과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 2>>

**서강대학교 컴퓨터공학과**

**[20151597]**

**[임준묵]**

목 차

1. 프로그램 개요 4

2. 프로그램 설명 4

2.1 프로그램 흐름도 4

3. 모듈 정의(20151597.c) 4

3.1 모듈 이름 : void print\_help() 4

3.1.1 기능 4

3.1.2 사용 변수 4

3.2 모듈 이름: void type\_filename(char \*input) 4

3.2.1 기능 4

3.2.2 사용 변수 4

3.3 모듈이름: hash \*return\_opcode\_node(hash \*\*header, char \*opcode) 5

3.3.1 기능 5

3.3.2 사용변수 5

3.4 모듈이름: symbol \*search\_symbol\_table(char \*ssymbol, symbol \*\*header) 5

3.4.1 기능 5

3.4.2 사용변수 5

3.5 모듈이름: symbol \*create\_symbol\_node(char \*ssymbol, int addr) 5

3.5.1 기능 5

3.5.2 사용변수 5

3.6 모듈이름 : void print\_symbol\_table(symbol \*\*header) 5

3.6.1 기능 5

3.6.2 사용변수 5

3.7 모듈 이름 : Void make\_symbol\_table(symbol \*\*header, int addr, char \*ssymbol) 5

3.7.1 기능 5

3.7.2 사용변수 6

3.8 모듈 이름 : Int format2(char \*opcode, char \*operand) 6

3.8.1 기능 6

3.8.2 사용 변수 6

3.9 모듈 이름 : Int format3(char \*opcode, int nixbpe, int disp) 6

3.9.1 기능 6

3.9.2 사용 변수 6

3.10 모듈 이름 : Int format4(char \*opcode, int nixbpe, int disp) 6

3.10.1 기능 6

3.10.2 사용 변수 7

3.11 모듈 이름 : Int pass1(char \*file\_name, int \*length, int \*base\_address, symbol \*\*symbol\_header, hash \*\*hash\_header, int \*object\_code\_len) 7

3.11.1 기능 7

3.11.2 사용 변수 7

3.12 모듈 이름 : int pass2(char \*file\_name, int \*length, int \*base\_address, symbol \*\*symbol\_header, hash \*\*hash\_header, int \*object\_code\_len) 8

3.12.1 기능 8

3.12.2 사용 변수 8

4. 구조체 정의(20151597.h) 9

4.1 Symbol 구조체 9

4.1.1 선언 9

4.1.2 기능 9

4.1.3 구성원 9

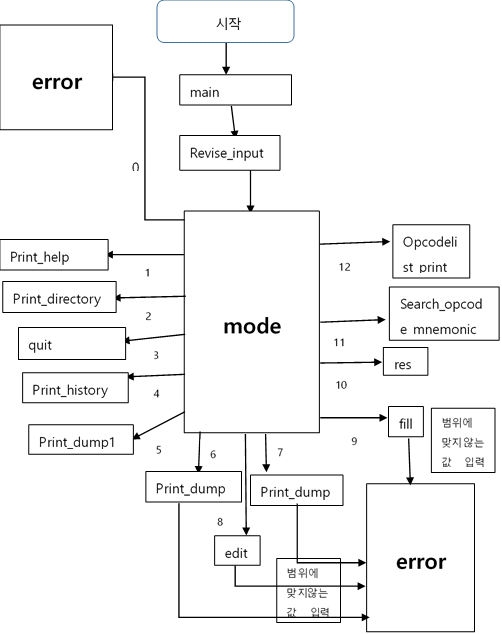
5. 코드 9

# 프로그램 개요

.asm 파일을 가지고 assemble 을 수행하여 .lst 파일과 .obj 파일을 생성하는 프로그램 이다.

# 프로그램 설명

## 프로그램 흐름도



type

13

assemble

15

symbol

15

파일이 없는 경우

.asm 파일이 아닌 경우

.

# 모듈 정의(20151597.c)

## 모듈 이름 : void print\_help()

### 기능

Project1 에서의 print\_help() 함수에 “assemble filename”, “type filename”, “symbol”을 추가하였다.

### 사용 변수

없음

## 모듈 이름: void type\_filename(char \*input)

### 기능

디렉토리 내에 있는 파일들의 내용을 출력해주는 함수 이다.

### 사용 변수

Char \*input : 열어서 내용을 출력해줄 파일의 이름이다.

FILE \*fp : 열어줄 파일의 파일 스트림 이다.

Char c : 파일 안에 있는 내용물들을 나타내는 문자 이다.

