=NULL){

new = (symbol\*)malloc(sizeof(symbol));

new->num = temp->num;

strcpy(new->symbol, temp->symbol);

new->next = NULL;

if(symbolhead[i]==NULL){

symbolhead[i] = new;

paste = symbolhead[i];

temp = temp->next;

continue;

}

else{

paste->next = new;

}

paste = paste->next;

temp = temp->next;

}

}

}

else if(sw == 1){ //assemble성공

symbol \*symboldel;

for(int i=0; i<26; i++){

while(symbolkeep[i]!=NULL){

symboldel = symbolkeep[i];

symbolkeep[i] = symbolkeep[i]->next;

free(symboldel);

}

}

for(int i=0; i<26; i++){

temp = symbolhead[i];

paste = symbolkeep[i];

while(temp!=NULL){

new = (symbol\*)malloc(sizeof(symbol));

new->num = temp->num;

strcpy(new->symbol, temp->symbol);

new->next = NULL;

if(symbolkeep[i]==NULL){

symbolkeep[i] = new;

temp = temp->next;

paste = symbolkeep[i];

continue;

}

else{

paste->next = new;

}

paste = paste->next;

temp = temp->next;

}

}

}

}

/\*

void Make\_SymbolTable(char \*arr, int addr)

symbol table에 symbol을 추가해

\*/

void Make\_SymbolTable(char \*arr, int addr){

symbol \*new=NULL;

symbol \*temp;

symbol \*prev;

symbol \*sw;

int hash;

sw = Check\_Duplicate(arr); //symbol의 중복여부 확인

if(sw!=NULL) //중복되는 경우

return;

hash = arr[0]-65;

new = (symbol\*)malloc(sizeof(symbol));

strcpy(new->symbol, arr);

new->num = addr;

new->next = NULL;

/\*새로운 노드 추가\*/

if(symbolhead[hash]==NULL){

symbolhead[hash]=new;

}

else{

temp = symbolhead[hash];

prev = temp;

while(temp->next!=NULL){

if(strcmp(temp->symbol, arr)>0){ //알파벳 내림차순으로 정렬하기위해 string비교

prev->next = new;

new->next = temp;

return;

}

prev = temp;

temp=temp->next;

}

if(strcmp(temp->symbol, arr)>0){ //링크드 리스트의 마지막 node의앞에 위치시켜야 하는 경우

if(prev == temp){ //head가 가리키는게 마지막 리스트였을 경우

symbolhead[hash]=new;

new->next=temp;

//printf("a\n");

return ;

}

prev->next = new;

new->next = temp;

return;

}

temp->next = new;

}

}

/\*

symbol\* Check\_Duplicate(char \*arr)

이미 symbol table에 저장된 symbol인지 확인

저장되어있는경우 그 노드를 반환, 아닌경우 NULL을 반환

\*/

symbol \* Check\_Duplicate(char \*arr){

symbol \*check = symbolhead[arr[0]-65];

while(check != NULL){

if(strcmp(arr, check->symbol)==0){

return check;

}

check = check->next;

}

return NULL;

}

/\*

int Assemble\_File(char\*arr)

file name에 해당하는 소스 파일을 읽어서 오브젝트파일과 리스팅파일을 만듦

\*/

int Assemble\_File(char\*arr){

char filename[100];

char\* token = NULL;

int sw;

token = strtok(arr, " ");

token = strtok(NULL, " ");

strcpy(filename, token);

for(int i=0; i<100; i++){

obj\_len[i]=0;

}

sw=Pass1(filename); //pass1 실행

if(sw==0){ //pass1 실행중 에러 발견

return 0;

}

sw = Pass2(filename); //pass2 실행

if(sw == 0){ //pass2 실행중 에러 발견

return 0;

}

return 1;

}

/\*

int Pass1(char\*filename)

pass1실행해 location해 location counter계산

\*/

int Pass1(char\* filename){

FILE\*ofpr,\*ofpw;

char str[100];

char mnemonic[100];

char sym[100];

char temp[50];

char \*token;

int line = 5;

int location=0;

int len;

int form;

hash \*opcode;

int operand;

int obj\_cnt=0;

int obj\_line=0;

symbol \*use;

ofpr = fopen(filename, "r");

if(ofpr==NULL){

printf("FILE DOES NOT EXIST!\n"); //.asm 파일 존재하지않음

return 0;

}

ofpw = fopen("loc.txt", "w");

fgets(str, sizeof(str), ofpr);

sscanf(str,"%s %s",temp, mnemonic); //START가 저장되어야함

if(strcmp(mnemonic, "START")!=0){ //START저장되어있지않음

fclose(ofpr);

fclose(ofpw);

error\_line = line;

return 0;

}

fprintf(ofpw, "%d %d\n", line, location);

line+=5;

fgets(str, sizeof(str), ofpr);

sscanf(str+7, "%s", mnemonic);

while(strcmp(mnemonic, "END") != 0){ //END나올 때 까지

len = strlen(str);

fprintf(ofpw, "%d %d\n", line, location);

line+=5;//line은 5 증가

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

}

else{

if(str[0]!=' '){ //symbol이 존재하는 경우symbol table에 저장

sscanf(str, "%s", sym);

use = Check\_Duplicate(sym); //중복 확인

if(use != NULL){

printf("SYMBOL ERROR!\n"); //같은 label이 두번 나오는 경우는 에러

fclose(ofpr);

fclose(ofpw);

error\_line = line-5;

return 0;

}

Make\_SymbolTable(sym, location);

//printf("%s\n",temp);

}

if(strcmp(mnemonic,"BYTE") == 0){ //BYTE

strncpy(temp, str+14, len-14);

if(temp[0]=='C'){ //C

token = strtok(temp+1, "'");

location += strlen(token);

if(obj\_cnt + strlen(token)>30){

obj\_len[obj\_line++]=obj\_cnt;

obj\_cnt=0;

}

obj\_cnt += strlen(token);

}

else if(temp[0]=='X'){ //X

token = strtok(temp+1, "'");

location += strlen(token)/2;

if(obj\_cnt + strlen(token)/2 >30){

obj\_len[obj\_line++] = obj\_cnt;

obj\_cnt=0;

}

obj\_cnt +=strlen(token)/2;

}

else{

printf("INVALID OPERAND!\n");

fclose(ofpr);

fclose(ofpw);

error\_line = line-5;

return 0;

}

}

else if(strcmp(mnemonic,"WORD")==0){ //WORD

location += 3;

if(obj\_cnt + 3 >30){

obj\_len[obj\_line++] = obj\_cnt;

obj\_cnt=0;

