과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 3>>

**서강대학교 컴퓨터공학과**

**[20151597]**

**[임준묵]**

목 차

1. 프로그램 개요 4

2. 프로그램 설명 4

2.1 프로그램 흐름도 4

3. 모듈 정의 5

3.1 모듈 이름 : void save\_progaddr(char \*input, int \*progaddr) 5

3.1.1 기능 5

3.1.2 사용 변수 5

3.2 모듈 이름: ext\_symbol \*create\_ext\_symbol\_node(char \*symbol, int address, int Length) 5

3.2.1 기능 5

3.2.2 사용 변수 5

3.3 모듈이름: void make\_estab(ext\_symbol \*\*header, char \*symbol, int address, int Length) 5

3.3.1 기능 5

3.3.2 사용변수 5

3.4 모듈이름: ext\_symbol \*search\_estab(ext\_symbol \*\*header, char \*symbol) 5

3.4.1 기능 5

3.4.2 사용변수 5

3.5 모듈이름: print\_estab(ext\_symbol \*\*header) 6

3.5.1 기능 6

3.5.2 사용변수 6

3.6 모듈 이름 : int linking\_loader\_pass1(char \*filename, ext\_symbol \*\*header, int \*csaddr) 6

3.6.1 기능 6

3.6.2 사용변수 6

3.7 모듈 이름 : int linking\_loader\_pass2(unsigned char \*\*memory, char \*filename, ext\_symbol \*\*header, int \*csaddr) 7

3.7.1 기능 7

3.7.2 사용 변수 7

3.8 모듈 이름 : store\_operation(unsigned char \*\*memory, int ta, int value) 7

3.8.1 기능 7

3.8.2 사용 변수 7

3.9 모듈 이름 : int decision\_xbp(unsigned char \*\*memory, int current\_loc) 7

3.9.1 기능 7

3.9.2 사용변수 7

3.10 모듈 이름 : Int get\_target\_addr(unsigned char \*\*memory, int format, int pc, int reg[10]) 8

3.10.1 기능 8

3.10.2 사용변수 8

3.11 모듈 이름 : int bp\_clear\_or\_set(char \*input) 8

3.11.1 기능 8

3.11.2 사용 변수 8

3.12 모듈 이름 : void bp\_set(int \*bp, char \*input, int \*bp\_count) 8

3.12.1 기능 8

3.12.2 사용 변수 8

3.13 모듈 이름 : void bp\_print(int \*bp, int \*bp\_count) 8

3.13.1 기능 8

3.13.2 사용 변수 9

3.14 모듈 이름 : void bp\_clear(int \*bp, int \*bp\_count) 9

3.14.1 기능 9

3.14.2 사용 변수 9

3.15 모듈 이름 : int run(unsigned char \*\*memory, int progaddr, int endaddr, int reg[10], int \*bp, int\* bp\_count, int \*first\_run\_flag) 9

3.15.1 기능 9

3.15.2 사용 변수 9

3.16 모듈 이름 : int decision\_opcode(unsigned char \*\*memory, int current\_loc, int ni) 9

3.16.1 기능 9

3.16.2 사용 변수 9

3.17 모듈 이름 : int decision\_format(unsigned char \*\*memory, int current\_loc) 9

3.17.1 사용 변수 9

3.17.2 기능 10

3.18 모듈 이름 : int decision\_ni(unsigned char \*\*memory, int current\_loc) 10

3.18.1 기능 10

3.18.2 사용 변수 10

3.19 모듈 이름 : free\_ext\_symbol\_node(ext\_symbol \*\*head) 10

3.19.1 기능 10

3.19.2 사용 변수 10

4. 전역 변수 정의 10

5. 20151597.h 10

5.1 Enum registers{A=0, X, L, B, S, T, F, P, PC, SW}; 10

5.1.1 기능 10

5.2 Typedef struct \_ext\_symbol{ 10

5.2.1 기능 10

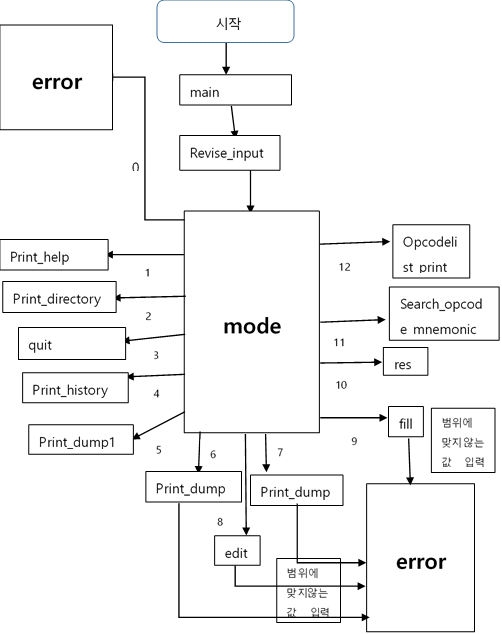
6. 코드 11

# 프로그램 개요

Object file들을 읽어 들여 memory에 load 해주고 실행을 시켜준다.

# 프로그램 설명

## 프로그램 흐름도



19

bp

18

run

17

Linking\_loader

16

Save\_progaddr

type

13

assemble

15

symbol

15

파일이 없는 경우

.asm 파일이 아닌 경우

# 모듈 정의

## 모듈 이름 : void save\_progaddr(char \*input, int \*progaddr)

### 기능

프로그램의 시작 주소를 받아서 저장한다.

### 사용 변수

Char \*input : 사용자가 입력한 input을 나타낸다.

Int \*progaddr : 사용자가 입력한 input에서 시작 주소를 뽑아낸다.

Char \*token1, token2 : 사용자가 입력한 input을 tokenize 하기 위해서 선언한 변수

## 모듈 이름: ext\_symbol \*create\_ext\_symbol\_node(char \*symbol, int address, int Length)

### 기능

External symbol table의 node를 만들어준다.

### 사용 변수

Char \*symbol : object file에서 읽은 symbol을 나타낸다.

Int address : symbol의 주소를 나타낸다.

Int length : program의 시작 주소를 나타낸다.

Ext\_symbol \*tmp : external symbol node를 의미한다.

## 모듈이름: void make\_estab(ext\_symbol \*\*header, char \*symbol, int address, int Length)

### 기능

External symbol table을 만들어준다. External symbol table은 hash table이다.

### 사용변수

Ext\_symbol \*\*header : external symbol tab의 포인터를 나타낸다. 크기가 20인 배열이다.

Char \*symbol : object file에서 읽은 symbol을 나타낸다.

Int address : symbol의 주소를 나타낸다.

Int length : program의 시작 주소를 나타낸다.

Int index : external symbol table의 포인터의 배열의 index를 저장한다.

Ext\_symbol \*newnode : external symbol table에 추가해줄 노드를 의미한다.

Ext\_symbol \*pointer : external symbol table의 노드에 접근하기 위해서 선언한 포인터 이다.

Char \*ssymbol : 공백을 제거한 symbol을 저장한다.

## 모듈이름: ext\_symbol \*search\_estab(ext\_symbol \*\*header, char \*symbol)

### 기능

External symbol table에서 해당 symbol의 노드를 반환한다.

### 사용변수

Ext\_symbol \*\*header : external symbol tab의 포인터를 나타낸다. 크기가 20인 배열이다.

Char \*symbol : object file에서 읽은 symbol을 나타낸다.

Int hash\_key : 해시 테이블의 키를 의미한다.

Ext\_symbol \*ptr : external symbol table에서 노드에 접근하기 위해서 선언한 포인터이다.

Ext\_symbol \*tmp : 반환해줄 노드 이다.

## 모듈이름: print\_estab(ext\_symbol \*\*header)

### 기능

External symbol table을 출력해주는 함수이다. 시작 주소를 기준으로 오름차순으로 정렬하여 출력해준다.

### 사용변수

Ext\_symbol \*\*header : external symbol tab의 포인터를 나타낸다. 크기가 20인 배열이다.

Int I : external symbol table의 pointer에 접근하기 위해서 선언한 변수이다.

Int total\_length : program의 전체 길이

Ext\_symbol \*pointer : external symbol table의 노드에 접근하기 위해서 선언한 포인터 이다.

Ext\_symbol \*new\_list : external symbol table의 노드들을 시작 주소를 기준으로 오름차순으로 정렬해서 연결리스트를 만들었을 때 그 연결리스트를 가리키는 포인터 이다.

Ext\_symbol \*newnode : external symbol table에서 찾아낸 node를 저장한다.

Ext\_symbol \*cur\_ptr : new\_list의 현재 node를 가리키기 위해서 선언한 포인터 이다.

Ext\_symbol \*prev : new\_list의 이전 node를 가리키기 위해서 선언한 포인터이다.

## 모듈 이름 : int linking\_loader\_pass1(char \*filename, ext\_symbol \*\*header, int \*csaddr)

### 기능

Object file을 읽어서 시작 주소를 기준으로 external symbol의 주소를 계산하고 estab에 external symbol과 그것의 주소, 그것의 길이를 저장한다.

### 사용변수

Char \*filename : object file의 이름을 나타낸다.

Ext\_symbol \*\*header : external symbol table의 포인터를 나타낸다.

Int \*csaddr : control section의 시작 주소를 나타낸다.

Char line[256] : object file에서 읽어 들이는 줄을 의미한다.

FILE \*fp : 파일 포인터

Int flag : pass1이 제대로 실행 되었는지를 저장하기 위해서 선언한 변수이다. 제대로 실행되었다면 flag=0 이고 flag=1이다.

Int cs\_length : control section의 길이를 저장하기 위해서 선언한 변수이다.

Int sym\_addr : symbol의 주소를 저장하기 위해서 선언한 변수이다.

Int I : 반복문을 사용하기 위해서 선언한 변수 이다.

## 모듈 이름 : int linking\_loader\_pass2(unsigned char \*\*memory, char \*filename, ext\_symbol \*\*header, int \*csaddr)

### 기능

Object file을 memory에 load 해준다.

### 사용 변수

Unsigned char \*\*memory : memory를 의미한다.

Char \*filename : object file의 이름을 나타낸다.

Ext\_symbol \*\*header : external symbol table의 포인터를 나타낸다.

Int \*csaddr : control section의 시작 주소를 나타낸다.

Char line[256] : object file에서 읽어 들이는 줄을 의미한다.

FILE \*fp : 파일 포인터

Char symbol[6] : object file 에서 읽어들인 symbol을 저장한다.

Char iindex[2] : object file 에서 읽어 들인 symbol들의 reference number을 저장한다.

Char \*ssymbol : symbol의 공백을 제거하여서 저장한다.

Char Addr[6], Length[2], objcode[2] : T record 에서 시작 위치와 길이, object code를 받기 위해서 선언한 변수 이다.

Int flag : pass2이 제대로 실행 되었는지를 저장하기 위해서 선언한 변수이다. 제대로 실행되었다면 flag=0 이고 flag=1이다.

Int addr, len, index, I : 주소와 길이, reference number을 정수로 저장하기 위해서 선언한 변수 이다.

Int x, y : memory의 행과 열의 index를 나타내기 위해서 선언한 변수 이다.

Int value, j : T record 에서 object code를 2roTlr 읽기 위해서 선언한 변수 이다.

Int length : T record에서 object code들의 길이를 저장하기 위해서 선언한 변수 이다.

Int sum, shift : m record에서 memory에 있는 object code들의 합을 구하기 위해서 선언한 변수 이다.

Ext\_symbol \*node : external symbol table의 노드들을 가리키는 변수 이다.

Int ext\_reference[10] : external symbol의 주소를 저장하기 위해서 선언한 배열 이다.

## 모듈 이름 : store\_operation(unsigned char \*\*memory, int ta, int value)

### 기능

Memory에서 target address로 이동하여서 value를 저장한다.

