과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 2>>

서강대학교 컴퓨터공학과

[20161631]

[임동진]

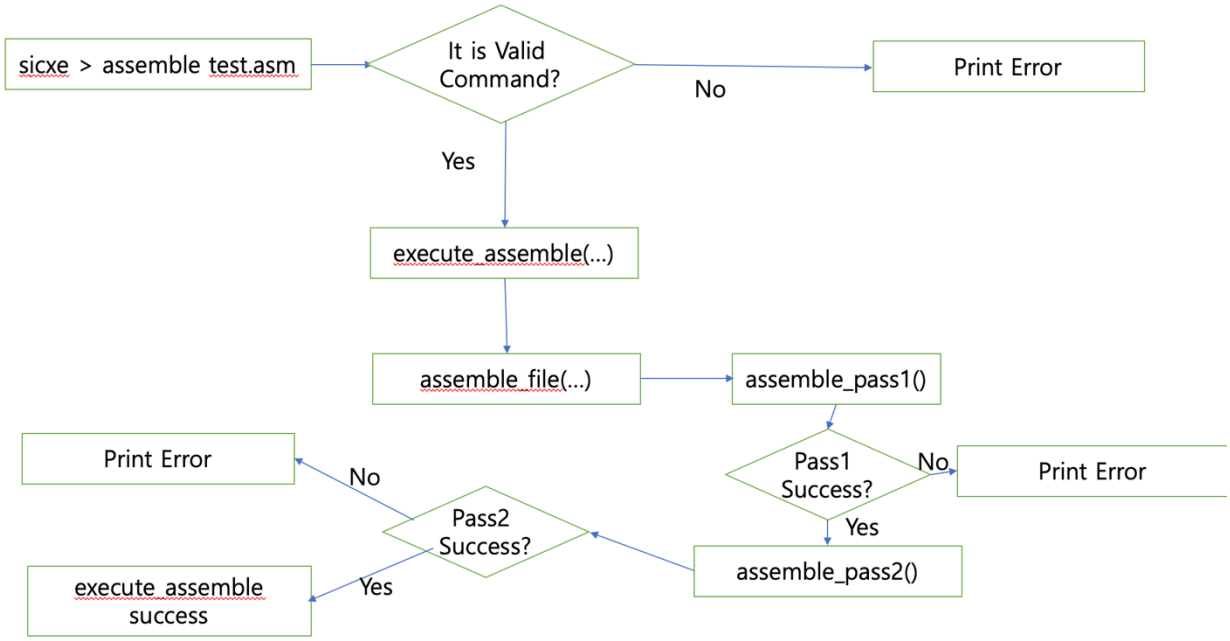
목 차

1. 프로그램 개요

이전 프로젝트에서 어셈블러 기능이 추가된 프로그램이다.
asm 소스파일을 assemble 하여 lst 파일과 obj 파일을 생성한다.

2. 프로그램 설명

2.1 프로그램 흐름도



< Assemble 과정 흐름도>

3. 모듈 정의

3.1 모듈 이름 : execute_assemble(Command *, State*)

3.1.1 *기능*

assemble_file() 함수를 호출하고 assemble_file() 함수의 성공 실패 여부(어셈블 성공 실패 여부)에 따라서 적절한. shell_status enum 값을 리턴해준다.

3.1.2 사용 변수

없음.

3.2 모듈 이름: assemble_file(State *, char *)

3.2.1 기능

이전의 symbol table 을 비워주고 새로운 symbol table 구조체를 할당받고, assemble_pass1 함수와 assemble_pass2 함수를 차례대로 호출해준다

3.2.2 사용변수

없음

3.3 모듈이름: assemble_pass1(State *, char *)

3.3.1 기능

어셈블러의 pass1 과정을 구현한 함수이다. 어셈블 모듈(assemble.c/.h)에 있는 함수들을 이용하여 pass2 에서 이용될 중간 파일인 .tmp 파일을 만들고, symbol table 을 만든다.

3.3.2 사용변수

FILE* asm_fp	.asm 파일에 대한 파일 포인터
char* tmp_file_name,	.tmp 파일의 이름을 저장할 변수
char* prefix	Prefix 를 저장한다(예를들어 2_5.asm 라면 prefix는 2_5이다.
char* lst_file_name	.lst 파일의 이름을 저장할 변수
char* obj_file_name	.obj 파일의 이름을 저장할 변수
Int location_counter	Location_counter 값을 저장할 변수
Int stmt_size	Statement 의 사이즈를 저장하는 변수
Int line_num	라인 번호를 저장한다
Statement stmt	Statement 구조체 변수
FILE* tmp_fp	.tmp 파일에 대한 파일 포인터

3.4 모듈이름: assemble_pass2(State *, char *)

3.4.1 기능

어셈블러의 pass2 과정을 구현한 함수이다. 어셈블 모듈(assemble.c/.h)에 있는 함수들을 이용하여 pass2 알고리즘을 실행하며, .lst 와 .obj 파일을 구축하고 어셈블 성공시에 파일을 유지한다.