}

obj\_cnt += 3;

}

else if(strcmp(mnemonic, "RESB")==0){ //RESB

sscanf(str+14, "%d", &operand);

//printf("%d\n",operand);

location += operand;

if(obj\_cnt !=0){

obj\_len[obj\_line++] = obj\_cnt;

obj\_cnt=0;

}

}

else if(strcmp(mnemonic, "RESW")==0){ //RESW

sscanf(str+14, "%d", &operand);

location +=operand\*3;

if(obj\_cnt !=0){

obj\_len[obj\_line++] = obj\_cnt;

obj\_cnt=0;

}

}

else if(strcmp(mnemonic, "BASE")==0){ //BASE

}

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

sscanf(mnemonic+1, "%s", temp);

opcode = Find\_Opcode(temp);

if(opcode ==NULL){ //opcode.txt에 존재하지 않는 mnemonic이 들어옴

printf("OPCODE ERROR!\n");

fclose(ofpr);

fclose(ofpw);

error\_line = line-5;

return 0;

}

else{

location += 4;

if(obj\_cnt + 4 > 30){

obj\_len[obj\_line++] = obj\_cnt;

obj\_cnt = 0;

}

obj\_cnt += 4;

}

}

else{ //format 1, 2, 3

opcode = Find\_Opcode(mnemonic);

if(opcode == NULL){ //opcode.txt에 존재하지 않는 mnemonic이 들어

printf("OPCODE ERROR!\n");

fclose(ofpr);

fclose(ofpw);

error\_line = line-5;

return 0;

}

else{

form = opcode->format[0]-48;

location += form;

if(obj\_cnt+form > 30){

obj\_len[obj\_line++] = obj\_cnt;

obj\_cnt=0;

}

obj\_cnt += form;

}

}

}

fgets(str, sizeof(str), ofpr);

sscanf(str+7,"%s", mnemonic);

}

program\_len = location;

obj\_len[obj\_line] = obj\_cnt;

fclose(ofpr); //파일 닫아줌

fclose(ofpw);

return 1;

}

/\*

int Pass2(char\*filename)

pass2 실행해 .lst파일과 .obj파일 생성

\*/

int Pass2(char\*filename){

FILE \*ofp, \*loc, \*ofpl, \*ofpj;

char str[100];

char sym[100];

char opcode[100];

char operand[10];

char temp[100]; //format4일때 사용

int modi[100];

char \*token;

int line, location;

int len;

int imme;

int comma=0;

int flag=1;

int obj;

int form;

int pc;

int cnt = 0;

int obj\_cnt=0;

int obj\_line=0;

hash \*now;

symbol \*use;

ofp = fopen(filename, "r");

loc = fopen("loc.txt", "r");

strcpy(str, filename); //.lst파일 생성

token = strtok(str, ".");

strcpy(str, token);

strcat(str, ".lst");

ofpl = fopen(str, "w");

strcpy(str, filename); //.obj파일 생성

token = strtok(str, ".");

strcpy(str, token);

strcat(str, ".obj");

ofpj = fopen(str, "w");

fgets(str, sizeof(str), ofp);

fscanf(loc, "%d %d", &line, &location);

//printf("aa\n");

sscanf(str, "%s", sym);

len = strlen(sym);

for(int i=0; i<6-len; i++){

strcat(sym," ");

}

base\_addr = 0;

len = strlen(str);

str[len-1] = '\0';

fprintf(ofpl, "%d\t%04X\t%s\n", line, location, str); //START

fprintf(ofpj, "H%s%06X%06X", sym, 0, program\_len);

//printf("%d\t%04X\t%s\n", line, location, str);

fgets(str, sizeof(str), ofp);

fscanf(loc, "%d %d", &line, &location);

sscanf(str+7, "%s", opcode);

while(strcmp(opcode,"END")!=0){

// now = (hash\*)malloc(sizeof(hash));

comma=0; //operand의 콤마 수 초기화

len = strlen(str);

str[len-1] = '\0';

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

fprintf(ofpl, "%d\t\t%s\n", line, str);

}

else{

if(strcmp(opcode, "WORD")==0){ //WORD

//printf("aaa");

//strncpy(operand, str+14, 10);

if(sscanf(str+14, "%d", &imme)==1){

//printf("aa");

if(imme>=0x000000 && imme <=0xffffff){ //범위 확인

fprintf(ofpl, "%d\t%04X\t%s\t\t%06X\n",line, location, str, imme);

if(obj\_cnt + 3 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

fprintf(ofpj, "%06X", imme);

obj\_cnt += 3;

}

else{

printf("INVALID OPERAND!\n");

flag=0;

error\_line = line;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag= 0;

error\_line = line;

break;

}

}

else if(strcmp(opcode, "RESW")==0){ //RESW

fprintf(ofpl,"%d\t%04X\t%s\n", line, location, str);

if(obj\_cnt !=0){

obj\_cnt=0;

}

}

else if(strcmp(opcode, "RESB")==0){ //RESB

fprintf(ofpl,"%d\t%04X\t%s\n", line, location, str);

if(obj\_cnt !=0){

obj\_cnt=0;

}

}

else if(strcmp(opcode, "BYTE")==0){ //BYTE

strncpy(operand, str+14, len-14);

if(operand[0]=='C'){ //C

token = strtok(operand+1, "'");

fprintf(ofpl, "%d\t%04X\t%s\t\t", line, location, str);

for(int i=0; i<strlen(token); i++){

fprintf(ofpl, "%02X", token[i]);

}

fprintf(ofpl, "\n");

if(obj\_cnt + strlen(token) > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

for(int i=0; i<strlen(token); i++){

fprintf(ofpj, "%02X", token[i]);

}

obj\_cnt += strlen(token);

}

else if(operand[0]=='X'){ //X

token = strtok(operand+1, "'");

fprintf(ofpl, "%d\t%04X\t%s\t\t", line, location, str);

for(int i=0; i<strlen(token); i++){

fprintf(ofpl,"%c",token[i]);

}

fprintf(ofpl, "\n");

if(obj\_cnt + strlen(token)/2 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

for(int i=0; i<strlen(token); i++){

fprintf(ofpj,"%c",token[i]);

}

obj\_cnt +=strlen(token)/2;

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else if(strcmp(opcode, "BASE")==0){ //BASE

fprintf(ofpl,"%d\t\t%s\n", line, str);

sscanf(str+14, "%s", operand);

use = Check\_Duplicate(operand);

base\_addr = use->num;

//printf("%06X\n", base\_addr);