### 사용 변수

Unsigned char \*\*memory : memory를 의미한다.

Int ta : target address를 나타낸다.

Int value : memory에 저장 해야할 값을 나타낸다.

## 모듈 이름 : int decision\_xbp(unsigned char \*\*memory, int current\_loc)

### 기능

Memory에 저장되어 있는 object code의 xbp를 반환해준다.

### 사용변수

Unsigned char \*\*memory : memory를 의미한다

Int current\_loc : 접근 해야할 memory의 위치를 나타낸다.

Int k : current\_loc를 저장한다.

Int x, y : memory의 행과 열 index를 저장한다.

Int xbp : xbp를 저장한다.

Int content : 해당 memory의 값을 저장한다.

## 모듈 이름 : Int get\_target\_addr(unsigned char \*\*memory, int format, int pc, int reg[10])

### 기능

Memory에 저장되어 있는 object code들의 target address를 반환해준다.

### 사용변수

Unsigned char \*\*memory : memory를 의미한다

Int format : memory에 저장되어 있는 object code들의 format을 나타낸다.

Int pc : memory에서 현재 위치를 나타낸다.

Int reg[10] : 레지스터를 의미한다.

Int disp : displacement 를 의미한다.

Int ta : target address를 저장하기 위해서 선언한 변수 이다.

Int sign\_flag : displacement가 음수인 경우 1을 저장하고 아닌 경우에는 0을 저장한다.

Int xbp : object code의 xbp를 저장한다.

## 모듈 이름 : int bp\_clear\_or\_set(char \*input)

### 기능

사용자의 입력이 bp를 set하는지 bp를 출력해주는지 bp를 clear 해주는지를 결정해준다.

### 사용 변수

Char \*input : 사용자의 input을 의미한다.

Char \*token1, \*token2 : 사용자의 input을 tokenize하기 위해서 선언한 변수이다.

Int flag : bp set인지 bp clear인지 bp print인지를 저장하는 변수 이다.

## 모듈 이름 : void bp\_set(int \*bp, char \*input, int \*bp\_count)

### 기능

사용자가 입력한 Break point를 저장해준다.

### 사용 변수

Int \*bp : break point들을 저장하기 위해서 선언한 변수 이다.

Char \*input : 사용자의 input을 의미한다.

Int \*bp\_count : 사용자가 입력한 break point의 개수를 나타낸다.

Char \*token1, \*token2 : 사용자의 input을 tokenize하기 위해서 선언한 변수이다.

Int addr : 사용자가 입력한 break point를 나타낸다.

## 모듈 이름 : void bp\_print(int \*bp, int \*bp\_count)

### 기능

Break point를 출력해준다.

### 사용 변수

Int \*bp : 사용자가 입력한 break point를 저장한다.

Int \*bp\_count : 사용자가 입력한 break point의 개수를 나타낸다.

Int I : 반복문을 사용하기 위해서 선언한 변수 이다.

## 모듈 이름 : void bp\_clear(int \*bp, int \*bp\_count)

### 기능

Break point를 clear 해준다.

### 사용 변수

Int \*bp : 사용자가 입력한 break point를 저장한다.

Int \*bp\_count : 사용자가 입력한 break point의 개수를 나타낸다

Int I : 반복문을 사용하기 위해서 선언한 변수 이다.

## 모듈 이름 : int run(unsigned char \*\*memory, int progaddr, int endaddr, int reg[10], int \*bp, int\* bp\_count, int \*first\_run\_flag)

### 기능

Memory에 load한 프로그램을 실행시킨다.

### 사용 변수

Unsigned char \*\*memory : 메모리를 나타낸다.

Int progaddr : program의 시작 주소를 나타낸다.

Int endaddr : program의 끝나는 주소를 나타낸다.

Int reg[10] : 레지스터를 나타낸다.

Int \*bp : break point를 저장하는 배열을 나타낸다.

Int \*bp\_count : break point의 개수를 나타낸다.

Int \*first\_run\_flag : program을 첫번째로 실행하는지를 나타낸다.

## 모듈 이름 : int decision\_opcode(unsigned char \*\*memory, int current\_loc, int ni)

### 기능

Memory에 load 되어 있는 object code의 Opcode를 결정해준다.

### 사용 변수

Unsigned char \*\*memory : 메모리를 나타낸다.

Int current\_loc : memory에서 현재 위치를 나타낸다.

Int ni : object code의 ni를 나타낸다.

## 모듈 이름 : int decision\_format(unsigned char \*\*memory, int current\_loc)

### 사용 변수

Unsigned char \*\*memory : 메모리를 나타낸다.

Int current\_loc : memory에서 현재 위치를 나타낸다.

### 기능

Memory에 load 되어 있는 object code의 format을 결정해준다.

## 모듈 이름 : int decision\_ni(unsigned char \*\*memory, int current\_loc)

### 기능

Memory에 load 되어 있는 object code의 ni를 결정해준다.

### 사용 변수

Unsigned char \*\*memory : 메모리를 나타낸다.

Int current\_loc : memory에서 현재 위치를 나타낸다.

## 모듈 이름 : free\_ext\_symbol\_node(ext\_symbol \*\*head)

### 기능

External symbol table을 free 해준다.

### 사용 변수

Ext\_symbol \*\*head : external symbol table를 가리키는 포인터를 나타낸다.

# 전역 변수 정의

전역 변수는 선언하지 않았다.

# 20151597.h

## Enum registers{A=0, X, L, B, S, T, F, P, PC, SW};

### 기능

레지스터의 값을 배열에 저장하였는데 배열의 index로 사용하기 위해서 선언하였다.

## Typedef struct \_ext\_symbol{

Char name[20];

Int length;

Int addr;

Struct \_ext\_symbol \*link;

} ext\_symbol;

### 기능

External symbol table을 만들기 위해서 선언하였다.

# 코드

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "20151597.h"

//for print help

void print\_help(){

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

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

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

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

printf("opcodelist \n");

printf("assemble filename\n");

printf("type filename\n");

printf("symbol\n");

printf("progaddr\n");

printf("load\n");

printf("run\n");

printf("bp\n");

printf("clear\n");

}

void type\_filename(char \*input){

char \*file\_name, \*buffer;

char \*line;

FILE \*fp;

char c;

DIR \*dir;

dir = opendir(input);

if(dir)

printf("this is directory");

// file\_name[strlen(file\_name)-1] = '\0';

fp = fopen(input, "r");

// printf("file name : %s\n", input);

if(!fp){

printf("file open error!\n");

return ;

}

else{

while(1){

// fgets(line, sizeof(line), fp);

// if(line)

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

c = fgetc(fp);

if(c==EOF)

break;

else

printf("%c", c);

}

}

printf("\n");

fclose(fp);

}

//string convert to decimal

int convert\_dec(char \*input){

int i, len, result;

len = strlen(input);

result = 0;

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

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

result = 16\*result+(input[i]-'0');

}

else if(input[i]>='A' && input[i]<= 'F'){

result = 16\*result+(input[i]-'A'+10);

}

else if(input[i]>='a' && input[i]<= 'f')

result = 16\*result+(input[i]-'a'+10);

else if(input[i]=='x')

result = 16 \* result + (input[i] - 'x');

else if(input[i]=='X')

result = 16 \* result + (input[i] - 'X');

else

result = -1;

}

return result;

}

//memory array 0

void reset(unsigned char \*\*memory){

int i, j;

for(i = 0; i<65536; i++){

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

memory[i][j] = 0;

}

}

}

//dump1(no start address, no end address)

void print\_dump1(unsigned char \*\*memory, int \*start){

int start1, end1, i, j;

int first\_column;

int start\_index, end\_index;

int start\_index2, end\_index2;

start1 = \*start;

first\_column = 0;

end1 = start1 +159;

start\_index = start1/16;

start\_index2 = start1%16;

first\_column = start\_index\*16;

if(end1>1048575){

end1 = 1048575;

end\_index = end1/16;

end\_index2 = end1%16;

}

else{

end\_index = end1/16;

end\_index2 = end1%16;

}

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

printf("%05X\t", first\_column);

first\_column = first\_column+16;

if(i == start\_index){

for(j = 0; j<start\_index2; j++)

printf("\t");

for(j = start\_index2; j<16; j++){

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

}

printf(";");

for (j = 0; j < start\_index2; j++)

printf(".");

for(j = start\_index2; j<16; j++){

if(memory[i][j]>=32&&memory[i][j]<=126)

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

else

printf(".");

}

printf("\n");

}

else if(i==end\_index){

for(j = 0; j<=end\_index2; j++){

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

}

for(j = end\_index2+1; j<16; j++)

printf("\t");

printf(";");

for(j = 0; j<=end\_index2; j++){

if(memory[i][j]>=32&&memory[i][j]<=126)

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

else

printf(".");

}

for (j = end\_index2 + 1; j<16;j++)

printf(".");

printf("\n");

}

else{

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

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

printf(";");

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

if(memory[i][j]>=32&&memory[i][j]<=126)

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

else

printf(".");

}

printf("\n");

}

}

if(end1 == 1048575){

\*start = 0;

}

else{

\*start = end1+1;

}

}

//dump2(start address, no end address)

int print\_dump2(unsigned char \*\*memory, char \*input, int \*start){

int start1, end1, i, j;

int first\_column;

int start\_index, end\_index;

int start\_index2, end\_index2;

char k[100];

char \*start\_point;

char \*buffer;

int flag = 0;

start\_point = strtok(input, "\_");

start\_point = strtok(NULL, "");

start1 = convert\_dec(start\_point);

if(start1<0||start1>1048575){

printf("dump error : start input range over\n");

flag = -1;

}

else {

end1 = start1 + 159;

start\_index = start1 / 16;

start\_index2 = start1 % 16;

first\_column = start\_index \* 16;

if (end1 > 1048575) {

end1 = 1048575;

end\_index = end1 / 16;

end\_index2 = end1 % 16;

}

else {

end\_index = end1 / 16;

end\_index2 = end1 % 16;

}

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

printf("%05X\t", first\_column);

first\_column = first\_column + 16;

if (i == start\_index) {

for(j = 0; j<start\_index2; j++){

printf("\t");

}

for (j = start\_index2; j < 16; j++) {

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

}

printf(";");

for(j = 0; j<start\_index2; j++){

printf(".");

}

for (j = start\_index2; j < 16; j++) {

if (memory[i][j] >= 32 && memory[i][j] <= 126) {

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

}

else

printf(".");

}

printf("\n");

}

else if (i == end\_index) {

for (j = 0; j <= end\_index2; j++) {

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

}

for(j=end\_index2+1; j<16; j++)

printf("\t");

printf(";");

for(j = end\_index2+1;j<16; j++)

printf(" ");

for (j = 0; j <= end\_index2; j++) {

if (memory[i][j] >= 32 && memory[i][j] <= 126) {

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

}

else

printf(".");

}

printf("\n");

}

else {

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

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

}

printf(";");

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

if (memory[i][j] >= 32 && memory[i][j] <= 126) {

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

}

else

printf(".");

}

printf("\n");

}

}

if (end1 == 1048575)

\*start = 0;

else

\*start = end1 + 1;

}

return flag;

}

//dump3(start address, end address)

int print\_dump3(unsigned char \*\*memory, char \*input, int \*start) {

int start1, end1, i, j;

int first\_column;

int start\_index, start\_index2;

int end\_index, end\_index2;

int flag;

char \*start\_point, \*end\_point, \*temp;

flag = 0;

temp = strtok(input, "\_");

start\_point = strtok(NULL, ",");

end\_point = strtok(NULL, "");

start1 = convert\_dec(start\_point);

end1 = convert\_dec(end\_point);

if (start1 < 0 || start1>1048575) {

printf("dump error : start input range over\n");

flag = -1;