## 모듈이름: hash \*return\_opcode\_node(hash \*\*header, char \*opcode)

### 기능

Operator가 속해있는 노드를 반환해주는 함수 이다.

### 사용변수

Hash \*\*header : 해시 테이블을 가리키는 헤드 노드 이다.

Char \*opcode : operator를 가리키는 변수 이다.

int hash\_key : 변수 opcode의 해시 값을 가리키는 변수 이다.

Hash \*ptr , \*temp : 해시 테이블 연결리스트의 노드를 가리키는 변수 이다.

## 모듈이름: symbol \*search\_symbol\_table(char \*ssymbol, symbol \*\*header)

### 기능

Symbol table 내에서 symbol이 속해 있는 노드를 반환해준다.

### 사용변수

Char \*ssymbol : symbol을 가리키는 변수 이다.

Symbol \*\*header : symbol table의 헤드 노드를 가리키는 변수 이다.

Int hash\_key : symbol의 해시 값을 저장하는 변수 이다.

Symbol \*ptr, \*tmp : symbol table 연결 리스트의 노드를 가리키는 변수 이다.

## 모듈이름: symbol \*create\_symbol\_node(char \*ssymbol, int addr)

### 기능

Symbol 과 symbol 의 주소를 입력 받아서 symbol table 연결리스트의 노드 한 개를 만들어 주는 함수 이다.

### 사용변수

Char \*ssymbol : symbol을 가리키는 변수 이다.

Int addr : symbol의 주소를 가리키는 변수 이다.

Symbol \*tmp : symbol table 연결리스트의 노드를 가리키는 변수 이다. 이 노드를 반환해준다.

## 모듈이름 : void print\_symbol\_table(symbol \*\*header)

### 기능

Symbol table을 출력해주는 함수 이다. 알파벳 역순으로 출력을 해준다.

### 사용변수

Symbol \*\*header : symbol table의 헤드 노드를 가리키는 변수 이다.

Int I : 반복문 사용을 위해 선언한 변수 이다.

Symbol \*pointer : symbol table 연결리스트의 노드를 가리키는 변수이다.

## 모듈 이름 : Void make\_symbol\_table(symbol \*\*header, int addr, char \*ssymbol)

### 기능

Symbol과 symbol의 주소를 받아서 symbol table 을 만들어 주는 함수 이다.

### 사용변수

Symbol \*\*header : symbol table 연결리스트의 헤드 노드를 가리키는 변수 이다.

Int addr : symbol 의 주소를 가리키는 변수 이다.

Char \*ssymbol : symbol을 가리키는 변수 이다.

Int index : symbol table의 각각의 헤드 노드에 대한 index를 가리키는 변수 이다.

Int I : 반복문 사용을 위해 선언한 변수 이다.

Symbol \*newnode : 추가할 노드를 가리키는 변수 이다.

Symbol \*pointer : symbol table 연결리스트의 노드를 가리키는 변수 이다.

Symbol \*prev : symbol table 연결 리스트의 노드를 가리키는 변수 이다.(pointer가 가리키는 노드의 이전의 노드를 가리킨다.)

## 모듈 이름 : Int format2(char \*opcode, char \*operand)

### 기능

포맷이 2인 operator에 대해서 operator와 레지스터를 입력 받아서 object code를 반환해주는 함수 이다.

### 사용 변수

Char \*opcode : operator의 opcode를 가리키는 변수 이다.

Char \*operand : operand를 가리키는 변수 이다. Format2 이므로 레지스터를 가리킨다.

Int sum : object code를 계산한 값을 저장하는 변수 이다. 이 값을 반환해준다.

Int opcd : 문자로 표현된 opcode를 16진수로 변환한 값을 저장하는 변수 이다.

Int value\_reg1, value\_reg2 : 레지스터의 값을 저장하는 변수 이다.

Char reg1[10], reg2[10] : operand로 들어온 문자열을 comma로 구분해서 저장하는 변수 이다. Operand에 레지스터 하나만 들어오면 reg2[10]은 0이 된다.

## 모듈 이름 : Int format3(char \*opcode, int nixbpe, int disp)

### 기능

포맷이 3/4인 operator에 대해서 object code 의 포맷이 3인 경우의 object code를 반환해주는 함수 이다.

### 사용 변수

Char \*opcode : operator의 opcode를 가리키는 변수이다.

Int nixbpe : n, I, x, b, p, e의 값을 10진수의 정수로 변환한 값을 저장한 변수 이다.

Int disp : displacement를 저장하는 변수 이다.

Int opcd : 문자로 표현된 opcode를 16진수로 변환한 값을 저장하는 변수 이다.