}

else if(strcmp(opcode, "RSUB")==0){ //RSUB

//printf("[%s]\n", opcode);

fprintf(ofpl, "%d\t%04X\t%s\t\t\t%06X\n", line, location, str,0x4F0000);

if(obj\_cnt + 3 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

fprintf(ofpj, "%06X",0x4F0000);

obj\_cnt += 3;

}

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

sscanf(opcode+1, "%s", temp);

now = Find\_Opcode(temp);

if(now!=NULL){

strncpy(operand, str+14, 10);

for(int i=0; i<strlen(operand); i++){

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

comma++;

}

}

token = strtok(operand, ".\n");

strcpy(operand, token);

// printf("%s\n", operand);

if(comma == 1){ //simple

token = strtok(operand, ",");

use = Check\_Duplicate(token);

if(use!=NULL){ //nixbpe = 111001

obj = Format4(now->opcode, 0x39, use->num);

modi[cnt]=location;

cnt++;

fprintf(ofpl, "%d\t%04X\t%s\t\t%08X\n", line, location, str, obj);

if(obj\_cnt + 4 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=4;

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else if(comma ==0){

if(operand[0]=='#'){ //immediate

sscanf(token, "%s", operand);

if(sscanf(operand+1,"%d", &imme) !=1){

use = Check\_Duplicate(operand+1);

if(use != NULL){ //nixbpe = 010001

obj = Format4(now->opcode,0x11, use->num);

fprintf(ofpl, "%d\t%04X\t%s\t\t%08X\n", line, location, str, obj);

modi[cnt]=location;

cnt++;

if(obj\_cnt + 4 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=4;

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else{

sscanf(operand+1, "%d", &imme); //nixbpe = 010001

obj = Format4(now->opcode, 0x11, imme);

fprintf(ofpl, "%d\t%04X\t%s\t\t%08X\n", line, location, str, obj);

if(obj\_cnt + 4 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=4;

}

}

else if(operand[0] == '@'){ //indirect

sscanf(token,"%s", operand);

use = Check\_Duplicate(operand+1);

if(use!=NULL){ //nixbpe = 100001

obj = Format4(now->opcode, 0x21, use->num);

fprintf(ofpl, "%d\t%04X\t%s\t\t%08X\n", line, location, str, obj);

modi[cnt]=location;

cnt++;

if(obj\_cnt + 4 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=4;

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else{ //simple format4는 pc base 사용X

sscanf(token, "%s", operand);

use = Check\_Duplicate(operand);

if(use !=NULL){

obj = Format4(now->opcode, 0x31, use->num); //nixbpe = 110001

fprintf(ofpl, "%d\t%04X\t%s\t\t%08X\n", line, location, str, obj);

modi[cnt]=location;

cnt++;

if(obj\_cnt + 4 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=4;

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else{

now = Find\_Opcode(opcode);

if(now!=NULL){

form = now->format[0]-48;

strncpy(operand, str+14, 10);

token = strtok(operand,".\0");

for(int i=0; i<strlen(operand); i++){

if(operand[i]==',')

comma++;

} //operand안의 comma수 세기

if(comma == 1){ //comma 한개

if(form == 2){ //format2

obj = Format2(now->opcode, operand[0], operand[3]);

fprintf(ofpl, "%d\t%04X\t%s\t\t%04X\n", line, location, str, obj);

if(obj\_cnt + 2 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=2;

}

else if(form == 3){ //format3

token = strtok(operand, ",");

use = Check\_Duplicate(token);

if(use != NULL){

pc = location+3;

if(use->num - pc >= -2048 && use->num - pc <=2047){ //pc relative

obj = Format3(now->opcode, 0x3A, use->num - pc); //nixbpe = 111010

fprintf(ofpl, "%d\t%04X\t%s\t\t%06X\n", line, location, str, obj);

if(obj\_cnt + 3 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=3;

}

else if(use->num - base\_addr >= 0 && use->num-base\_addr < 4096){ //base relative

obj = Format3(now->opcode, 0x3C, use->num-base\_addr); //nixbpe = 111100

fprintf(ofpl, "%d\t%04X\t%s\t\t%06X\n", line, location, str, obj);

if(obj\_cnt + 3 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=3;

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag =0;

error\_line = line;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else if (comma == 0){

if(form == 1){ //format1

obj = now->opcode;

fprintf(ofpl, "%d\t%04X\t%s\t\t%06X\n", line, location, str, obj);

if(obj\_cnt + 1 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt += 1;

}

else if(form == 2){ //format2

strncpy(operand, str+14, 10);

//if(strcmp(opcode,"CLEAR")==0 || strcmp(opcode,"TIXR")==0){

//printf("%c\n", operand[0]);

obj = Format2(now->opcode, operand[0], 0);

fprintf(ofpl, "%d\t%04X\t%s\t\t\t%04X\n", line, location, str, obj);

if(obj\_cnt + 2 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=2;

//}

}

else if(form == 3){ //format3

sscanf(operand,"%s", temp);

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

if(sscanf(temp+1,"%d", &imme) != 1){ //숫자아님

sscanf(temp+1, "%s", operand);

use = Check\_Duplicate(operand);

if(use!=NULL){

pc = location+3;

if(use->num - pc >= -2048 && use->num - pc <=2047){ //pc relative

obj = Format3(now->opcode, 0x12, use->num - pc); //nixbpe = 010010

fprintf(ofpl, "%d\t%04X\t%s\t\t%06X\n", line, location, str, obj);

if(obj\_cnt + 3 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=3;

}

else if(use->num - base\_addr >= 0 && use->num-base\_addr < 4096){ //base relative

obj = Format3(now->opcode, 0x14, use->num-base\_addr); //nixbpe = 010100

fprintf(ofpl, "%d\t%04X\t%s\t\t%06X\n", line, location, str, obj);

if(obj\_cnt + 3 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=3;

}

else{

//printf("b\n");

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else{

//printf("a\n");

printf("INVALID OPERAND!\n");

flag =0;

error\_line = line;

break;

}

}

else{ //숫자

if(imme >= -2048 && imme<=2047){

obj = Format3(now->opcode, 0x10, imme); //nixbpe = 010000

fprintf(ofpl, "%d\t%04X\t%s\t\t%06X\n", line, location, str, obj);

if(obj\_cnt + 3 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=3;

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

}

else if(temp[0]=='@'){ //indirect

sscanf(temp+1, "%s", operand);

use = Check\_Duplicate(operand);

if(use!=NULL){

pc = location+3;

if(use->num - pc >= -2048 && use->num - pc <=2047){ //pc relative

obj = Format3(now->opcode, 0x22, use->num - pc); //nixbpe = 100010

fprintf(ofpl, "%d\t%04X\t%s\t\t%06X\n", line, location, str, obj);

if(obj\_cnt + 3 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=3;