}

if (end1 < 0 || end1>1048575) {

printf("dump error : end input range over\n");

flag = -1;

}

if (start1 > end1) {

printf("dump error : start input larger than end input\n");

flag = -1;

}

if (flag == 0) {

start\_index = start1 / 16;

start\_index2 = start1 % 16;

first\_column = start\_index \* 16;

end\_index = end1 / 16;

end\_index2 = end1 % 16;

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

printf("%05X\t", first\_column);

first\_column = first\_column + 16;

if (i == start\_index && i == end\_index) {

for (j = 0; j < start\_index2; j++)

printf("\t");

for (j = start\_index2; j <= end\_index2; j++)

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

for (j = end\_index2 + 1; j < 16; j++)

printf("\t");

printf(";");

for (j = 0; j < start\_index2; j++)

printf(".");

for (j = start\_index2; j <= end\_index2; j++) {

if (memory[i][j] >= 32 && memory[i][j] <= 126)

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

else

printf(".");

}

for (j = start\_index2 + 1; j < 16; j++)

printf(".");

printf("\n");

}

else if (i == start\_index) {

for (j = 0; j < start\_index2; j++)

printf("\t");

for (j = start\_index2; j < 16; j++)

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

printf(";");

for (j = 0; j < start\_index2; j++)

printf(".");

for (j = start\_index2; j < 16; j++) {

if (memory[i][j] >= 32 && memory[i][j] <= 126)

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

else

printf(".");

}

printf("\n");

}

else if (i == end\_index) {

for (j = 0; j <= end\_index2; j++)

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

for (j = end\_index2 + 1; j < 16; j++)

printf("\t");

printf(";");

for (j = 0; j <= end\_index2; j++) {

if (memory[i][j] >= 32 && memory[i][j] <= 126)

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

else

printf(".");

}

for (j = end\_index2 + 1; j < 16; j++)

printf(".");

printf("\n");

}

else {

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

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

printf(";");

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

if (memory[i][j] >= 32 && memory[i][j] <= 126)

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

else

printf(".");

}

printf("\n");

}

}

if (end1 < 1048575)

\*start = end1 + 1;

else if (end1 == 1048575)

\*start = 0;

}

return flag;

}

//target address's value -> value of user input

int edit(unsigned char \*\*memory, char \*input) {

char \*value\_, \*target\_point, \*temp;

int target, value;

int target\_index, target\_index2;

int flag = 0;

temp = strtok(input, "\_");

target\_point = strtok(NULL, ",");

value\_ = strtok(NULL, "");

target = convert\_dec(target\_point);

value = convert\_dec(value\_);

if (target < 0 || target>1048575) {

printf("edit error : target point range over!\n");

flag = -1;

}

if (value > 255 || value < 0) {

printf("edit error : value range over!\n");

flag = -1;

}

if (flag == 0) {

target\_index = target / 16;

target\_index2 = target % 16;

memory[target\_index][target\_index2] = value;

}

return flag;

}

//from start address to end address, memory's value->value of user input

int fill(unsigned char \*\*memory, char \*input) {

char \*start\_point, \*end\_point, \*value\_, \*temp;

int start\_index, end\_index;

int start\_index2, end\_index2;

int value, start, end;

int i, j;

int flag = 0;

temp = strtok(input, "\_");

start\_point = strtok(NULL, ",");

end\_point = strtok(NULL, ",");

value\_ = strtok(NULL, "");

start = convert\_dec(start\_point);

end = convert\_dec(end\_point);

value = convert\_dec(value\_);

if (start < 0 || start>1048575) {

printf("fill error : start input range over!\n");

flag = -1;

}

if (end < 0 || end>104875) {

printf("fill error : end input range over!\n");

flag = -1;

}

if (value < 0 || value>255) {

printf("fill error : value input range over!\n");

flag = -1;

}

if(start>end){

printf("fill error : start input more than end input!\n");

flag = -1;

}

if (flag == 0) {

start\_index = start / 16;

start\_index2 = start % 16;

end\_index = end / 16;

end\_index2 = end % 16;

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

if (i == start\_index && i == end\_index) {

for (j = start\_index2; j <= end\_index2; j++) {

memory[i][j] = value;

}

}

else if (i == start\_index) {

for (j = start\_index2; j < 16; j++)

memory[i][j] = value;

}

else if (i == end\_index) {

for (j = 0; j <= end\_index2; j++)

memory[i][j] = value;

}

else {

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

memory[i][j] = value;

}

}

}

return flag;

}

//print file, directory in current directory

void print\_directory(){

DIR \*dir;

struct dirent \*dir\_entry;

struct stat astat;

dir = opendir(".");

while((dir\_entry = readdir(dir))!=NULL){

lstat(dir\_entry->d\_name, &astat);

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

if(strncmp(dir\_entry->d\_name, ".", 1)!=0 && strncmp(dir\_entry->d\_name, "..",2)!=0){

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

printf("/\n");

}

}

else{

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

printf("%s\*\n", dir\_entry->d\_name);

}

else

printf("%s\n", dir\_entry->d\_name);

}

}

}

//create history node for history linked list

history \*create\_history\_node(char \*input){

history \*tmp;

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

strcpy(tmp->name, input);

tmp->link = NULL;

return tmp;

}

//make history linked list

void history\_made(history \*\*header, history \*newnode){

history \*tmp = \*header;

if(header == NULL){

\*header = newnode;

}

else{

while(tmp->link != NULL){

tmp = tmp->link;

}

tmp->link = newnode;

tmp->link->index = tmp->index+1;

}

}

//print history linked list

void print\_history(history \*header){

history \*ptr = header;

while(ptr->link !=NULL){

printf("%d\t%s\n", ptr->link->index, ptr->link->name);

ptr = ptr->link;

}

}

//generate hash key for mnemonic

int hash\_function(char \*input) {

int result,i, len;

result = 0;

len = strlen(input);

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

result = (int)input[i]+result;

}

result = result%20;

return result;

}

//create hash node for hash table

hash \*create\_hash\_node(FILE \*fp) {

hash \*tmp;

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

fscanf(fp, "%s%s%s", tmp->opcode, tmp->instructions, tmp->ss);

tmp->link = NULL;

return tmp;

}

//make hash table

void hashtable\_made(hash \*header[20], FILE \*fp) {

hash \*pointer;

hash \*newnode;

int i, index;

for (i = 0; i < 58; i++) {

newnode = create\_hash\_node(fp);

index = hash\_function(newnode->instructions);

pointer = header[index];

if (header[index] == NULL)

header[index] = newnode;

else {

while (pointer->link != NULL)

pointer = pointer->link;

pointer->link = newnode;

}

}

}

hash \*return\_opcode\_node(hash \*\*header, char \*opcode){

int hash\_key;

hash \*ptr;

hash \*temp=NULL;

hash\_key = hash\_function(opcode);

if(hash\_key>=0 && hash\_key<20){

ptr = header[hash\_key];

while(ptr != NULL){

if(strcmp(ptr->instructions, opcode)==0){

temp = ptr;

break;

}

ptr = ptr->link;

}

}

return temp;

}

//print hash table

void opcodelist\_print(hash \*\*header) {

int i;

hash \*ptr;

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

ptr = header[i];

printf("%d : ", i);

if(ptr!=NULL){

printf("[%s, %s]", ptr->instructions, ptr->opcode);

ptr = ptr->link;

}

while (ptr != NULL) {

printf("->[%s, %s]", ptr->instructions, ptr->opcode);

ptr = ptr->link;

}

printf("\n");

}

}

//search mnemonic's opcode in hash table

int search\_opcode\_mnemonic(hash \*\*header, char \*input){

char \*opcode, \*mnemonic;

int hash\_key, flag;

int result = 0;

hash \*ptr;

flag = 0;

opcode = strtok(input, "\_");

mnemonic = strtok(NULL, "");

printf("input: %s\n", input);

hash\_key = hash\_function(mnemonic);

ptr = header[hash\_key];

while(ptr!=NULL){

if(strcmp(ptr->instructions, mnemonic)==0){

flag++;

break;

}

ptr = ptr->link;

}

if(flag==1)

printf("opcode is %s\n", ptr->opcode);

else {

printf("this mnemonic doesn't exist in hash table\n");

result = -1;

}

return result;

}

symbol \*create\_symbol\_node(char \*ssymbol, int addr){

symbol \*tmp;

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

strcpy(tmp->sym, ssymbol);

tmp->address = addr;

tmp->link = NULL;

return tmp;

}

symbol \*search\_symbol\_table(char \*ssymbol, symbol \*\*header){

int hash\_key, result;

symbol \*ptr;

symbol \*tmp=NULL;

hash\_key = ssymbol[0]-'A';

if(hash\_key>=0 && hash\_key<=25){

ptr = header[hash\_key];

while(ptr !=NULL){

if(strcmp(ptr->sym, ssymbol)==0){

tmp = ptr;

break;

}

ptr = ptr->link;

}

}

// printf("symbol search\n");

return tmp;

}

void print\_symbol\_table(symbol \*\*header){

int i;

symbol \*pointer;

for(i = 25; i>=0; i--){

pointer = header[i];

if(pointer != NULL){

while(pointer != NULL){

printf("%s %04X\n", pointer->sym, pointer->address);

pointer = pointer->link;

}

}

}

}

void make\_symbol\_table(symbol \*\*header, int addr, char \*ssymbol){

int index = 0;

int i;

symbol \*newnode;

symbol \*pointer;

symbol \*prev;

symbol \*ptr;

newnode = create\_symbol\_node(ssymbol, addr);

index = ssymbol[0] - 'A';

if(header[index] == NULL){

header[index] = newnode;

// printf("symbol table : %s\n", header[index]->sym);

}

else{

if(strcmp(newnode->sym, header[index]->sym)>0){

newnode->link = header[index];

header[index] = newnode;

}

else{

pointer = header[index];

while(pointer != NULL && strcmp(newnode->sym, pointer->sym)<0){

prev = pointer;

pointer = pointer->link;

}

newnode->link = prev->link;

prev->link = newnode;

}

}

}

int char\_dec(char \*input){

int i;

int result = 0;

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

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

result = result\*10 + input[i]-'0';

}

}

return result;

}

int format2(char \*opcode, char \*operand){

int sum, opcd;

int value\_reg1, value\_reg2;

char reg1[10] = {0};

char reg2[10] = {0};

value\_reg1 = -1;

value\_reg2 = -1;

opcd = convert\_dec(opcode);

strncpy(reg1, operand, 1);

strncpy(reg2, operand+2, 1);

if(reg2[0]==' ')

value\_reg2 = 0;

if(strcmp(reg1, "A")==0)

value\_reg1 = 0;

else if(strcmp(reg1, "X")==0)

value\_reg1 = 1;

else if(strcmp(reg1, "L")==0)

value\_reg1 = 2;

else if(strcmp(reg1, "PC")==0)

value\_reg1 = 8;

else if(strcmp(reg1, "SW")==0)

value\_reg1 = 9;

else if(strcmp(reg1, "B")==0)

value\_reg1 = 3;

else if(strcmp(reg1, "S")==0)

value\_reg1 = 4;

else if(strcmp(reg1, "T")==0)

value\_reg1 = 5;

else if(strcmp(reg1, "F")==0)

value\_reg1 = 6;

// printf("value reg1 : %d\n", value\_reg1);

//뒤에 아무것도 없는 경우 value\_reg2 = 0

if(strcmp(reg2, "A")==0)

value\_reg2 = 0;

else if(strcmp(reg2, "X")==0)

value\_reg2 = 1;

else if(strcmp(reg2, "L")==0)

value\_reg2 = 2;

else if(strcmp(reg2, "PC")==0)

value\_reg2 = 8;

else if(strcmp(reg2, "SW")==0)

value\_reg2 = 9;

else if(strcmp(reg2, "B")==0)

value\_reg2 = 3;

else if(strcmp(reg2, "S")==0)

value\_reg2 = 4;

else if(strcmp(reg2, "T")==0)

value\_reg2 = 5;

else if(strcmp(reg2, "F")==0)

value\_reg2 = 6;

