**과목명: 시스템프로그래밍**

**1 분반**

**<<Project #2>>**

**서강대학교 [컴퓨터 공학과]**

**[20171653]**

**[신나현]**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
3. **모듈 정의**
   1. Print\_File(char \*arr)
      1. 사용 변수
      2. 모듈 기능
   2. Reset\_SymbolTable()
      1. 사용 변수
      2. 모듈 기능
   3. Assemble\_File(char \*arr)
      1. 사용 변수
      2. 모듈 기능
   4. Pass1(char\*filename)
      1. 사용 변수
      2. 모듈 기능
   5. Pass2(char\*filename)
      1. 사용 변수
      2. 모듈 기능
   6. Make\_SymbolTable(char \*arr, int addr)
      1. 사용 변수
      2. 모듈 기능
   7. Check\_Duplicate(char \*arr)
      1. 사용 변수
      2. 모듈 기능
   8. Find\_Opcode(char \*arr)
      1. 사용 변수
      2. 모듈 기능
   9. Print\_SymbolTable()
      1. 사용 변수
      2. 모듈 기능
   10. Format4(int opcode, int reg, int addr)
       1. 사용변수
       2. 모듈 기능
   11. Format3(int opcode, int reg, int addr)
       1. 사용변수
       2. 모듈 기능
   12. Format2(int opcode, char reg1, char reg2)
       1. 사용변수
       2. 모듈 기능
   13. PasteSymbol(int sw)
       1. 사용 변수
       2. 모듈 기능
4. **전역 변수 정의**
   1. symbol \*symbolhead[26]
   2. symbol \*symbolkeep[26]
   3. int obj\_len[100]
   4. int program\_len
   5. int base\_addr
5. **코드 설명**
   1. 20171653.h
   2. 20171653.c
6. **프로그램 개요**

프로젝트1에서 구현한 셀(shell)에 assemble 기능을 추가하는 프로그램. SIC/XE의 assembly program source 파일을 입력 받아서 object파일을 생성하고, 어셈블리 과정 중 생성된 symbol table과 결과물인 object 파일을 볼 수 있는 기능을 제공해야 함. 교재의 2.2까지 설명된 SIC/XE 어셈블러의 기능을 구현함을 원칙으로 한다.

1. **프로그램 설명**
   1. 프로그램 흐름도

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **모듈 정의**
   1. **Print\_File(char \*arr)**
      1. 사용 변수

char filename[100]: arr로부터 파일의 이름 부분을 잘라내 저장

char \*token: arr을 자를 때 사용

FILE \*ofp: 파일 입출력을 위해 이용

char get: 파일의 내용을 한글자씩 읽어와 저장

* + 1. 모듈 기능

현재 디렉터리에 해당 파일이 존재한다면 한 글자씩 읽어서 화면에 출력하고 1을 반환해준다.

해당 파일이 존재하지 않으면 ERROR메세지를 출력하고 0을 반환해 준다.

* 1. **Reset\_SymbolTable()**
     1. 사용 변수

symbol \*symboldel: symbolhead[]를 모두 초기화 해주기 위해 node를 따라갈때 사용

* + 1. 모듈 기능

Symbol table을 모두 초기화하기 위해 symbolhead[]의 노드를 모두 탐색하여 free해준다.

* 1. **Assemble\_File(char \*arr)**
     1. 사용 변수

char filename[100]: arr에서 잘라낸 파일이름을 저장하기 위해 사용

char\* token = NULL: arr을 자르기 위해 사용

int sw: Pass1이나 Pass2 실행 중 ERROR 발생 여부를 확인하기 위해 사용

* + 1. 모듈 기능

Assemble 기능을 실행하기위해 obj\_len[]을 초기화 해주고 Pass1함수와 Pass2함수를 호출하여 .lst파일과 .obj파일을 생성해준다. Assemble 도중에 에러를 발견한 경우엔 0을 반환하고 성공적으로 assemble된 경우엔 1을 반환한다.