}

else if(use->num - base\_addr >= 0 && use->num-base\_addr < 4096){ //base relative

obj = Format3(now->opcode, 0x24, use->num-base\_addr); //nixbpe = 100100

fprintf(ofpl, "%d\t%04X\t%s\t\t%06X\n", line, location, str, obj);

if(obj\_cnt + 3 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=3;

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else{ //simple

use = Check\_Duplicate(temp);

if(use!=NULL){

pc = location+3;

if(use->num - pc >= -2048 && use->num - pc <=2047){ //pc relative

obj = Format3(now->opcode, 0x32, use->num - pc); //nixbpe = 110010

fprintf(ofpl, "%d\t%04X\t%s\t\t%06X\n", line, location, str, obj);

if(obj\_cnt + 3 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=3;

}

else if(use->num - base\_addr >= 0 && use->num-base\_addr < 4096){ //base relative

obj = Format3(now->opcode, 0x34, use->num-base\_addr); //nixbpe = 110100

fprintf(ofpl, "%d\t%04X\t%s\t\t%06X\n", line, location, str, obj);

if(obj\_cnt + 3 > 30 || obj\_cnt == 0){

fprintf(ofpj, "\nT%06X%02X", location, obj\_len[obj\_line]);

obj\_cnt=0;

obj\_line++;

}

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

obj\_cnt +=3;

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

error\_line = line;

break;

}

}

}

fgets(str, sizeof(str), ofp);

fscanf(loc, "%d %d", &line, &location);

sscanf(str+7, "%s", opcode);

// free(now);

}

fprintf(ofpl,"%d\t\t%s", line+5, str);

for(int i=0; i<cnt; i++){ //modification record 출력

fprintf(ofpj,"\nM%06X05", modi[i]+1);

}

fprintf(ofpj, "\nE%06X\n", 0x000000);

fclose(ofp);

fclose(loc);

fclose(ofpl);

fclose(ofpj);

if(flag == 1)

printf("Successfully assemble %s\n", filename);

else{ //중간에 에러 존재

strcpy(str, filename); //.lst파일 삭제

token = strtok(str, ".");

strcpy(str, token);

strcat(str, ".lst");

remove(str);

strcpy(str, filename); //.obj파일 삭제

token = strtok(str, ".");

strcpy(str, token);

strcat(str, ".obj");

remove(str);

}

return flag;

}

/\*

[format4]

opcode, addr을 통해 objcode생성

\*/

int Format4(int opcode, int reg, int addr){

int obj=0;

addr&=0xFFFFF;

opcode <<= 24;

obj |= opcode;

reg <<= 20;

obj |= reg;

obj |= addr;

return obj;

}

/\*

[format3]

opcode, nixbpe, addr통해 objcode 생성\*/

int Format3(int opcode, int reg, int addr){

int obj = 0;

addr &=0xFFF;

opcode <<=16;

obj |= opcode;

reg <<= 12;

obj |= reg;

obj |= addr;

return obj;

}

/\*

[format2]

opcode, reg1, reg2 통해 objcode생성\*/

int Format2(int opcode, char reg1, char reg2){

int obj = 0;

int r1=0, r2=0;

if(reg1 == 'A')

r1 = 0;

else if(reg1 == 'X')

r1 = 1;

else if(reg1 == 'L')

r1 = 2;

else if(reg1 == 'B')

r1 = 3;

else if(reg1 == 'S')

r1 = 4;

else if(reg1 == 'T')

r1 = 5;

else if(reg1 == 'F')

r1 = 6;

if(reg2 == 'A')

r2 = 0;

else if(reg2 == 'X')

r2 = 1;

else if(reg2 == 'L')

r2 = 2;

else if(reg2 == 'B')

r2 = 3;

else if(reg2 == 'S')

r2 = 4;

else if(reg2 == 'T')

r2 = 5;

else if(reg2 == 'F')

r2 = 6;

opcode <<= 8;

obj |= opcode;

r1 <<= 4;

obj |= r1;

obj |= r2;

return obj;

}

/\*

void Print\_SymbolTable()

assemble한 파일의 symbol table 출력\*/

void Print\_SymbolTable(){

symbol \*temp;

for(int i=0; i<26; i++){

temp = symbolhead[i];

while(temp != NULL){

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

temp = temp->next;

}

}

//Reset\_SymbolTable();

}

/\*

int Linking\_Loader\_Pass1(char \*filename, int num)

obj 파일을 열어 linking loader의 pass1을 수행함

에러가 없으면 0을 반환

\*/

int Linking\_Loader\_Pass1(char \*filename, int num){

int error = 0;

int csaddr, cslth;

char str[100]={0};

char temp[100]={0};

char ad[10]={0};

int obj\_addr;

int len;

int i;

FILE \*ofp = fopen(filename, "r");

if(ofp == NULL){

printf("obj file open ERROR!\n");

return 1;

}

csaddr = progaddr[num];

//while(fgets(str, sizeof(str), ofp) != NULL){

if(fgets(str, sizeof(str), ofp)== NULL){

printf("obj file is Empty!\n");

return 1;

}

else{

if(str[0] == 'H'){ //Header

sscanf(str, "%s", temp);

sscanf(str+strlen(temp)+7, "%X", &cslth);

sscanf(temp+1, "%s", temp);

//printf("%s! %x!\n", temp, cslth);

error = Make\_ESTAB(temp, csaddr, cslth, num);

if(error == 1){

printf("Pass1 ERROR!\n");

return 1;

}

}

while(str[0] != 'E'){ //End가 아닐 동안

if(fgets(str, sizeof(str), ofp) == NULL)

break;

if(str[0] == 'D'){ //Define

len = strlen(str);

i=0;

while(i<len-2){ //external symbol 저장

sscanf(str+i+1, "%s", temp);

strncpy(ad, str+i+7, 6);

sscanf(ad, "%X", &obj\_addr);

error = Make\_ESTAB(temp, obj\_addr+ csaddr, 0, num);

i+=12;

if(error == 1){

printf("Pass1 ERROR!\n");

return 1;

}

}

}

}

progaddr[num+1] = csaddr + cslth;

}

//progaddr[num+1] = csaddr + cslth;

fclose(ofp);

return 0;