// printf("reg1, reg2 : %s, %s\n", reg1, reg2);

if(value\_reg1<0 || value\_reg2<0)

sum = value\_reg1+value\_reg2;

else if(value\_reg1>=0 && value\_reg2>=0){

opcd = opcd<<8;

value\_reg1 = value\_reg1<<4;

sum = opcd+value\_reg1+value\_reg2;

}

// printf("reg1, reg2, sum : %s %s %d\n", reg1, reg2, sum);

return sum;

}

int format3(char \*opcode, int nixbpe, int disp){

int opcd, sum, temp, disp2;

opcd = 0;

sum = 0;

// printf("format3 opcode : %s\n", opcode);

opcd = convert\_dec(opcode);

//opcd2 = opcd&temp;

opcd = opcd<<16;

nixbpe = nixbpe<<12;

if(disp>=0)

sum = opcd+nixbpe+disp;

else{

disp2 = (0x000FFF & disp);

sum = opcd+nixbpe+disp2;

}

// printf(" opcode : %s disp : %04X\n", opcode, disp2);

return sum;

}

int format4(char \*opcode, int nixbpe, int disp){

int opcd, sum, disp2;

opcd = convert\_dec(opcode);

sum = 0;

opcd = opcd<<24;

nixbpe = nixbpe<<20;

if(disp>=0)

sum = opcd+nixbpe+disp;

else{

disp2 = (0x0FFFFF & disp);

sum = opcd+nixbpe+disp2;

}

}

int pass1(char \*file\_name, int \*length, int \*base\_address, symbol \*\*symbol\_header, hash \*\*hash\_header, int \*object\_code\_len){

FILE \*fp1, \*fp2;

char line[128], line2[128], line3[128];

char \*name1, \*name2;

// buffer = strtok(std\_input, "\_");

// file\_name = strtok(NULL, "");

char buffer[20];

char \*k, \*kk;

char \*label, \*Operator, \*operand, \*operand2, \*temp;

char base\_operand[10];

int addr = 0;

int start\_addr = 0;

int oidx = 0;

int enter\_flag = 0;

int end\_flag = 0;

int assemble\_error\_flag = 0;

int file\_error\_flag = 0;

strcpy(buffer, file\_name);

// printf("FIle name : %s", buffer);

name1 = strtok(file\_name, ".");

name2 = strtok(NULL, "");

// printf("name1, name2 : %s %s\n", name1, name2);

fp2 = fopen("intermediate.txt","w");

if(!name2){

file\_error\_flag = 1;

printf("not assembley file !\n");

return file\_error\_flag;

}

else if(strcmp(name2, "asm")!=0){

file\_error\_flag = 1;

printf("not assembly file!\n");

return file\_error\_flag;

}

if(file\_error\_flag == 0){

fp1 = fopen(buffer, "r");

if(fp1 == NULL){

file\_error\_flag = 1;

printf("file open error!\n");

return file\_error\_flag;

}

else{

fgets(line, sizeof(line), fp1);

line[strlen(line)-1] = '\0';

strcpy(line2, line);

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

label = " ";

Operator = strtok(line, " \t");

operand = strtok(NULL, "");

}

else{

label = strtok(line, " \t");

Operator = strtok(NULL, " \t");

operand = strtok(NULL, "");

}

// printf("line : %s\n", line);

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

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

start\_addr = strtol(operand, &temp,16);

addr = start\_addr;

// printf("start address : %04X, operadn : %s\n", start\_addr, operand);

// printf("start : %s %s %s \n", label, Operator, operand);

fprintf(fp2, "%04X %-7s %-7s %-10s\n", addr, label, Operator, operand);

}

else{

printf("%s error : don't have START\n", line2);

assemble\_error\_flag = 1;

return assemble\_error\_flag;

}

while(strstr(Operator, "END")==0){

fgets(line, sizeof(line), fp1);

if(feof(fp1))

break;

line[strlen(line)-1] = '\0';

if(line[0] != '.'){

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

label = " ";

Operator = strtok(line, " \t");

operand = strtok(NULL, " \t");

operand2 = strtok(NULL, "");

if(!operand){

operand = " ";

}

else{

if(operand2)

strcat(operand, operand2);

}

// printf("line1 : %s, %s, %s\n", label, Operator, operand);

}

else{

label = strtok(line, " \t");

Operator = strtok(NULL, " \t");

operand = strtok(NULL, " \t");

operand2 = strtok(NULL, "");

// printf("line1 : %s, %s, %s \n", label, Operator, operand);

if(!operand){

operand = " ";

}

else{

if(operand2)

strcat(operand, operand2);

}

// printf("line1 : %s, %s, %s \n", label, Operator, operand);

}

if(label[0] != ' '){

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

assemble\_error\_flag = 1;

printf("label front END\n");

return assemble\_error\_flag;

}

else if(!search\_symbol\_table(label, symbol\_header)){

make\_symbol\_table(symbol\_header, addr, label);

// printf("make sym table\n");

}

else{

printf("duplicate label in SYMTAB!\n");

assemble\_error\_flag = 1;

return assemble\_error\_flag;

}

}

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

fprintf(fp2, " %-7s %-7s %-10s\n", label, Operator, operand);

end\_flag = 1;

// printf("end \_flag : %d\n", end\_flag);

break;

}

else{

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

fprintf(fp2, " %-7s %-7s %-10s\n", label, Operator, operand);

}

else{

/\* if(label[0] = ' '){

fprintf(fp2, " %-7s %-7s %-10s\n", label, Operator, operand);

}\*/

fprintf(fp2, "%04X %-7s %-7s %-10s\n", addr, label, Operator, operand);

}

}

/\*

else{

//fgets(line2, sizeof(line2), fp1);

printf("line : %s\n", line3);

printf("line2: %s\n", line2);

if(!line2){

fprintf(fp2, " %-7s %-7s %-10s\n", label, Operator, operand);

break;

}

else{

fprintf(fp2, "%04X %-7s %-7s %-10s\n", addr, label, Operator, operand);

}

}\*/

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

k = strtok(Operator+1, "");

if(return\_opcode\_node(hash\_header, k)){

addr = addr+4;

if(object\_code\_len[oidx]+4>30 || enter\_flag == 1){

oidx++;

enter\_flag=0;

}

object\_code\_len[oidx] +=4;

}

else{

assemble\_error\_flag = 1;

return assemble\_error\_flag;

}

}

else if(Operator[0] != '+'){

if(return\_opcode\_node(hash\_header, Operator)){

if(strcmp(return\_opcode\_node(hash\_header, Operator)->ss, "2")==0){

addr = addr+2;

if(object\_code\_len[oidx]+2>30||enter\_flag==1){

oidx++;

enter\_flag=0;

}

object\_code\_len[oidx] += 2;

}

else{

addr = addr+3;

if(object\_code\_len[oidx]+3>30 || enter\_flag == 1){

oidx++;

enter\_flag = 0;

}

object\_code\_len[oidx] +=3;

}

}

else{

if(strcmp(Operator, "WORD")==0){

addr = addr+3;

if(object\_code\_len[oidx]+3>30 || enter\_flag==1){

oidx++;

enter\_flag=0;

}

object\_code\_len[oidx] +=3;

}

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

addr = addr+3\*char\_dec(operand);

enter\_flag = 1;

}

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

addr = addr+char\_dec(operand);

enter\_flag = 1;

}

else if(strcmp(Operator, "BYTE")==0){

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

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

addr = addr+strlen(k);

if(object\_code\_len[oidx]+strlen(k)>30 || enter\_flag == 1){

oidx++;

enter\_flag = 0;

}

object\_code\_len[oidx] += strlen(k);

}

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

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

addr = addr+strlen(k)/2;

if(object\_code\_len[oidx]+strlen(k)/2>30 || enter\_flag == 1){

oidx++;

enter\_flag = 0;

}

object\_code\_len[oidx] += strlen(k)/2;

}

else{

printf("invalid operand!\n");

assemble\_error\_flag = 1;

break;

}

}

else if(strcmp(Operator, "BASE")==0){

\*base\_address = 0;

strcpy(base\_operand, operand);

// printf("base\_operand : %s\n", base\_operand);

}

}

}

}

}

//strcpy(line3, line2);

}

\*length = addr-start\_addr;

\*base\_address = search\_symbol\_table(base\_operand, symbol\_header)->address;

// fprintf(fp2, " %-7s %-7s %-10s\n", label, Operator, operand);

if(end\_flag ==0){

printf("%s don't have END\n", line);

printf("assemble error!\n");

assemble\_error\_flag = 1;

}

// printf("base address : %04X\n", \*base\_address);

fclose(fp1);

fclose(fp2);

// return assemble\_error\_flag;

// printf("assemble flag : %d\n", assemble\_error\_flag);

}

return assemble\_error\_flag;

}

int pass2(char \*file\_name, int \*length, int \*base\_address, symbol \*\*symbol\_header, hash \*\*hash\_header, int \*object\_code\_len){

//fp1 = intermediate.txt, fp2 = file\_name.lst, fp3 = file\_name.obj

FILE \*fp1, \*fp2, \*fp3;

char \*write\_file\_name, lst\_file\_name[20], obj\_file\_name[20];

//line of intermediate file

char line[128], line2[128];

char buffer[128];//line을 저장하기 위해서

char label[10], operator[10], operand[10], address[10];//intermediate 파일에서 읽은 line을 이와같이 나눈다.

char \*rev\_label, \*rev\_operator, \*rev\_operand, \*rev\_address;//각각에 대해서 공백을 제거

char \*rev\_operand2;//X'F1', C'EOF'에서 F1과 EOF를 분리하기 위해서

int line\_num;//몇번째 줄인지 나타내기 위해서

int record\_length;

int object\_code;//object code

int object\_code\_byte;

int error\_flag;

int i;//반복문을 위한 index

int enter\_flag;//object file에서 개행을 해야할 때를 알려주는 flag

int pc;//program counter

int modicode[100], midx;//modification을 처리하기 위해서

int ta;//target address

int nixbpe;

int constant; // #뒤에 붙은 상수를 처리하기 위해

int first\_execution\_addr;//프로그램에서 첫번 째로 수행되는 instruction의 주소

symbol \*sym\_node;

hash \*hash\_node;

int oidx = 0;//object\_code\_len을 위한 index

midx = 0;

pc = 0;

object\_code\_byte = 0;

enter\_flag = 1;

object\_code\_byte=0;

line\_num = 5;

record\_length = 0;

object\_code = 0;

error\_flag = 0;

fp1 = fopen("intermediate.txt", "r");

write\_file\_name = strtok(file\_name, ".");

strcpy(lst\_file\_name, write\_file\_name);

strcpy(obj\_file\_name, write\_file\_name);

strcat(lst\_file\_name, ".lst");

strcat(obj\_file\_name, ".obj");

fp2 = fopen(lst\_file\_name, "w");

fp3 = fopen(obj\_file\_name, "w");

fgets(line, sizeof(line), fp1);

line[strlen(line)-1] = '\0';

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

strncpy(address, line, 5);

strncpy(label, line+5, 8);

label[6] = '\0';

strncpy(operator, line+13, 8);

operator[6] ='\0';

strncpy(operand, line+21, strlen(line)-21);

operand[strlen(operand)-1] = '\0';

rev\_address = strtok(address, " \t");

// printf("label : %s\n", label);

rev\_label = strtok(label, " \t");