* 1. **Pass1(char\*filename)**
     1. 사용 변수

FILE\* ofpr,\*ofpw: filename에 해당하는 파일과 loc.txt파일을 각각 읽기와 쓰기 모드로 열어 사용하기 위해 사용

char str[100]: filename에 해당하는 파일을 한줄씩 읽어오기 위해 사용

char mnemonic[100]: 명령어를 저장하기 위해 사용.

char sym[100]: symbol이 있는 경우 그를 저장하기 위해 사용

char temp[50]: 중간에 사용하는 string저장 하기 위해 사용

char \*token: strtok로 string자르는데 사용

int line: line을 계산해 저장

int location: location counter 계산해 저장

int len: str의 길이 저장

int form: format을 저장

hash \*opcode: Find\_Opcode함수를 호출해 해당하는 노드 저장

int operand: operand 저장

int obj\_cnt: obj코드의 한줄에 출력할 글자수 계산 위해 사용

int obj\_line: obj코드의 한줄에 출력할 글자수 계산 위해 사용

symbol \*use: Check\_Duplicate함수를 호출하여 해당하는 노드 저장

* + 1. 모듈 기능

Filename에 해당하는 파일을 열어 한 줄씩 읽어와 symbol, 명령어, operand를 각각 저장해 각각의 명령어에 맞는 format을 받아오고 변수나 상수의 경우 그에 맞는 숫자를 이용해 location counter와 line을 계산해 loc.txt에 한 줄씩 저장해준다.

Assemble 과정에 에러가 있는 경우 0을 반환하고 없는 경우 1을 반환해준다.

* 1. **Pass2(char\*filename)**
     1. 사용 변수

FILE \*ofp, \*loc, \*ofpl, \*ofpj: 파일입출력을 위해 사용

char str[100]: .asm파일의 내용을 한 줄씩 저장

char sym[100]: symbol저장

char opcode[100]: 명령어 저장

char operand[10]: operand 저장

char temp[100]: format4일때 명령어에서 +뗀 부분 저장

int modi[100]: modification을 위해 그 위치의 location 저장

char \*token: strtok하는데 사용

int line, location: loc.txt에서 line과 location받아와 저장

int len: 필욯한 string의 길이 저장

int imme: immediate addressing의 경우 숫자 저장

int comma: operand의 콤마 개수 저장

int flag: 에러를 체크하는데 사용

int obj: object code 저장

int form: format 저장

int pc: program counter 저장

int cnt: modification 필요한 갯수 저장

int obj\_cnt: obj\_len[]에 저장한 대로 .obj파일 만들기 위해 사용

int obj\_line: obj\_len[]에 저장한 대로 .obj파일 만들기 위해 사용

hash \*now: Find\_Opcode함수 실행해 노드 저장

symbol \*use: Check\_Duplicate함수 실행해 노드 저장

* + 1. 모듈 기능

.asm파일과 loc.txt파일을 읽어 와 .obj파일과 .lst파일 생성한다.

중간에 에러가 발생하는 경우 0을 반환하고 없는 경우엔 1을 반환한다.

* 1. **Make\_SymbolTable(char \*arr, int addr)**
     1. 사용 변수

symbol \*new: 링크드 리스트의 새로운 내용을 추가해주기 위해 사용

symbol \*temp: 링크드리스트로 구현된 symboltable의 탐색에 사용

symbol \*prev: 현재 노드의 전 노드를 저장

symbol \*sw: 해당 symbol이 이미 symbol table에 저장되어 있는지 확인

int hash: symbol의 첫글자 알파벳에 해당하는 숫자 저장(0~25)

* + 1. 모듈 기능

Check\_Duplicate 함수를 호출하여 Symbol이 중복되지 않는 경우 링크드리스트로 구현된 symbol table에 사전적 순서에 맞도록 저장해준다.

* 1. **Check\_Duplicate(char \*arr)**
     1. 사용 변수

symbol \*check: 링크드리스트로 구현된 symbol table을 탐색하는데 사용

* + 1. 모듈 기능

arr에 해당하는 symbol이 이미 symbol table에 저장된 symbol인지 확인하여 이미 저장되어 있는 경우 그 노드를 반환해 주고, 아닌 경우엔 NULL을 반환해준다.