}

/\*

int Linking\_Loader\_Pass2(char \*filename, int num)

linking loader의 pass2 수행함

오류가 발생하지않으면 0을 반환\*/

int Linking\_Loader\_Pass2(char \*filename, int num){

char str[100];

char temp[100];

char temp2[10];

char \*token;

int csaddr, cslth;

int ref\_num, memory\_hex;

int specified\_addr;

int external\_addr[10]= {0};

int i, j, len, col, row;

int result;

est\* exist\_sym;

FILE \*ofp = fopen(filename, "r");

if(ofp == NULL){

printf("obj file open ERROR!\n");

return 1;

}

csaddr = progaddr[num];

execaddr[num] = progaddr[num];

external\_addr[1] = csaddr;

if(fgets(str, sizeof(str), ofp)==NULL){

printf("obj file is Empty!\n");

return 1;

}

else{

if(str[0] == 'H'){ //Header

sscanf(str, "%s", temp);

sscanf(str+strlen(temp)+7, "%X", &cslth); //길이 저장

}

while(str[0]!='E'){

if(fgets(str, sizeof(str), ofp) == NULL)

break;

len = strlen(str);

if(str[0] == 'R'){ //Reference Number

for(i=1; i<len-2; i+=8){

strncpy(temp2, str +i, 2);

sscanf(temp2, "%X", &ref\_num); //ref num 저장

token = strtok(str+i+2, " \n");

//printf(" !%s! ", token);

exist\_sym = Find\_ESTAB(token); //ESTAB에 존재하는 symbol인지 확인

if(exist\_sym == NULL){

printf("Pass2 : Symbol does not exist!\n");

return 1;

}

external\_addr[ref\_num] = exist\_sym->addr; //사용할 ref num에 해당하는 주소 저장

}

}

else if(str[0] == 'T'){ //Text

strncpy(temp2, str+1, 6);

strcpy(temp2+6, "\0");

sscanf(temp2, "%X", &specified\_addr); //주소저장

specified\_addr += csaddr;

strncpy(temp2, str+7, 2);

strcpy(temp2+2, "\0");

sscanf(temp2, "%X", &len); //길이 저장

//printf("%X\n", len);

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

col = specified\_addr % 16;

row = specified\_addr / 16;

strncpy(temp2, str+9+i\*2, 2);

strcpy(temp2+2, "\0");

sscanf(temp2,"%X", &memory\_hex); //두글자씩 16진수로

memory[col][row] = memory\_hex; //메모리에 저장

specified\_addr ++;

}

}

else if(str[0] == 'M'){ //Modification

result = 0;

j=0;

strncpy(temp2, str+1, 6);

strcpy(temp2+6, "\0");

sscanf(temp2, "%X", &specified\_addr); //주소 저장

specified\_addr += csaddr;

strncpy(temp2, str+7, 2);

strcpy(temp2+2, "\0");

sscanf(temp2, "%X", &len); //길이 저장

strncpy(temp2, str+10, 2);

strcpy(temp2+2, "\0");

sscanf(temp2, "%X", &ref\_num); //reference number 저장

//printf("%d %d %x\n", num,ref\_num,external\_addr[ref\_num]);

if(len == 5){ //길이가 5

for(i = specified\_addr +2 ; i > specified\_addr; i--){

col = i%16;

row = i/16;

result += memory[col][row] << j;

j+=8;

}

col = specified\_addr%16;

row = specified\_addr/16;

result += (memory[col][row] &0x0F) <<j; //5BYTE

}

else if(len == 6){ //길이가 6

for(i= specified\_addr + 2; i>=specified\_addr; i--){

col = i%16;

row = i/16;

result += memory[col][row] << j;

j+=8;

}

} //6BYTE

if(str[9] == '+'){ //'+'

result += external\_addr[ref\_num];

}

else if(str[9] == '-'){ //'-'

result -= external\_addr[ref\_num];

}

result &= 0xFFFFFF;

if(len == 5){

for(i = specified\_addr+2; i>specified\_addr; i--){

col = i%16;

row = i/16;

j = 0xFF;

memory[col][row] = (result & j);

result >>= 8;

} //메모리에 2BYTE씩 다시 저장

col = specified\_addr%16;

row = specified\_addr/16;

//j = 0x0F;

//result = result & j;

j = 0xF0;

memory[col][row] &= j;

//memory[col][row] |= result;

j=0x0F;

memory[col][row] |= (result&j);

result >>= 4;

}

else if(len == 6){

for(i = specified\_addr +2; i>= specified\_addr; i--){

col = i%16;

row = i/16;

j= 0xFF;

memory[col][row] = (result &j);

result >>= 8;

}

} //메모리에 2BYTE씩 다시 저장

}

}

if(str[0] == 'E'){ //End

strncpy(temp2, str+1, 6);

strcpy(temp2+6, "\0");

if(sscanf(temp2, "%X", &specified\_addr)==1){

execaddr[num] = csaddr+specified\_addr;

}

}

}

fclose(ofp);

return 0;

}

/\*

void Reset\_ESTAB()

ESTAB을 reset 시켜주는 함수\*/

void Reset\_ESTAB(){

est \*del;

for(int i=0; i<10; i++){

while(estab[i]!=NULL){

del = estab[i];

estab[i] = estab[i]->next;

free(del);

}

//progaddr[i]=0;

//execaddr[i]=0;

}

}

/\*

int Make\_ESTAB(char \*symbol, int csaddr, int cslth)

ESTAB만들어준다. 이미 있는 symbol이면 1을 반환

\*/

int Make\_ESTAB(char \*symbol, int csaddr, int cslth, int num){

est \*new;

est \*temp;

if(Find\_ESTAB(symbol) != NULL){ //이미 존재하는 symbol

return 1;

}

new = (est \*)malloc(sizeof(est));

strcpy(new->symbol, symbol);

new->addr = csaddr;

new->len = cslth;

new->next = NULL;

if(estab[num] == NULL){

estab[num] = new;

}

else{

temp = estab[num];

while(temp->next != NULL){

temp = temp->next;

}

temp->next = new;

}

return 0;

}

/\*

int Find\_ESTAB(char \*symbol)

ESTAB에서 symbol 찾으면 그 링크드리스트 반환\*/

est \*Find\_ESTAB(char \*symbol){

est \*list;

est \*search=NULL;

for(int i=0; i<10; i++){

if(search != NULL){

break;

}

list = estab[i];

while(list!=NULL){

if(strcmp(list->symbol, symbol)==0){

search = list;

break;

}

list = list->next;

}

}

return search;

}

/\*

void Print\_ESTAB()

load map 출력해줌\*/

void Print\_ESTAB(){

est \*temp;

prog\_len=0;

printf("control\tsymbol\taddress\tlength\n");

printf("section\tname\n");