// printf("rev\_label : %s\n", rev\_label);

if(!rev\_label){

rev\_label = " ";

// printf("rev label : %s\n", rev\_label);

}

rev\_operator = strtok(operator, " \t");

rev\_operand = strtok(operand, " \t");

// printf("%s\n",rev\_address);

// printf("%s\n", rev\_label);

// printf("%s\n", rev\_operator);

// printf("%s\n", rev\_operand);

if(strcmp(rev\_operator, "START")==0){

fprintf(fp2, "%3d %s\n", line\_num, line);

line\_num = line\_num+5;

if(rev\_label[0]==' '){

fprintf(fp3, "H %06X%06x", convert\_dec(rev\_address), \*length);

}

else{

fprintf(fp3, "H%-6s%06X%06X", rev\_label, convert\_dec(rev\_address), \*length);

}

}

/\* else{

printf("line : %d don't exist START\n", line\_num);

error\_flag = 1;

return error\_flag;

}\*/

first\_execution\_addr = convert\_dec(rev\_address);

while(strstr(rev\_operator, "END")==0){

fgets(line, sizeof(line), fp1);

line[strlen(line)-1] = '\0';

// printf("line : %s\n", line);

strcpy(buffer, line);

strcpy(line2, line);

if(line[0]!='.'){

strncpy(address, line, 4);

strncpy(label, line+5, 7);

label[6] = '\0';

strncpy(operator, line+13, 7);

operator[6] = '\0';

strncpy(operand, line+21, strlen(line)-21);

operand[strlen(operand)-1] = '\0';

if(!rev\_address){

rev\_address = " ";

}

rev\_label = strtok(label, " \t");

if(!rev\_label){

rev\_label = " ";

}

rev\_operator = strtok(operator, " \t");

rev\_operand = strtok(operand, " \t");

// operand2 = strtok(NULL, "");

if(!rev\_operand){

rev\_operand = " ";

}

if(strcmp(rev\_operator, "RESW")==0){

fprintf(fp2, "%3d %s\n", line\_num, buffer);

line\_num = line\_num+5;

enter\_flag = 1;

}

else if(strcmp(rev\_operator, "RESB")==0){

fprintf(fp2, "%3d %s\n", line\_num, buffer);

line\_num = line\_num+5;

enter\_flag = 1;

}

else if(strcmp(rev\_operator, "END")==0){

fprintf(fp2, "%3d %s\n", line\_num, buffer);

// fprintf(fp3, "\nE%06X", first\_execution\_addr);

}

else if(strcmp(rev\_operator, "WORD")==0){

object\_code = char\_dec(rev\_operand);

if(object\_code>=0&&object\_code<=16777215){

fprintf(fp2, "%3d %s %06X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+3>30 || enter\_flag ==1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%06X", object\_code);

object\_code\_byte = object\_code\_byte+3;

}

else{

printf("%s error : range over!\n", line2);

error\_flag = 1;

break;

}

line\_num = line\_num+5;

}

else if(strcmp(rev\_operator, "BYTE")==0){

rev\_operand2 = strtok(rev\_operand+1, "'");

fprintf(fp2, "%3d %s ", line\_num, buffer);

if(rev\_operand[0]=='C'){

for(i = 0; i<strlen(rev\_operand2); i++){

fprintf(fp2, "%2X", rev\_operand2[i]);

// fprintf(fp3, "%2X", rev\_operand2[i]);

}

fprintf(fp2, "\n");

if(object\_code\_byte+strlen(rev\_operand2)>30 || enter\_flag ==1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

for(i = 0; i<strlen(rev\_operand2); i++){

fprintf(fp3, "%2X", rev\_operand2[i]);

}

object\_code\_byte = object\_code\_byte+strlen(rev\_operand2);

}

else if(rev\_operand[0] == 'X'){

for(i = 0; i<strlen(rev\_operand2); i++){

fprintf(fp2, "%c", rev\_operand2[i]);

// fprintf(fp3, "%c", rev\_operand2[i]);

}

if(object\_code\_byte+strlen(rev\_operand2)/2>30 || enter\_flag == 1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

fprintf(fp3, "%02X", 0);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

for(i = 0; i<strlen(rev\_operand2); i++){

fprintf(fp3, "%c", rev\_operand2[i]);

}

object\_code\_byte = object\_code\_byte+strlen(rev\_operand2)/2;

fprintf(fp2, "\n");

}

else{

printf("%s error : invalid operand!\n", line2);

error\_flag = 1;

break;

}

line\_num = line\_num+5;

}

else if(strcmp(rev\_operator, "BASE")==0){

fprintf(fp2, "%3d %s\n", line\_num, buffer);

line\_num = line\_num+5;

}

else if(strcmp(rev\_operator, "RSUB")==0){

fprintf(fp2, "%3d %s %06X\n", line\_num, buffer, 0x4F0000);

if(object\_code\_byte+3>30 || enter\_flag ==1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%06X", 0x4F0000);

object\_code\_byte +=3;

line\_num = line\_num+5;

// fprintf(fp3, "%06X", 0x4F0000);

}

else if(rev\_operator[0] != '+'){

hash\_node = return\_opcode\_node(hash\_header, rev\_operator);

if(hash\_node){

if(strcmp(hash\_node->ss, "2")==0){

// printf("opcode type : %s %s\n", hash\_node->instructions,hash\_node->ss);

object\_code = format2(hash\_node->opcode, rev\_operand);

// printf("object\_code : %d\n", object\_code);

if(object\_code>=0)

fprintf(fp2, "%3d %s %4X\n", line\_num, buffer, object\_code);

else{

printf("%s error : invalid register!\n", line2);

error\_flag = 1;

return error\_flag;

}

if(object\_code\_byte+2>30||enter\_flag == 1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%04X", object\_code);

object\_code\_byte = object\_code\_byte+2;

}

else if(strcmp(hash\_node->ss, "1")==0){

object\_code = convert\_dec(hash\_node->opcode);

fprintf(fp2, "%3d %s %2X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+1>30 || enter\_flag==1){

fprintf(fp3, "\nT%06X%01X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%04X", object\_code);

object\_code\_byte+=1;

}

else if(strcmp(hash\_node->ss, "3/4")==0){

if(rev\_operand[0] == '#'){

sym\_node = search\_symbol\_table(rev\_operand+1, symbol\_header);

// printf("# symbol : %s\n", rev\_operand+1);

if(sym\_node){

// printf("exist symbol\n");

pc = convert\_dec(rev\_address)+3;

ta = sym\_node->address;

if(ta-pc>=-2048 && ta-pc<=2047){

nixbpe = 18;

object\_code = format3(hash\_node->opcode, nixbpe, ta-pc);

fprintf(fp2, "%3d %s %06X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+3>30 || enter\_flag == 1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%06X", object\_code);

object\_code\_byte = object\_code\_byte+3;

// line\_num = line\_num+5;

}

//base

else if(ta-\*base\_address>=0 && ta-\*base\_address<=4095){

// disp = ta-\*base\_address;

nixbpe = 20;

object\_code = format3(hash\_node->opcode, nixbpe, ta-\*base\_address);

fprintf(fp2, "%3d %s %06X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+3>30 || enter\_flag == 1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%06X", object\_code);

object\_code\_byte = object\_code\_byte+3;

}

else{

printf("%s error : range over!\n", line2);

error\_flag = 1;

break;

}

//range over

}

else{

constant = char\_dec(rev\_operand+1);

if(constant>=-2048 && constant<=4095){

nixbpe = 16;

object\_code = format3(hash\_node->opcode, nixbpe, constant);

fprintf(fp2, "%3d %s %06X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+3>30 || enter\_flag == 1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%06X", object\_code);

object\_code\_byte +=3;

}//range over

}

}//if(operant[0] == '#')

else if(rev\_operand[0] == '@'){

sym\_node = search\_symbol\_table(rev\_operand+1, symbol\_header);

if(sym\_node){

pc = convert\_dec(rev\_address)+3;

ta = sym\_node->address;

if(ta-pc>=-2048 && ta-pc<=2047){

nixbpe = 34;

object\_code = format3(hash\_node->opcode, nixbpe, ta-pc);

fprintf(fp2, "%3d %s %06X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+3>30 || enter\_flag == 1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%06X", object\_code);

object\_code\_byte +=3;

}

else if(ta-\*base\_address>=0 && ta-\*base\_address<=4096){

nixbpe = 36;

// disp = ta-\*base\_address;

object\_code = format3(hash\_node->opcode, nixbpe, ta-\*base\_address);

fprintf(fp2, "%3d %s %06X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+3>30 || enter\_flag == 1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%06X", object\_code);

object\_code\_byte+=3;

}

else{

printf("%s error : range over\n", line2);

error\_flag = 1;

break;

}

}

else{

printf("%s error : symbol doesn't exist in SYMTAB\n", line2);

error\_flag = 1;

return error\_flag;

}

}

else if(strstr(rev\_operand, ",")){

rev\_operand2 = strtok(rev\_operand, ",");

sym\_node = search\_symbol\_table(rev\_operand2, symbol\_header);

if(sym\_node){

pc = convert\_dec(rev\_address)+3;

ta = sym\_node->address;

// printf("rev\_operand2 : %s\n", rev\_operand2);

if(ta-pc>=-2048 && ta-pc<=2047){

nixbpe = 58;

object\_code = format3(hash\_node->opcode, nixbpe, ta-pc);

fprintf(fp2, "%3d %s %06X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+3>30 || enter\_flag==1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%06X", object\_code);

object\_code\_byte +=3;

}

else if(ta - \*base\_address>=0 && ta- \*base\_address<=4096){

nixbpe = 60;

object\_code = format3(hash\_node->opcode, nixbpe, ta-\*base\_address);

fprintf(fp2, "%3d %s %06X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+3>30 || enter\_flag==1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%06X", object\_code);

object\_code\_byte+=3;

}

else{

printf("%s error : range over!\n", line2);

error\_flag = 1;

break;

}

}

else{

printf("%s error : symbol doesn't exist in SYMTAB\n", line2);

error\_flag = 1;

return error\_flag;

}

}

else{

sym\_node = search\_symbol\_table(rev\_operand, symbol\_header);

// printf("simple addressing, symbol %s!\n", rev\_operand+1);

if(sym\_node){

pc = convert\_dec(rev\_address)+3;

ta = sym\_node->address;

if(ta-pc>=-2048 && ta-pc<=2047){

nixbpe = 50;

object\_code = format3(hash\_node->opcode, nixbpe, ta-pc);

// printf("pc address : %04X object\_code : %06X\n", convert\_dec(rev\_address), object\_code);

fprintf(fp2, "%3d %s %06X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+3>30 || enter\_flag==1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%06X", object\_code);

object\_code\_byte +=3;

}

else if(ta-\*base\_address>=0 && ta-\*base\_address<=4096){

// disp = ta-\*base\_address;

nixbpe = 52;

object\_code = format3(hash\_node->opcode, nixbpe, ta-\*base\_address);

fprintf(fp2, "%3d %s %06X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+3>30 || enter\_flag==1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%06X", object\_code);

object\_code\_byte +=3;

}

else{

// printf("%s disp : %d target address : %d error!\n", buffer, disp, ta);

error\_flag = 1;

break;

}

}

else{

printf("%s error : symbol doesn't exist in SYMTAB\n", line2);

error\_flag = 1;

break;

}

}

}

line\_num = line\_num+5;

}

}//else if(rev\_operator[0]!='+')

else if(rev\_operator[0] == '+'){

hash\_node = return\_opcode\_node(hash\_header, rev\_operator+1);

if(hash\_node){

if(strcmp(hash\_node->ss, "3/4")==0){

if(rev\_operand[0]=='#'){

sym\_node = search\_symbol\_table(rev\_operand+1, symbol\_header);

if(sym\_node){

ta = sym\_node->address;

nixbpe = 17;

object\_code = format4(hash\_node->opcode, nixbpe, ta);

fprintf(fp2, "%3d %s %08X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+4>30 || enter\_flag ==1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

modicode[midx] = convert\_dec(rev\_address)+1;

midx++;

fprintf(fp3, "%08X", object\_code);

object\_code\_byte += 4;