* 1. **Find\_Opcode(char \*arr)**
     1. 사용 변수

hash \*temp: hash table이 저장된 링크드리스트를 탐색하는데 사용

int num: arr에 해당하는 명령어가 저장된 링크드리스트 헤드의 hash를 저장

* + 1. 모듈 기능

Hash\_Function을 호출해 arr에 해당하는 hash를 반환받아 그 링크드리스트를 탐색하여 해당 명령어가 opcode table에 존재하는 경우 그 노드를 반환해준다. 존재하지 않는 경우에는 NULL을 반환해 준다.

* 1. **Print\_SymbolTable()**
     1. 사용 변수

symbol \*temp: 링크드리스트로 구현된 symbol table을 탐색하는데 사용

* + 1. 모듈 기능

링크드리스트로 구현되어 있는 symbol table을 탐색하며 출력해준다.

* 1. **Format4(int opcode, int reg, int addr)**
     1. 사용변수

int obj: object code 저장

* + 1. 모듈 기능

Format4의 경우에 해당하는 object code생성해 반환해준다.

* 1. **Format3(int opcode, int reg, int addr)**
     1. 사용변수

int obj: object code 저장

* + 1. 모듈 기능

Format4의 경우에 해당하는 object code생성해 반환해준다.

* 1. **Format2(int opcode, char reg1, char reg2)**
     1. 사용변수

int obj: object code 저장

int r1, r2 : reg1과 reg2에 해당하는 레지스터에 맞게 number저장.

* + 1. 모듈 기능

Format2의 경우에 해당하는 object code생성해 반환해준다.

* 1. **PasteSymbol(int sw)**
     1. 사용 변수

symbol \*temp: 복사할 링크드 리스트를 탐색하기 위해 사용

symbol \*new: 복사될 링크드리스트를 생성하기 위해 사용

symbol \*paste: 복사될 링크드리스트를 탐색하기 위해 사용

* + 1. 모듈 기능

sw가 0인 경우와 1인 경우를 나눠 1인경우엔 assemble이 성공적으로 완료된 경우이고 sw가 0인 경우에는 assemble 도중 ERROR가 발생한 경우이다.

sw가 1인 경우에는 symbolkeep[]을 초기화해준 뒤 제대로 assemble되어 symbolhead[]에 저장된 symbol table의 내용을 symbolkeep[]에 복사해 둔다.

sw가 0인 경우엔 가장 최근에 성공적으로 assemble된 경우의 symbol table이 저장되어 있는 symbolkeep[]의 내용을 symbolhead[]에 복사해 준다.

1. **전역 변수 정의**
   1. **symbol \*symbolhead[26]**

symbol table을 링크드리스트로 저장하기 위해 사용

* 1. **symbol \*symbolkeep[26]**

assemble에 성공한 symbol table을 복사해두기 위해 사용

* 1. **int obj\_len[100]**

.obj 파일의 한 줄에 출력할 내용의 길이를 저장하기 위해 사용

* 1. **int program\_len**

프로그램의 전체 길이를 저장하기 위해 사용

* 1. **int base\_addr**

base의 위치를 저장하기 위해 사용

1. **코드 설명**
   1. **20171653.h**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<dirent.h>

#include<sys/stat.h>

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

* 1. **20171653.c**

#include "20171653.h"

unsigned char memory[16][65536];

history \*head;

hash \*hashhead[20];

symbol \*symbolhead[26];

symbol \*symbolkeep[26];

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

int dumpend=-1;

int dumpstart;

int program\_len;

int base\_addr;

/\*

main함수\*/

int main(void){

Read\_Opcode();

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");

}

/\*

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);

}

}

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);

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);

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);

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);

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);

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;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag= 0;

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;

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;

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;

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;

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;

break;

}

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

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;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag =0;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

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;

break;

}

}

else{

//printf("a\n");

printf("INVALID OPERAND!\n");

flag =0;

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;

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;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

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;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

break;

}

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

break;

}

}

else{

printf("INVALID OPERAND!\n");

flag = 0;

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();

}

/\*

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};

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

int i, comma=0;

int len;

char \*temp=NULL;

int cnt=0;

int sw;

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 저장우

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("ASSEMBLE ERROR!\n");

}

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{

printf("command ERROR!\n");

}

}