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

for(int i=0; i<10; i++){

temp = estab[i];

while(temp != NULL){

if(temp->len != 0){

printf("%s\t\t%04X\t%04X\n", temp->symbol, temp->addr, temp->len);

prog\_len+=temp->len;

}

else{

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

}

temp = temp->next;

}

}

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

printf("\ttotal length\t%04X\n", prog\_len);

prog\_end = progaddr[0] + prog\_len;

//printf("%04X\n", prog\_end);

}

/\*

void Set\_BreakPoint()

sicsim에 break point를 지정\*/

void Set\_BreakPoint(int num){

//int temp;

bp[bp\_cnt] = num;

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

bp\_cnt++;

}

/\*

void Clear\_BreakPoint()

sicsim에 존재하는 break point를 모두 삭제\*/

void Clear\_BreakPoint(){

for(int i=0; i<bp\_cnt; i++){

bp[i] = 0;

}

bp\_cnt=0;

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

}

/\*

void Print\_BreakPoint()

sicsim에 존재하는 break point를 전부 화면에 출력\*/

void Print\_BreakPoint(){

printf("\t\tbreakpoint\n");

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

for(int i=0; i<bp\_cnt; i++){

printf("\t\t%X\n", bp[i]);

}

}

/\*

메모리에 load된 프로그램 실행\*/

void Run(){

// int start\_addr;

// int end\_addr;

int opcode;

int op;

int ni, xbpe, x, b, p, e;

int reg1, reg2;

int \*point\_reg1, \*point\_reg2;

int disp, addr, loc, value;

char form[5];

int col, row;

hash \*find;

PC = exec\_addr;

//printf("PC : %X\n", PC);

//printf("%x %x\n", PC, end\_addr);

/\* op n i x b p e addr\*/

while(PC >= progaddr[0] && PC < prog\_end){

//printf("%X\n", PC);

col = PC%16;

row = PC/16;

op = memory[col][row];

opcode = op & 0xFC; //1111 1100

ni = op & 0x03; //0000 0011

//printf("%02X\n", opcode);

find = Find\_Opcode\_By\_Num(opcode);

if(find == NULL){

strcpy(form, "0");

}

else{

strcpy(form, find ->format);

}

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

PC +=3;

}

else{

if(form[0] == '1'){ //format1

PC++;

}

else if(form[0] == '2'){ //format2

PC++;

col = PC % 16;

row = PC / 16;

reg1 = memory[col][row];

reg2 = reg1 & 0x0F;

reg1 = (reg1 >> 4) & 0x0F;

PC++;

//printf("format 2\n");

switch (opcode) {

case 0x90: //ADDR r1, r2

break;

case 0xB4: //CLEAR r1

switch (reg1){

case 0x00:

A=0;

break;

case 0x01:

X=0;

break;

case 0x02:

L=0;

break;

case 0x03:

B=0;

break;

case 0x04:

S=0;

break;

case 0x05:

T=0;

break;

case 0x08:

PC=0;

break;

}

//printf("clear! %x\n", reg1);

break;

case 0xA0: //COMPR r1, r2

switch (reg1){

case 0x00:

point\_reg1 = &A;

break;

case 0x01:

point\_reg1 = &X;

break;

case 0x02:

point\_reg1 = &L;

break;

case 0x03:

point\_reg1 = &B;

break;

case 0x04:

point\_reg1 = &S;

break;

case 0x05:

point\_reg1 = &T;

break;

case 0x08:

point\_reg1 = &PC;

break;

}

switch (reg2){

case 0x00:

point\_reg2 = &A;

break;

case 0x01:

point\_reg2 = &X;

break;

case 0x02:

point\_reg2 = &L;

break;

case 0x03:

point\_reg2 = &B;

break;

case 0x04:

point\_reg2 = &S;

break;

case 0x05:

point\_reg2 = &T;

break;

case 0x08:

point\_reg2 = &PC;

break;

}

if(\*point\_reg1 == \*point\_reg2)

CC=0;

else if(\*point\_reg1 > \*point\_reg2)

CC=1;

else

CC=-1;

break;

case 0x9C: //DIVR r1, r2

break;

case 0x98: //MULR r1, r2

break;

case 0xAC: //RMO r1, r2

break;

case 0xA4: //SHIFTVTL r1, n

break;

case 0x94: //SUBR r1, r2

break;

case 0xB0: //SVC n

break;

case 0xB8: //TIXR r1

X +=1;

switch (reg1){

case 0x00:

point\_reg1 = &A;

break;

case 0x01:

point\_reg1 = &X;

break;

case 0x02:

point\_reg1 = &L;

break;

case 0x03:

point\_reg1 = &B;

break;

case 0x04:

point\_reg1 = &S;

break;

case 0x05:

point\_reg1 = &T;

break;

case 0x08:

point\_reg1 = &PC;

break;

}

if(X>\*point\_reg1) CC=1;

else if(X==\*point\_reg1) CC=0;

else CC=-1;

break;

}

}

else if(form[0] == '3'){ //format 3/4

PC++;

col = PC % 16;

row = PC / 16;

xbpe = memory[col][row];

e = (xbpe >> 4) % 2;

p = (xbpe >> 5) % 2;

b = (xbpe >> 6) % 2;

x = (xbpe >> 7) % 2; //xbpe구해준다.

disp = xbpe & 0x0F;

if (e == 1){ //format4

//printf("format 4\n");

form[0] = '4';

PC ++;

col = PC % 16;

row = PC / 16;

xbpe = memory[col][row];

disp = (disp << 8) + xbpe;

PC++;

col = PC % 16;

row = PC / 16;

xbpe = memory[col][row];

disp = (disp << 8) + xbpe;

}

else{

PC++;

col = PC % 16;

row = PC / 16;

xbpe = memory[col][row];

disp = (disp << 8) +xbpe;

//printf("%03X\n", disp);

}

PC++;

addr = disp;

if(p==1){

if((addr>>11)%2==1){

addr = PC - (0xFFF - addr + 1); //2의 보수

}

else{

addr += PC;

}

}

else if(b==1)

addr += B;

if (x==1)

addr += X;

loc = addr;

switch (ni){

case 1: //01 immediate

value = addr;

break;

case 2: //10 indirect

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

value = 0;

col = loc % 16;

row = loc / 16;

xbpe = memory[col][row];

value += xbpe;

loc++;

col = loc % 16;

row = loc / 16;

xbpe = memory[col][row];

value = (value << 8) + xbpe;

loc++;

col = loc % 16;

row = loc / 16;

xbpe = memory[col][row];

value = (value<<8) + xbpe;

loc++;

if(i==0){

loc = value;

addr = value;