}//if(sym\_node)

else{

constant = char\_dec(rev\_operand+1);

nixbpe = 17;

object\_code = format4(hash\_node->opcode, nixbpe, constant);

fprintf(fp2, "%3d %s %08X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+4>30 || enter\_flag==1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

fprintf(fp3, "%08X", object\_code);

object\_code\_byte +=4;

}

}//if(rev\_operand[0]=='#')

else if(rev\_operand[0]=='@'){

sym\_node = search\_symbol\_table(rev\_operand+1, symbol\_header);

if(sym\_node){

ta = sym\_node->address;

nixbpe = 33;

object\_code = format4(hash\_node->opcode, nixbpe, ta);

fprintf(fp2, "%3d %s %08X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+4>30 || enter\_flag==1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

modicode[midx] = convert\_dec(rev\_address)+1;

midx++;

fprintf(fp3, "%08X", object\_code);

object\_code\_byte+=4;

}

else{

printf("%s error : symbol doesn't exist in SYMTAB\n", line2);

error\_flag = 1;

break;

}

}//else if(rev\_operand[0]=='@')

else{

sym\_node = search\_symbol\_table(rev\_operand, symbol\_header);

if(sym\_node){

ta = sym\_node->address;

nixbpe = 49;

object\_code = format4(hash\_node->opcode, nixbpe, ta);

fprintf(fp2, "%3d %s %08X\n", line\_num, buffer, object\_code);

if(object\_code\_byte+4>30 || enter\_flag == 1){

fprintf(fp3, "\nT%06X%02X", convert\_dec(rev\_address), object\_code\_len[oidx]);

object\_code\_byte = 0;

enter\_flag = 0;

oidx++;

}

modicode[midx] = convert\_dec(rev\_address)+1;

midx++;

fprintf(fp3, "%08X", object\_code);

object\_code\_byte+=4;

}

else{

printf("%s error : symbol doesn't exist in SYMTAB\n", line2);

error\_flag = 1;

break;

}

}//else

}//if(strcmp(hash\_node->ss, "3/4")

}//if(hash\_node)

line\_num = line\_num+5;

}//else if(rev\_operator[0]=='+')

// printf("%s %s %s %s\n", rev\_address, rev\_label, rev\_operator, rev\_operand);

}//if(line[0]!='.')

// printf("\n");

}

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

fprintf(fp3, "\nM%06X05", modicode[i]);

}

fprintf(fp3, "\nE%06X\n", first\_execution\_addr);

printf("output file : [%s], [%s]\n", obj\_file\_name, lst\_file\_name);

fclose(fp1);

fclose(fp2);

fclose(fp3);

return error\_flag;

}

//user input->standard input

//standard input : command\_startaddress,endaddress,value

void revise\_input(char \*input1, char \*input2){

int i, len1, len2, j;

int space, comma;

space = comma = 0;

len2 = 0;

len1 = strlen(input1);

input1[len1-1] = '\0';

for(i = 0; i<len1; i++){

if(input1[i] != ' ' && input1[i] != ' '){

input2[len2] = input1[i];

len2++;

}

else if(input1[i] == ' '||input1[i]==' '){

if(input1[i-1] != ' ' && input1[i+1] != ',' && input1[i-1] != ',' && i-1>=0 && i+1<len1 && input1[i-1] != ' ')

input2[len2++] = '\_';

}

}

if(input2[len2-2] == '\_')

input2[len2-2] = '\0';

for(i = 0; i<len2; i++){

if(input2[i] == '\_' && i+1 < len2 && input2[i+1] == ','){

for(j = i; j<len2; j++){

input2[j] = input2[j+1];

}

}

}

}

//determine mode for user input

int mode(char \*input){

int result, comma, underbar, character;

int i, len;

int comma1, character1;

char \*token, \*token2;

comma = result = 0;

comma1 = 0;

underbar = character = character1 = 0;

len = strlen(input);

if(input[0] == '\0')

result = 0;

else{

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

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

comma++;

comma1 = i;

if(input[i+1] != '\0')

character++;

}

else if(input[i] == '\_')

underbar++;

}

if(input[comma1+1]!='\0')

character1++;

token = strtok(input, "\_");

if(strcmp(token, "help")==0 || strcmp(token, "h")==0)

result = 1;

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

result = 2;

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

result = 3;

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

result = 4;

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

if(comma==0 && underbar ==0)

result = 5;

else if(comma==0 && underbar > 0)

result = 6;

else if(comma == 1 && underbar ==1&&character > 0)

result = 7;

}

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

if(comma==1 && underbar==1&&character>0)

result = 8;

}

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

if(comma==2&&character1>0)

result = 9;

}

else if(strcmp(token, "reset")==0)

result = 10;

else if(strcmp(token, "opcode")==0)

result = 11;

else if(strcmp(token, "opcodelist")==0)

result = 12;

else if(strcmp(token, "type")==0)

result = 13;

else if(strcmp(token, "symbol")==0)

result = 14;

else if(strcmp(token, "assemble")==0)

result = 15;

else if(strcmp(token, "progaddr")==0)

result = 16;

else if(strcmp(token, "loader")==0)

result = 17;

else if(strcmp(token, "run")==0)

result = 18;

else if(strcmp(token, "bp")==0)

result = 19;

}

return result;

}

void save\_progaddr(char \*input, int \*progaddr){

char \*token1, \*token2;

token1 = strtok(input, "\_");

token2 = strtok(NULL, "");

\*progaddr = convert\_dec(token2);

printf("Program starting address set to %#X\n", \*progaddr);

}

ext\_symbol \*create\_ext\_symbol\_node(char \*symbol, int address, int Length){

ext\_symbol \*tmp;

tmp = (ext\_symbol\*)malloc(sizeof(ext\_symbol));

strcpy(tmp->name, symbol);

tmp->addr = address;

tmp->length = Length;

tmp->link = NULL;

// printf("%s %x %x\n", tmp->name, tmp->addr, tmp->length);

return tmp;

}

void make\_estab(ext\_symbol \*\*header, char \*symbol, int address, int Length){

int index = 0;

ext\_symbol \*newnode;

ext\_symbol \*pointer;

char \*ssymbol;

ssymbol = strtok(symbol, " \n");

index = hash\_function(ssymbol);

newnode = create\_ext\_symbol\_node(ssymbol, address, Length);

// printf("%s %x %x\n", newnode->name, newnode->addr, newnode->length);

if(header[index] == NULL){

header[index] = newnode;

// printf("%s %x %X\n", header[index]->name, header[index]->addr, header[index]->length);

}

else{

pointer = header[index];

while(pointer->link!=NULL){

pointer = pointer->link;

}

pointer->link = newnode;

}

}

ext\_symbol \*search\_estab(ext\_symbol \*\*header, char \*symbol){

int hash\_key;

ext\_symbol \*ptr;

ext\_symbol \*tmp = NULL;

hash\_key = hash\_function(symbol);

ptr = header[hash\_key];

while(ptr != NULL){

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

tmp = ptr;

break;

}

ptr = ptr->link;

}

return tmp;

}

void free\_sorted\_estab\_node(ext\_symbol \*\*list){

ext\_symbol \*tmp;

while(\*list){

tmp = \*list;

\*list = (\*list)->link;

free(tmp);

}

}

void print\_estab(ext\_symbol \*\*header){

int i;

int total\_length = 0;

ext\_symbol \*pointer=NULL;//hash table 에서 움직이는 포인터

ext\_symbol \*new\_list=NULL;

ext\_symbol \*newnode;

ext\_symbol \*cur\_ptr;//new\_list에서 움직이는 포인터

ext\_symbol \*prev=NULL;

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

pointer = header[i];

while(pointer!=NULL){

newnode = create\_ext\_symbol\_node(pointer->name, pointer->addr, pointer->length);

if(new\_list==NULL){

new\_list = newnode;

// printf("%s %x %x\n", new\_list->name, new\_list->addr, new\_list->length);

}

else{

cur\_ptr = new\_list;

if(newnode->addr<new\_list->addr){

newnode->link = new\_list;

new\_list = newnode;

// printf("%s %x %x\n", new\_list->name, new\_list->addr, new\_list->length);

}

else{

while(1){

if(newnode->addr>cur\_ptr->addr){

if(cur\_ptr->link == NULL){

cur\_ptr->link = newnode;

break;

}

prev = cur\_ptr;

cur\_ptr = cur\_ptr->link;

}

else if(newnode->addr<=cur\_ptr->addr){

prev->link = newnode;

newnode->link = cur\_ptr;

break;

}

}

// prev->link = newnode;

// newnode->link = cur\_ptr;

// printf("%s %x %x\n", prev->name, prev->addr, prev->length);

}

}

pointer = pointer->link;

}

}

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

printf("section\tname\n");

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

while(new\_list){

if(new\_list->length!=0){

printf("%s\t\t%04X\t%04X\n", new\_list->name, new\_list->addr, new\_list->length);

total\_length = total\_length+new\_list->length;

}

else if(new\_list->length==0){

printf("\t\%s\t%04X\t\t\n", new\_list->name, new\_list->addr);

total\_length = total\_length+new\_list->length;

}

new\_list = new\_list->link;

}

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

printf("\t\ttotal length\t%X\n", total\_length);

free\_sorted\_estab\_node(&new\_list);

}

int linking\_loader\_pass1(char \*filename, ext\_symbol \*\*header, int \*csaddr){

char line[256];

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

int flag = 0;

int cs\_length;

int sym\_addr;

//반복문을 위한 index

int i;

if(fp == NULL){

printf("file open error\n");

return 1;

}

while(1){

if(fgets(line, sizeof(line), fp)==NULL)

break;

char symname[6];

char llength[6];

char aaddr[6];

if(line[0]=='H'){

strncpy(llength, line+13, 6);

sscanf(llength, "%X", &cs\_length);

strncpy(symname, line+1, 6);

make\_estab(header, symname, \*csaddr, cs\_length);

}

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

if(fgets(line, sizeof(line), fp) == NULL)

break;

else if(line[0]=='D'){

for(i = 1; i<strlen(line)-1; i = i+12){

strncpy(symname, line+i, 6);

strncpy(aaddr, line+i+6, 6);

sscanf(aaddr, "%X", &sym\_addr);

// printf("%s %x\n", symname, sym\_addr);

if(search\_estab(header, symname)==NULL){

// printf("%s %x %x\n", symname, sym\_addr, 0);

sym\_addr = \*csaddr+sym\_addr;

make\_estab(header, symname, sym\_addr, 0);

}

else if(search\_estab(header, symname)){

printf("pass1 error : external symbol duplicate!\n");

flag = 1;

break;

}

}

}

}

}

// printf("cslth : %04X\n", cs\_length);

\*csaddr = \*csaddr+cs\_length;

fclose(fp);

return flag;

}

int linking\_loader\_pass2(unsigned char \*\*memory, char \*filename, ext\_symbol \*\*header, int \*csaddr){

char line[256];//object file에서 한 줄씩 읽을 때

int cslth;

char symbol[6], iindex[2], \*ssymbol;

char Addr[6], Length[2], objcode[2];//T record에서 시작 위치와 길이, object code를 받기 위해서 선언한 변수