Int sum : object code를 계산한 값을 저장하는 변수 이다. 이 값을 반환해준다.

Int disp2 : disp 가 음수인 경우 이 값을 2의 보수법을 이용해서 변환한 값을 저장한 변수 이다.

## 모듈 이름 : Int format4(char \*opcode, int nixbpe, int disp)

### 기능

포맷이 3/4인 operator에 대해서 object code의 포맷이 4인 경우의 object code를 반환해 주는 함수 이다.

### 사용 변수

Char \*opcode : operator의 opcode를 가리키는 변수 이다.

Int nixbpe : n, I, x, b, p, e의 값을 10진수의 정수로 변환한 값을 저장한 변수 이다.

Int disp : operand의 target address를 저장한 변수 이다.

Int opcd : 문자로 표현된 opcode를 16진수로 변환한 값을 저장하는 변수 이다.

Int sum : object code를 계산한 값을 저장하는 변수 이다. 이 값을 반환해준다.

Int disp2 : disp가 음수인 경우 이 값을 2의 보수법을 이용해서 변환한 값을 저장한 변수 이다.

## 모듈 이름 : Int pass1(char \*file\_name, int \*length, int \*base\_address, symbol \*\*symbol\_header, hash \*\*hash\_header, int \*object\_code\_len)

### 기능

.asm파일을 읽으면서 pass1을 수행한다. 즉, 각 instruction에 대해서 주소를 부여하고 symbol table을 생성한다. Error 가 있는 경우에는 1을 반환하고 그렇지 않은 경우에는 0을 반환한다.

### 사용 변수

Char \*file\_name : 파일의 이름을 가리키는 변수 이다.

Int \*length : .asm 파일의 길이를 가리키는 변수 이다.

Int \*base\_address : base 주소를 저장하는 변수 이다.

Symbol \*\*symbol\_header : symbol table 연결 리스트를 가리키는 변수 이다.

Hash \*\*hash\_header : hash table 연결리스트를 가리키는 변수 이다.

Int \*object\_code\_len : .obj 파일에 object code를 쓸 때 각 줄의 길이를 저장하는 변수 이다.

FILE \*fp1, \*fp2 : 각각 읽어야 하는 파일의 파일 스트림, 써야 하는 파일의 파일 스트림 포인터 변수 이다.

Char line[128] : 파일에서 읽은 한 줄을 저장하는 변수 이다.

Char line2[128], line3[128] : 전부 line[128]을 복사해서 저장하는 변수 이다.

Char \*name1, \*name2 : file\_name을 dot으로 구분 했을 때 각각을 저장하는 변수 이다.

Char buffer[20] : file\_name을 복사해서 저장하는 변수 이다.

char \*label, \*Operator, \*operand : 각각 .asm파일에서 label, operator, operand를 저장하는 변수 이다.

Char \*operand2 : operand에 comma가 있는 경우 comma로 나누어진 부분을 저장하는 변수 이다.

Char \*temp : 시작 주소를 받아서 저장하는 변수 이다.

Char base\_operand[10] : base operand를 저장하는 변수 이다.

Int addr : 주소를 저장하는 변수 이다.

Int start\_addr : 시작 주소를 저장하는 변수 이다.

Int oidx : object\_code\_len의 index를 가리키는 변수 이다. .obj 파일에서 개행을 해야 한다면 1이 증가 한다.

Int enter\_flag : .obj 파일에서 개행을 해야할지를 알려주는 flag 이다. 개행을 해야 한다면 1이 저장된다.

Int end\_flag : .asm 파일에 END 가 있는지를 알려주는 flag 이다. END 가 있다면 1이 저장된다.

Int assemble\_error\_flag : assemble 과정에서 error가 있는지 알려 주는 flag 이다. Error 가 있다면 1이 저장된다.

Int file\_error\_flag : .asm파일이 아닌 경우를 알려주는 flag 이다. .asm 파일이 아닌 경우에는 1이 저장된다.

## 모듈 이름 : int pass2(char \*file\_name, int \*length, int \*base\_address, symbol \*\*symbol\_header, hash \*\*hash\_header, int \*object\_code\_len)

### 기능

Pass1에서 symbol table을 만든 후에 이를 바탕으로 assemble을 수행해주는 함수 이다. 즉, .obj 파일과 .lst 파일을 만들어준다. Error가 없는 경우에는 0을 반환해주고 error가 있는 경우에는 1을 반환해준다.

### 사용 변수

Char \*file\_name : 파일의 이름을 가리키는 변수 이다.

Int \*length : .asm 파일의 길이를 가리키는 변수 이다.

Int \*base\_address : base 주소를 저장하는 변수 이다.