}

}

//printf("\*\*%X %X\n",value, addr);

break;

case 3: //11 simple

value = 0;

col = loc % 16;

row = loc / 16;

xbpe = memory[col][row];

value += xbpe;

loc++;

col = loc % 16;

row = loc / 16;

xbpe = memory[col][row];

value = (value <<8) + xbpe;

loc++;

col = loc % 16;

row = loc / 16;

xbpe = memory[col][row];

value = (value <<8 ) + xbpe;

loc++;

break;

}

switch (opcode){

case 0x18: //ADD m

A += value;

break;

case 0x40: //AND m

A = A & value;

break;

case 0x28: //COMP m

if(A == value)

CC=0;

else if(A < value)

CC = -1;

else

CC = 1;

break;

case 0x24: //DIV m

A = A/value;

break;

case 0x3C: //J m

PC = addr;

break;

case 0x30: //JEQ m

if(CC == 0)

PC = addr;

break;

case 0x34: //JGT m

if(CC == 1)

PC = addr;

break;

case 0x38: //JLT m

if(CC == -1){

PC = addr;

}

//printf("%X\n", PC);

break;

case 0x48: //JSUB m

L = PC;

PC = addr;

break;

case 0x00: //LDA m

A = value;

break;

case 0x68: //LDB m

B = value;

break;

case 0x50: //LDCH m

//A = A&0xFF00;

A = (A >>8)<<8;

value = (unsigned)(value >> 16);

A += value;

break;

case 0x08: //LDL m

L = value;

break;

case 0x6C: //LDS m

S = value;

break;

case 0x74: //LDT m

T = value;

break;

case 0x04: //LDX m

X = value;

break;

case 0x20: //MUL m

A = A \* value;

break;

case 0x44: //OR m

A = A |value;

break;

case 0xD8: //RD m

CC = 1;

break;

case 0x4C: //RSUB

PC = L;

break;

case 0x0C: //STA m

loc -=3;

Store\_Reg(loc, A);

break;

case 0x78: //STB m

loc -=3;

Store\_Reg(loc, B);

break;

case 0x54: //STCH m

loc -=3;

value = A & (unsigned)0xFF;

memcpy(memory+loc, &value, 1);

//Store\_Reg(loc, A);

break;

case 0x14: //STL m

loc -=3;

Store\_Reg(loc, L);

break;

case 0x7C: //STS m

loc -=3;

Store\_Reg(loc, S);

break;

case 0x84: //STT m

loc -=3;

Store\_Reg(loc, T);

break;

case 0x10: //STX m

loc -=3;

Store\_Reg(loc, X);

break;

case 0x1C: //SUB m

A = A-value;

break;

case 0xE0: //TD m

CC = -1;

break;

case 0x2C: //TIX m

X +=1;

if( X == value) CC =0;

else if(X < value) CC = -1;

else CC=1;

break;

case 0xDC: //WD m

break;

}

}

}

//printf("%s %s\n", find->command, find->format);

//

int f=0;

for(int i=0; i<bp\_cnt; i++){

if(bp[i] == PC){

exec\_addr = PC;

//printf("!!!%x\n", PC);

Print\_Register();

printf("\t\tStop at checkpoint[%X]\n", bp[i]);

f=1;

break;

}

}

if(f==1)

break;

}

//Print\_Register();

bp\_index++;

if(bp\_index > bp\_cnt){

bp\_index = 0;

Print\_Register();

printf("\t\tEnd Program\n");

exec\_addr =0;

}

}

/\*

void Store\_Reg(int addr, int value)

ST\_같은 symbol 처리하기위한 함수\*/

void Store\_Reg(int addr, int value){

int col, row;

unsigned char change;

change = value >> 16;

change = change & 0xFF;

col = addr%16;

row = addr/16;

memory[col][row] = change;

addr++;

change = value >> 8;

change = change & 0xFF;

col = addr%16;

row = addr/16;

memory[col][row] = change;

addr++;

change = value & 0xFF;

col = addr%16;

row = addr/16;

memory[col][row] = change;

}

/\*

hash \*Find\_Opcode\_By\_Num(int op)

opcode 번호로 해쉬테이블에서 해당하는 노드 찾음\*/

hash \*Find\_Opcode\_By\_Num(int op){

hash \*temp;

//temp = (hash\*)malloc(sizeof(hash));

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

temp = hashhead[i];

if(temp != NULL){

while(op != temp -> opcode){

temp = temp->next;

if(temp == NULL)

break;

}

if(temp != NULL){

return temp;

}

}

}

return NULL;

}

/\*

void Print\_Register

레지스터 출력해줌\*/

void Print\_Register(){

printf("A : %06X\t X : %06X\n", A, X);

printf("L : %06X\tPC : %06X\n", L, PC);

printf("B : %06X\t S : %06X\n", B, S);

printf("T : %06X\n", T);

}

/\*

Get\_Command()

command를 입력받아 유효한 명령어인 경우 각각의 명령어에 맞는 함수를 실행해 주고

유효하지 않은 경우엔 ERROR를 출력해 준다\*/

void Get\_Command(){

char str[100]={0};

char save[100]={0};

char use[100]={0};

char command[100] = {0};

char check[100]={0};

char filename[100]={0};

int num[5]={-1, -1, -1,-1,-1};

int i,j, comma=0;

int len;

char \*temp=NULL;

int cnt=0;

int sw;

int bp;

printf("sicsim> ");

fgets(str, sizeof(str), stdin); //명령어 입력받음

len = strlen(str);

str[len-1] = '\0';

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

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

comma++;

}

}//dump [start,end], edit, fill 인경우엔 0이 아님

strcpy(save, str); //history에 push하기위해 원래 명령 그대로 저장

temp = strtok(str, ", ");

strcpy(command, temp); //명령어의 첫단어만 잘라서 저장

//printf("%s", command);

temp = strtok(NULL, ", ");

while(temp != NULL){

sscanf(temp, "%X", &num[cnt]); //숫자부분 저장

cnt++;

temp = strtok(NULL,", ");

}

if(strcmp(command,"h")==0 || strcmp(command, "help")==0){

if(cnt!=0)

printf("command ERROR\n");

else{

Push\_Command(save);

Print\_All\_Command();

}

}// h[elp]

else if(strcmp(command, "q")==0 || strcmp(command,"quit")==0){

if(cnt!=0)

printf("command ERROR!\n");

else{

Push\_Command(save);

Quit\_Sicsim();