FILE \*fp;

int flag;

int addr, len, index, i;

int x, y;//memory의 index

int j, value;//t record에서 object code를 2개씩 읽기 위해서 선언한 변수

int length;//t record에서 object code들의 길이를 저장하기 위해서 선언한 변수

int sum, shift;//m record에서 memory에 있는 object code들의 합을 구하기 위해서 선언한 변수

//shift는 m record에서 값들을 shift해주기 위해서 선언한 변수

ext\_symbol \*node;

int ext\_reference[10]={-1};

char token[8] = {0};

char Cslth[6] = {0};

addr = 0;

ext\_reference[1] = \*csaddr;

flag = 0;

shift = 16;

fp = fopen(filename, "r");

if(fp==NULL){

printf("file open error!\n");

}

// printf("ext reference address : %X\n", ext\_reference[1]);

while(fgets(line, sizeof(line), fp)!= NULL){

if(line[0]=='H'){

strncpy(Cslth, line+13, 6);

sscanf(Cslth, "%X", &cslth);

// printf("cslth : %04X\n", cslth);

}

// printf("line1 : %s\n", line);

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

if(fgets(line, sizeof(line), fp)==NULL)

break;

// printf("line2 : %s\n", line);

if(line[0] == 'R'){

len = strlen(line);

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

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

strncpy(token, line+i, 8);

strncpy(iindex, token, 2);

strncpy(symbol, token+2, 6);

sscanf(iindex, "%X", &index);

// printf("token : %s\n",token);

ssymbol = strtok(symbol, " ");

// printf("symbol : %s, index : %X\n", symbol, index);

node = search\_estab(header, ssymbol);

// printf("symbol : %s,index : %x\n", node->name, node->addr);

if(node==NULL){

printf("external symbol not in estab\n");

flag = 1;

break;

}

ext\_reference[index] = node->addr;

// printf("ext reference address : %X\n", ext\_reference[index]);

}

}

else if(line[0] == 'T'){

j = 0;

len = strlen(line);

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

strncpy(Addr, line+1, 6);

strncpy(Length, line+7, 2);

sscanf(Addr, "%X", &addr);

sscanf(Length, "%X", &length);

// printf("Addr : %X, Length : %d\n", addr, length);

addr = addr+\*csaddr;

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

strncpy(objcode, line+9+j, 2);

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

value = convert\_dec(objcode);

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

x = addr/16;

y = addr%16;

memory[x][y] = value;

j = j+2;

addr = addr+1;

}

// printf("\n");

}

else if(line[0] == 'M'){

sum = 0;

strncpy(Addr, line+1, 6);

strncpy(Length, line+7, 2);

strncpy(iindex, line+10, 2);

sscanf(Addr, "%X", &addr);

sscanf(Length, "%X", &length);

sscanf(iindex, "%X", &index);

addr = addr+ \*csaddr;

if((ext\_reference[index] != -1)||(ext\_reference[index]==-1&&line[9]=='\n')){

// printf("ext\_reference[%d] = %X\n", index, ext\_reference[index]);

if(length%2==0){

for(i = addr; i<addr+length/2; i++){

sum = sum<<8;

x = i/16;

y = i%16;

sum = sum+memory[x][y];

}

if(sum>>23==1){

sum =sum+0xFF000000;

}

// printf("addr : %X sum = %06X\n", addr, sum);

}

else if(length%2==1){

for(i = addr; i<addr+length/2+1; i++){

sum = sum<<8;

x = i/16;

y = i%16;

sum = sum+memory[x][y];

}

if(sum>>23==1){

sum = sum+0xFF000000;

}

// printf("addr : %X sum = %06X\n", addr, sum);

}

// printf("addr : %X sum : %06X, %d\n", addr, sum, sum);

if(line[9]=='+')

sum = sum+ext\_reference[index];

else if(line[9] == '-')

sum = sum-ext\_reference[index];

else if(line[9] == '\n')

sum = sum+ext\_reference[1];

sum = sum&0xFFFFFF;

// printf("addr : %X sum = %06X\n", addr, sum);

if(length%2==0){

shift = 16;

for(i = addr; i<addr+length/2; i++){

x = i/16;

y = i%16;

memory[x][y] = (sum>>shift)&0xFF;

// printf("%02X", memory[x][y]);

shift = shift-8;

}

}

else if(length%2==1){

shift = 16;

for(i = addr; i<addr+length/2+1; i++){

x = i/16;

y = i%16;

memory[x][y] = (sum>>shift)&0xFF;

// printf("%02X", memory[x][y]);

shift = shift-8;

}

}

// printf("addr : %X sum = %06X\n", addr, sum);

}

else if(ext\_reference[index]==-1 && line[9] != '\n'){

printf("linking loader pass2 error\n");

flag = 1;

break;

}

}

}

}

\*csaddr = \*csaddr+cslth;

fclose(fp);

return flag;

}

void store\_operation(unsigned char \*\*memory, int ta, int value){

memory[ta/16][ta%16] = value/(16\*16\*16\*16);

memory[(ta+1)/16][(ta+1)%16] = (value%(16\*16\*16\*16))/(16\*16);

memory[(ta+2)/16][(ta+2)%16] = (value%(16\*16\*16\*16))%(16\*16);

}

int decision\_xbp(unsigned char \*\*memory, int current\_loc){

int k = current\_loc;

int x = (k+1)/16;

int y = (k+1)%16;

int xbp = -1;

int content = memory[x][y]/16;

if(content==0)

xbp = 0;

else if(content==1)

xbp = 1;

else if(content==2)

xbp = 2;

else if(content==4)

xbp = 4;

else if(content==8)

xbp = 8;

else if(content==10)

xbp = 10;

else if(content==12)

xbp = 12;

return xbp;

}

int get\_target\_addr(unsigned char \*\*memory, int format, int pc, int reg[10]){

int disp, ta, sign\_flag;

int xbp = decision\_xbp(memory, pc);

sign\_flag = 0;

if(format==4){

disp = memory[(pc+3)/16][(pc+3)%16]+memory[(pc+2)/16][(pc+2)%16]\*16\*16+(memory[(pc+1)/16][(pc+1)%16]%16)\*16\*16\*16\*16;

}

else if(format==3)

disp = memory[(pc+2)/16][(pc+2)%16]+(memory[(pc+1)/16][(pc+1)%16]%16)\*16\*16;

if(((memory[(pc+1)/16][(pc+1)%16]%16)>>3)==0x0001)

sign\_flag=1;

if(xbp == 0)

ta = disp;

else if(xbp==1)

ta = disp;

else if(xbp==2)

ta = disp+reg[PC]+format;

else if(xbp==4)

ta = disp+reg[B];

else if(xbp==8)

ta = disp+reg[X];

else if(xbp==10)

ta = disp+reg[PC]+reg[X];

else if(xbp==12)

ta = disp+reg[B]+reg[X];

if(sign\_flag==1){

ta = ta+0xF000;

ta = ta&0x0FFFF;

}

// printf("format : %d, sign : %d\n", format, sign\_flag);

// printf("xbp : %d\n", xbp);

// printf("ta in function : %06X\n", ta);

return ta;

}

int bp\_clear\_or\_set(char \*input){

char \*token1, \*token2;

int flag = 0;

token1 = strtok(input, "\_");

token2 = strtok(NULL, "\_");

if(token2){

if(strcmp(token2, "clear")==0)

flag = 0;

else

flag = 1;

}

else if(token2==NULL)

flag = 2;

// printf("flag : %d\n", flag);

return flag;

}

void bp\_set(int \*bp, char \*input, int \*bp\_count){

char \*token1, \*token2;

int addr;

token1 = strtok(input, "\_");

token2 = strtok(NULL, "\_");

// printf("token2 : %s\n", token2);

sscanf(token2, "%X", &addr);

// printf("addr : %04X\n", addr);

if(\*bp\_count==100){

printf("full bp array\n");

}

else{

bp[\*bp\_count]=addr;

(\*bp\_count)++;

}

printf("[ok] create breakpoint %04X\n", addr);

}

void bp\_print(int \*bp, int \*bp\_count){

int i;

if(\*bp\_count>0){

printf("breakpoints\n");

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

for(i = 0; i<\*bp\_count; i++){

printf("%04X\n", bp[i]);

}

}

else if(\*bp\_count==0)

printf("no breakpoints set\n");

}

void bp\_clear(int \*bp, int \*bp\_count){

int i;

for(i = 0; i<\*bp\_count; i++){

bp[i] = 0;

}

\*bp\_count=0;

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

}

int run(unsigned char \*\*memory, int progaddr, int endaddr, int reg[10], int \*bp, int \*bp\_count, int \*first\_run\_flag){

int opcode, ni, xbp, format;

int pc;

int r1, r2;

int error\_flag = 0;

int ta, value;

int jump\_flag, bp\_flag, pc\_in\_bp\_flag;

int k;

int break\_point;

int i;

if((\*first\_run\_flag)==0){

reg[PC] = progaddr;

reg[L] = endaddr;

(\*first\_run\_flag)=1;

}

// printf("reg[PC] : %04X, reg[L] = %04X\n", reg[PC], reg[L]);

while(reg[PC]<endaddr){

pc = reg[PC];

// printf("program counter : %04X\n", pc);

ni = decision\_ni(memory, pc);

opcode = decision\_opcode(memory, pc, ni);

/\* for(i = 0; i<(\*bp\_count); i++){

if(bp[i]==pc){

pc\_in\_bp\_flag=1;

break;

}

}

if(pc\_in\_bp\_flag==1){

bp\_flag = 1;

break\_point = pc;

}\*/

if(opcode==0xA0 || opcode==0xB4 || opcode==0xB8){

format = 2;

r1 = memory[(pc+1)/16][(pc+1)%16]/16;

r2 = memory[(pc+1)/16][(pc+1)%16]%16;

if(r1<0 || r1>9 || r1==7){

printf("error : first register is invalid register\n");

error\_flag = 1;

break;

}

else if(r2<0 || r2>9 || r2==7){

printf("error : second register is invalid register\n");

error\_flag = 1;

break;

}

}

else{

if(decision\_format(memory, pc)==4)

format = 4;

else if(decision\_format(memory, pc)==3)

format = 3;

ta = get\_target\_addr(memory, format, pc, reg);

k = ta;

//immediate addressing

if(ni==1)

value = ta;

else{

if(ta<0||ta>endaddr){

printf("error : target address range over\n");

break;

}

//indirect addressing

value = memory[(ta+2)/16][(ta+2)%16]+memory[(ta+1)/16][(ta+1)%16]\*16\*16+memory[ta/16][ta%16]\*16\*16\*16\*16;

if(ni==2)

ta = value;

}

}

switch(opcode){

//LDA

case 0x00 :

reg[A] = value;

break;

//LDB

case 0x68 :

reg[B] = value;

break;

//LDT

case 0x74 :

reg[T] = value;

break;

//LDCH

case 0x50 :

reg[A] = value/(16\*16\*16\*16);

break;

//STA

case 0x0C :

store\_operation(memory, ta, reg[A]);

break;

//STX

case 0x10 :

store\_operation(memory, ta, reg[X]);

break;

//STL

case 0x14 :

store\_operation(memory, ta, reg[L]);

break;

//STCH

case 0x54 :

memory[ta/16][ta%16] = reg[0]%256;

break;

//J

case 0x3C :

// printf("%02X%02X%02X\n", memory[k/16][k%16], memory[(k+1)/16][(k+1)%16], memory[(k+2)/16][(k+2)%16]);

reg[PC] = ta;

jump\_flag = 1;

break;

//JSUB

case 0x48 :

reg[L] = reg[PC] + format;

reg[PC] = ta;

jump\_flag = 1;

break;

//JLT

case 0x38 :

if(reg[SW]==2){

printf("error : condition doesn't set\n");

error\_flag = 1;

}

else if(reg[SW]==-1){

reg[PC] = ta;

jump\_flag = 1;

}

break;

//JEQ

case 0x30 :

if(reg[SW]==2){

printf("error : condition doesn't set\n");

error\_flag = 1;