Symbol \*\*symbol\_header : symbol table 연결 리스트를 가리키는 변수 이다.

Hash \*\*hash\_header : hash table 연결리스트를 가리키는 변수 이다.

Int \*object\_code\_len : .obj 파일에 object code를 쓸 때 각 줄의 길이를 저장하는 변수 이다.

FILE \*fp1, \*fp2, \*fp3 : 각각 intermediate 파일, .lst 파일, .obj 파일을 가리키는 파일 스트림 포인터 변수 이다.

Char \*write\_file\_name, lst\_file\_name[20], obj\_file\_name[20] : \*write\_file\_name은 file\_name에서 dot 뒤에 있는 부분을 떼어낸 이름을 저장하는 변수 이다. 나머지는 각각 .lst 파일과 .obj 파일의 이름을 저장하는 변수 이다.

Char label[10], operator[10], operand[10], address[10] : intermediate 파일에서 읽은 한 줄을 label, operator, operand, address로 나누었을 때 각각을 저장하는 변수 이다.

Char \*rev\_label, \*rev\_operator, \*rev\_operand, \*rev\_address : label, operator, operand, address 에서 뒤의 공백을 제거한 뒤 각각을 저장하는 변수 이다.

Char \*rev\_operand2 : X’F1, C’EOF와 같은 rev\_operand에서 F1 과 EOF를 분리하여 저장하는 변수 이다.

Int line\_num : 몇 번째 줄인지를 저장하는 변수 이다.

int object\_code : object code를 저장하는 변수 이다.

Int object\_code\_byte : .obj 파일에서 한 줄에 쓰여진 object\_code 들의 전체 byte를 저장하는 변수 이다.

Int error\_flag : assemble 과정에서 error가 있는지 알려주는 flag 이다. Error가 있는 경우에는 1을 설정해주고 없는 경우에는 0을 설정해준다.

Int pc : program counter을 저장하는 변수 이다.

Int modicode[100], midx : modification을 처리하기 위해서 선언한 변수 이다. Modicode[100]은 수정되어야 할 object code의 시작 주소를 저장한다.

Int ta : target address를 저장하는 변수 이다.

Int nixbpe : n, I, x, b, p, e의 값을 10진수로 변환한 값을 저장한다.

Int constant : #뒤에 붙은 상수를 처리하기 위해서 선언한 변수 이다. #뒤에 붙은 상수의 값이 저장된다.

Int first\_execution\_addr : 프로그램에서 첫 번째로 수행되는 instruction의 주소를 저장하는 변수 이다.

Symbol \*sym\_node : symbol table의 노드를 가리키는 변수 이다.

Hash \*hash\_node : hash table의 노드를 가리키는 변수 이다.

Int oidx : object\_code\_len을 위한 index 이다.

# 구조체 정의(20151597.h)

## Symbol 구조체

### 선언

typedef struct \_symbol{

char sym[10];

int address;

struct \_symbol \*link;

}symbol;

### 기능

Symbol table을 만들기 위해서 선언한 구조체 이다.

### 구성원

Char sym[10] : symbol을 저장하기 위해서 선언하였다.

Int address : symbol의 주소를 저장하기 위해서 선언하였다.

Struct \_symbol \*link : symbol node의 pointer 이다. 연결리스트를 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");

}

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);

}

}

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 = 1;

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%06X", 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;

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++;

char \*token, \*token2;

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;

}

return result;

}

//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);

}

}

}

//main for program

int main(void){

char \*input;

char \*std\_input, \*std\_input2, \*std\_input3;

char \*aa, \*bb;

int len, result, i, dump\_start;

//std\_input2's length

int len2;

history \*head;

history \*tmp;

unsigned char \*\*memory;

FILE \*fp1;

symbol \*symbol\_head[26];

hash \*hash\_head[20];

hash \*temp;

int length=0;

int base\_address = -1;

int object\_code\_len[100]={0};

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

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

head->link = NULL;

head->index = 0;

dump\_start = 0;

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

hash\_head[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);

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);

strcpy(std\_input2, std\_input);

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

strcpy(std\_input3, std\_input);

result = mode(std\_input);

aa = strtok(std\_input3, "\_");

bb = strtok(NULL, "");//file name

// printf("%s %s\n", aa, bb);

//print help

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);

print\_symbol\_table(symbol\_head);

}

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);

}

// aa = strtok(std\_input2, "\_");

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

// printf("%s %s\n", aa, bb);

// pass1(bb,&length, &base\_flag, symbol\_head, hash\_head);

}

//wrong command

else if(result ==0)

printf("wrong command!\n");

}

}