}

}//q[uit]

else if(strcmp(command, "hi")==0 || strcmp(command, "history")==0){

if(cnt!=0)

printf("command ERROR!\n");

else{

Push\_Command(save);

View\_History();

}

}//hi[story]

else if(strcmp(command, "d")==0 || strcmp(command, "dir")==0){

if(cnt!=0){

printf("command ERROR!\n");

}

else{

Push\_Command(save);

Current\_Directory\_File();

}

}//d[ir]

else if(strcmp(command, "opcode")==0 && cnt==1){

//Push\_Command(save);

strcpy(use, save);

sw = Print\_Opcode(use);

if(sw == 1){

Push\_Command(save); //opcode.txt에 존재하는 mnemonic일때만 history에 삽입

}

else{

printf("command ERROR!\n"); //존재하지않는 mnemonic인 경우 ERROR 출력

}

}//opcode mnemonic

else if(strcmp(command, "opcodelist")==0 && cnt==0){

Push\_Command(save);

Opcode\_List();

}//opcodelist

else if(strcmp(command, "dump")==0 || strcmp(command, "du")==0){

if(cnt>2){

printf("command ERROR!\n");

}

else{

if(comma>=2){

printf("command ERROR!\n");

}

else if(comma==1 && cnt==2){ //dump start, end

if(num[0]!=-1 && num[1]!=-1){

if((num[0]<=num[1]) && (num[0] >=0) &&(num[1]>=0) && (num[0]<=0xFFFFF) && (num[1]<=0xFFFFF)){

Push\_Command(save);

dump(num[0], num[1]);

dumpend = num[1];

dumpstart = num[0];

}

else{

printf("command ERROR!\n");

}

}

else{

printf("command ERROR!\n");

}

}

else if(comma==0){

if(num[0]==-1 && cnt==0){ //dump

Push\_Command(save);

dumpstart = dumpend+ 1;

if(dumpstart > 0xFFFFF){

dumpstart=0;

}

dumpend = dumpstart+159;

if(dumpend > 0xFFFFF){

dumpend = 0xFFFFF;

}

dump(dumpstart, dumpend);

}

else if(num[0]!=-1 && cnt==1){ //dump start

if(num[0]>=0 && num[0]<=0xFFFFF){

Push\_Command(save);

dumpstart = num[0];

dumpend = dumpstart+159;

if(dumpend>0xFFFFF){

dumpend=0xFFFFF;

}

dump(dumpstart, dumpend);

}

else{

printf("command ERROR!\n");

}

}

else{

printf("command ERROR!\n");

}

}

}

}//du[mp]

else if(strcmp(command, "e")==0 || strcmp(command, "edit")==0){

if(comma==1 && cnt==2){

if(num[0]>=0 && num[0]<=0xFFFFF && num[1]>=0 && num[1]<=0xFF){

Push\_Command(save);

Edit(num[0], num[1]);

}

else{

printf("command ERROR!\n");

}

}

else{

printf("command ERROR!\n");

}

}//e[dit]

else if(strcmp(command,"f")==0 || strcmp(command, "fill")==0){

if(comma == 2 && cnt==3){

if(num[0]>=0 && num[0]<=0xFFFFF && num[1]>=0 && num[1]<=0xFFFFF && num[0]<=num[1] && num[2]>=0 && num[2]<=0xFF){

Push\_Command(save);

Fill(num[0], num[1], num[2]);

}

else{

printf("command ERROR!\n");

}

}

else{

printf("command ERROR!\n");

}

}//f[ill]

else if(strcmp(command, "reset")==0 && cnt==0){

Push\_Command(save);

Reset();

}//reset

else if(strcmp(command,"type")==0 && cnt==1){

strcpy(use, save);

sw = Print\_File(use);

if(sw==1){

Push\_Command(save); //해당파일이 존재할 때만 명령을 history에 저장

}

}//type filename

else if(strcmp(command, "assemble")==0 && cnt==1){

Reset\_SymbolTable();

strcpy(use, save);

sw = Assemble\_File(use);

if(sw==1){

Push\_Command(save);

}

else{

printf("line %d :ASSEMBLE ERROR!\n", error\_line);

}

PasteSymbol(sw);

remove("loc.txt"); //추가로 사용한 txt파일 삭제

}//assemble filename

else if(strcmp(command, "symbol")==0 && cnt ==0){

Print\_SymbolTable();

Push\_Command(save);

}//symbol

else if(strcmp(command, "progaddr")==0 && cnt == 1){

if(num[0] < 0 || num[0] > 0xFFFFF){ //주소가 범위 벗어남

printf("시작 주소 지정 ERROR!\n");

}

else{

Reset\_ESTAB();

progaddr[0] = num[0];

Push\_Command(save);

}

//printf("%05X\n",progaddr);

}//주소 지정 명령어

else if(strcmp(command, "loader")==0 && cnt !=0){

//////pass1//////

i= strlen(command)+1;

j = 0;

Reset\_ESTAB();

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

sscanf(save+i, "%s", filename);

i += strlen(filename) + 1;

sw = Linking\_Loader\_Pass1(filename, j);

j++;

//

//printf("%s!\n", filename);

if(sw == 1){

printf("Linking Loader Pass1 ERROR!\n");

}

}

////pass2/////

if(sw==0){

i=strlen(command)+1;

j = 0;

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

sscanf(save+i, "%s", filename);

i += strlen(filename) + 1;

sw = Linking\_Loader\_Pass2(filename, j);

j++;

if(sw == 1){

printf("Linking Loader Pass2 ERROR!\n");

}

}

}

if(sw==0){

Push\_Command(save);

Print\_ESTAB();

///load후 레지스터 초기화////

A = 0;

X = 0;

L = prog\_len;

PC = progaddr[0];

B = 0;

S = 0;

T = 0;

//printf("%X %X\n", L, PC);

}

//Print\_ESTAB();

}//Lingking Loader

else if(strcmp(command, "run")==0){

Run();

Push\_Command(save);

}//프로그램 실행 run

else if(strcmp(command, "bp")==0 && cnt == 1){

strcpy(use, save);

temp = strtok(use, " ");

temp = strtok(NULL, " ");

strcpy(check, temp);

if (strcmp(check, "clear")==0){

Clear\_BreakPoint();

}//break point 모두 삭제

else{

sscanf(check, "%X", &bp);

Set\_BreakPoint(bp);

}// break point 지정

Push\_Command(save);

}

else if(strcmp(command, "bp")==0 && cnt==0){

Print\_BreakPoint();

Push\_Command(save);

}//break point 전부 출력

else{

printf("command ERROR!\n");

}

}