}

else if(reg[SW]==0){

reg[PC] = ta;

jump\_flag = 1;

}

break;

//RSUB

case 0x4C :

reg[PC] = reg[L];

jump\_flag = 1;

break;

//COMP

case 0x28 :

if(reg[A]>value)

reg[SW] = 1;

else if(reg[A] == value)

reg[SW] = 0;

else

reg[SW] = -1;

// printf("reg[A] = %04X\n", reg[A]);

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

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

break;

//COMPR

case 0xA0 :

if(reg[r1]>reg[r2])

reg[SW] = 1;

else if(reg[r1]==reg[r2])

reg[SW] = 0;

else

reg[SW] = -1;

break;

//CLEAR

case 0xB4 :

reg[r1] = 0;

break;

//TIXR

case 0xB8 :

reg[X] = reg[X]+1;

if(reg[X]>reg[r1])

reg[SW] = 1;

else if(reg[X]==reg[r1])

reg[SW] = 0;

else

reg[SW] = -1;

break;

//TD

case 0xE0 :

reg[SW] = -1;

break;

//RD

case 0xD8 :

reg[A] = 0;

break;

//WD

/\* case 0xDC :

reg[X] = reg[T];

break;\*/

}

if(jump\_flag==0)

reg[PC] = reg[PC]+format;

jump\_flag = 0;

value = 0;

for(i = 0; i<(\*bp\_count); i++){

if(reg[PC]==bp[i]){

pc\_in\_bp\_flag=1;

break;

}

}

if(pc\_in\_bp\_flag==1){

bp\_flag=1;

break\_point = reg[PC];

}

if(bp\_flag==1)

break;

// printf("pc : %04X\n", pc);

// printf("reg pc : %04X\n", reg[PC]);

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

// printf("reg L : %04X\n", reg[L]);

// printf("reg SW : %04X\n", reg[SW]);

/\* printf("target address : %04X\n", ta);

printf("A : %06X X : %06X\n", reg[A], reg[X]);

printf("L : %06X PC: %06X\n", reg[L], reg[PC]);

printf("B : %06X S : %06X\n", reg[B], reg[S]);

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

printf("SW : %06X\n\n", reg[SW]);\*/

// printf("end program\n");

// if(pc==0x5036)

// break;

}

printf("A : %06X X : %06X\n", reg[A], reg[X]);

printf("L : %06X PC: %06X\n", reg[L], reg[PC]);

printf("B : %06X S : %06X\n", reg[B], reg[S]);

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

if(bp\_flag==1){

printf("Stop at checkpoint[%04X]\n", break\_point);

bp\_flag = 0;

}

else if(bp\_flag==0)

printf("End program\n");

return error\_flag;

}

int decision\_opcode(unsigned char \*\*memory, int current\_loc, int ni){

int result;

result = memory[current\_loc/16][current\_loc%16]-ni;

return result;

}

int decision\_format(unsigned char \*\*memory, int current\_loc){

if((memory[(current\_loc+1)/16][(current\_loc+1)%16]/16)%2==1)

return 4;

else

return 3;

}

int decision\_ni(unsigned char \*\*memory, int current\_loc){

int k = current\_loc;

int ni = -1;

//n = 0, i = 0, sic machine

if(memory[k/16][k%16]%4==0){

ni = 0;

}

//n = 0, i = 1, immediate addressing

else if(memory[k/16][k%16]%4==1){

ni = 1;

}

//n = 1, i = 0, idirect addressing

else if(memory[k/16][k%16]%4==2){

ni = 2;

}

//n = 1, i = 1, simple addressing

else if(memory[k/16][k%16]%4==3){

ni = 3;

}

return ni;

}

//free memory array

void free\_memory\_array(unsigned char \*\*memory){

int i;

for(i = 0; i<65536; i++)

free(memory[i]);

free(memory);

}

//free history linked list

void free\_history\_node(history \*\*head){

history \*tmp;

while(\*head){

tmp = \*head;

\*head = (\*head)->link;

free(tmp);

}

}

//free hash table

void free\_hash\_node(hash \*\*head){

int i;

hash \*tmp;

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

while(\*head){

tmp = head[i];

head[i] = head[i]->link;

free(tmp);

}

}

}

void free\_symbol\_node(symbol \*\*head){

int i;

symbol \*tmp;

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

while(\*head){

tmp = head[i];

head[i] = head[i]->link;

free(tmp);

}

}

}

void free\_ext\_symbol\_node(ext\_symbol \*\*head){

int i;

ext\_symbol \*tmp;

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

while(head[i]){

tmp = head[i];

head[i] = head[i]->link;

free(tmp);

}

}

}

//main for program

int main(void){

char \*input;

char \*std\_input, \*std\_input2, \*std\_input3, \*std\_input4, \*std\_input5;

char \*aa, \*bb;

//load를 해줄 obj file들의 이름

char \*objname[3], \*ttemp;

int len, result, i, dump\_start;

//std\_input2's length

int len2;

int kkk;//linking loader pass1이 잘 되었는지 확인하기 위해 선언한 flag

history \*head;

history \*tmp;

unsigned char \*\*memory;

FILE \*fp1;

int bp[100] = {0};//break point

symbol \*symbol\_head[26];

hash \*hash\_head[20];

hash \*temp;

//estab의 head

ext\_symbol \*ext\_symbol\_header[20];

int symbol\_flag = 0;

int length=0;

int base\_address = -1;

int object\_code\_len[100]={0};

int reg[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 2};

//program의 시작 주소

int progaddr = 0;

//control section의 시작주소

int csaddr;

int endaddr;

int bp\_clear\_flag, first\_run\_flag;

int bp\_count;

fp1 = fopen("opcode.txt", "r");

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

head->link = NULL;

head->index = 0;

bp\_count=0;

dump\_start = 0;

first\_run\_flag = 0;

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

hash\_head[i] = NULL;

ext\_symbol\_header[i] = NULL;

}

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

symbol\_head[i] = NULL;

}

hashtable\_made(hash\_head, fp1);

memory = (unsigned char\*\*)calloc(65536, sizeof(unsigned char\*));

for(i = 0; i<65536; i++)

memory[i] = (unsigned char\*)calloc(16, sizeof(unsigned char));

while(1){

printf("sicsim>");

input = (char\*)malloc(sizeof(char)\*100);

fgets(input, sizeof(char)\*100, stdin);

len = strlen(input);

csaddr = progaddr;

std\_input = (char\*)malloc(sizeof(char)\*101);

revise\_input(input, std\_input);

len2 = strlen(std\_input);

std\_input2 = (char\*)malloc(sizeof(char)\*len2);

std\_input3 = (char\*)malloc(sizeof(char)\*len2);

std\_input4 = (char\*)malloc(sizeof(char)\*len2);

std\_input5 = (char\*)malloc(sizeof(char)\*len2);

strcpy(std\_input2, std\_input);

// printf("std\_input : %s\n", std\_input2);

strcpy(std\_input3, std\_input);

strcpy(std\_input4, std\_input);

strcpy(std\_input5, std\_input);

// printf("input : %s\n", std\_input4);

result = mode(std\_input);

aa = strtok(std\_input3, "\_");

bb = strtok(NULL, "");//file name

// printf("%s %s\n", aa, bb);

//print help

// ttemp = strtok(std\_input4, "\_");

// objname1 = strtok(NULL, "\_");

// objname2 = strtok(NULL, "\_");

// objname3 = strtok(NULL, "");

if (result == 1) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

print\_help();

}

//print directory

else if (result == 2) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

print\_directory();

}

//quit

else if (result == 3) {

free(input);

free(std\_input);

free(std\_input2);

free\_hash\_node(hash\_head);

free\_symbol\_node(symbol\_head);

free\_history\_node(&head);

free\_memory\_array(memory);

if(remove("intermediate.txt")==-1)

printf("file remove error!\n");

break;

}

//print history

else if (result == 4) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

print\_history(head);

}

//print dump 1

else if (result == 5) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

print\_dump1(memory, &dump\_start);

}

//print dump 2

else if (result == 6 && !print\_dump2(memory, std\_input2, &dump\_start)) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

}

//print dump3

else if (result == 7 && !print\_dump3(memory, std\_input2, &dump\_start)) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

}

//edit

else if (result == 8 && !edit(memory, std\_input2)) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

}

//fill

else if (result == 9 && !fill(memory, std\_input2)) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

}

//reset

else if (result == 10) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

reset(memory);

}

//opcode mnemonic

else if (result == 11&&search\_opcode\_mnemonic(hash\_head, std\_input2)<0) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

}

//opcodelist

else if (result == 12) {

tmp = create\_history\_node(input);

history\_made(&head, tmp);

opcodelist\_print(hash\_head);

}

else if(result == 13){

tmp = create\_history\_node(input);

history\_made(&head, tmp);

type\_filename(bb);

}

else if(result==14){

tmp = create\_history\_node(input);

history\_made(&head, tmp);

if(symbol\_flag==1)

print\_symbol\_table(symbol\_head);

else

printf("because of assemble error, we can't make symbol\_table\n");

}

else if(result==15 && !pass1(bb, &length, &base\_address, symbol\_head, hash\_head, object\_code\_len)){

if(!pass2(bb, &length, &base\_address, symbol\_head, hash\_head, object\_code\_len)){

tmp = create\_history\_node(input);

history\_made(&head, tmp);

symbol\_flag = 1;

}

}// aa = strtok(std\_input2, "\_");

// bb = strtok(NULL, "");

// printf("%s %s\n", aa, bb);

// pass1(bb,&length, &base\_flag, symbol\_head, hash\_head);

else if(result == 16){

save\_progaddr(std\_input2, &progaddr);

// csaddr = progaddr;

}

else if(result == 17){

// if(objname1 && !objname2 && !objname3 && !linking\_loader\_pass1(objname1, ext\_symbol\_header, &csaddr)){

ttemp = strtok(std\_input4, "\_");

objname[0] = strtok(NULL, "\_");

objname[1] = strtok(NULL, "\_");

objname[2] = strtok(NULL, "\_");

kkk = 1;

for(i = 0; i<3; i++){

// printf("object name : %s\n", objname[i]);

if(objname[i]){

if(!linking\_loader\_pass1(objname[i], ext\_symbol\_header, &csaddr)){

kkk = 0;

}

else

break;

}

else

break;

}

csaddr = progaddr;

// printf("csaddr : %04X\n", csaddr);

if(kkk == 0){

for(i = 0; i<3; i++){

if(objname[i]){

if(!linking\_loader\_pass2(memory, objname[i], ext\_symbol\_header, &csaddr)){

kkk = 0;

}

else

break;

}

else

break;

}

}

// else if(objname1 && objname2 && !objname3 && !linking\_loader\_pass1(objname1, ext\_symbol\_header, &csaddr)&& !linking\_loader\_pass1(objname1, )

if(kkk == 0){

print\_estab(ext\_symbol\_header);

}

// printf("loading good\n");

endaddr = csaddr;

// printf("endaddr : %04X\n", endaddr);

free\_ext\_symbol\_node(ext\_symbol\_header);

}

else if(result==18&&!run(memory, progaddr, endaddr, reg, bp, &bp\_count, &first\_run\_flag)){

// printf("good\n");

}

else if(result==19){

bp\_clear\_flag = bp\_clear\_or\_set(std\_input2);

if(bp\_clear\_flag==0){

// printf("bp clear\n");

bp\_clear(bp, &bp\_count);

}

else if(bp\_clear\_flag==1){

// printf("bp set\n");

bp\_set(bp, std\_input5, &bp\_count);

}

else if(bp\_clear\_flag==2){

bp\_print(bp, &bp\_count);

// printf("bp print\n");

}

}

//wrong command

else if(result ==0)

printf("wrong command!\n");

}

}