3.4.2 사용변수

FILE *tmp_fp, *lst_fp, *obj_fp	.tmp .lst .obj 파일에 대한 파일 포인터 변수들
char *tmp_file_name, *lst_file_name, *obj_file_name	tmp .lst .obj 파일의 이름을 저장하는 변수
char* prefix	Prefix 를 저장하는 변수
int line_num	라인 번호를 저장하는 변수
char* b_buf	obj 파일에 기록될 내용을 저장하기위한 변수
Statement stmt	Statement 구조체 변수
int location_counter, r_lc, start_lc, location_counter_cnt, *location_couters	Location Counter 정보를 저장하는 변수들
Int obj_code	.obj 파일에 기록될 내용을 저장하는 변수
int location_counter_cnt	Location_
Int stmt_size	Statement의 사이즈를 저장하는 변수

bool is_base	Base 인지 아닌지 확인하기위한 변수
int base;	Base 정보를 저장하는 변수
char symb[11]	Symbol 정보를 저장하는 변수
char* obj_buf	Obj 파일에 기록될 내용을 저장하는 변수
Char* rec_head	Obj, lst 파일에 기록될 내용을 저장하는 변수

3.5 모듈이름: Command Layer - command_validate_util 모듈

3.5.1 기능

명령어 검증과 관련한 함수들로 이루어졌다.

bool validate_tokenizing(char *str, int token_cnt, int max_token_num);	토크나이징이 적절하게 되었는지 검증한다. 예를들어 "du , 1 1" 과 같이 파라미터 사이에 콤마가 없거나 이상한 위치에 콤마가 있는 등의 문제를 잡아낸다.
shell_status validate_parameters(Command *user_command)	사용자가 입력한 파라미터가 적절한 파라미터 값인지 검증한다. (파라미터 개수, 크기, 범위 등)
shell_status validate_dump_parameters(Command *user_command)	Dump 명령어의 파라미터를 검증한다. 파라미터가 적절한 주소값인지, 범위를 넘어가지는 않는지 등을 체크한다.
shell_status validate_opcode_parameters(Command *user_command)	Opcode 명령어의 파라미터를 검증한다.
shell_status validate_edit_parameters(Command *user_command)	Edit 명령어의 파라미터를 검증한다.
shell_status validate_fill_parameters(Command *user_command)	Fill 명령어의 파라미터를 검증한다.

3.5.2 사용변수

없음

3.6 모듈이름: Assemble 모듈의 구조체들

3.6.1. 기능

3.6.2. 사용변수 (구조체)

typedef struct statement { char* raw_input; int token_cnt; char* tokens[]; **bool** comment; Opcode* opcode; char* raw_symbol; **bool** tmp_bool; bool plus; } Statement;	어셈블리 코드 한줄을 저장하는 구조체

typedef union bits_format2{
struct
{
uint16_t r2 : 4;
uint16_t r1 : 4;
uint16_t opcode : 8;
} bits;
uint16_t res;
}BitsFormat2;
	format 2 인 Statement 일때,
	비트 값으로 저장하기 위한 변수이다..

typedef union bits_format3{
struct{
uint32_t disp : 12;
uint32_t e : 1;
uint32_t p : 1;
uint32_t b : 1;
uint32_t x : 1;
uint32_t i : 1;
uint32_t n : 1;
	format 3 인 Statement 일때,
	비트 값으로 저장하기 위한 변수이다..

uint32_t opcode : 6; } bits; uint32_t res; }BitsFormat3;	

typedef union bits_format4{
struct{
uint32_t addr: 20;
uint32_t e : 1;
uint32_t p : 1;
uint32_t b : 1;
uint32_t x : 1;
uint32_t i : 1;
uint32_t n : 1;
uint32_t opcode : 6;
} bits;
uint32_t res;
}BitsFormat4;
	format 4 인 Statement 일때,
	비트 값으로 저장하기 위한 변수이다.

3.7 모듈이름: reg_mnemonic_num (char *)

3.7.1. 기능

파라미터로 들어온 Register Mnemonic 을 숫자로 변환하는 함수이다.

3.7.2. 사용변수

없음

3.8 모듈이름: is_format(Statement*, int)

3.8.1. 기능

파라미터로 들어온 statement 의 format 을 확인하는 모듈이다.

ex)

is_format(&stmt, 0) : END, BYTE, WORD, RESB 등인지 검사 (DEFAULT format 이라 부르겠다)

is_format(&stmt, 1) : format 1 인지 검사

is_format(&stmt, 2) : format 2 인지 검사

is_format(&stmt, 3) : format 3/4 인지 검사

3.8.2. 사용변수

없음

3.9 모듈이름: record_stmt_for_pass2(…)

3.9.1. 기능

Statement 의 format 에 따라서 적절히 * obj_code, location_counter, line_num 등을 .obj, .list 파일에 기록한다

3.9.2. 사용변수

Statement *stmt, const int *obj_code, const int *location_counter, int *r_lc, const int *line_num, FILE *lst_fp, FILE *obj_fp, char **obj_buf, char **byte_buf, char **rec_head	함수 인자
const char *format	.obj 파일에 기록할 형식 format 을 저장하는 변수

3.10 모듈이름: handling_format_default(…)

3.10.1. 기능

Default format (WORD, BASE 등등) 의 statement 를
적절히 handling 하여 is_base, obj_code, base, b_buf 등의 값을 조정한다.

3.10.2. 사용변수

SymbolTable *symbol_table, Statement *stmt, int *obj_code, bool *is_base, int *base, char **b_buf	함수 인자

3.11 모듈이름: handling_format3(…)

3.11.1. 기능

format 3/4 의 statement 를
적절히 handling 하여 obj_code 를 조정한다.

3.11.2. 사용변수

SymbolTable *symbol_table, Statement *stmt, int *obj_code, const int *location_counter, int **location_counters, int *location_counter_cnt, int stmt_size, const bool *is_base, const int *base	함수 인자
BitsFormat3 bitsFormat3; BitsFormat4 bitsFormat4;	Statement 에 따라서 비트 값 및 flag 값을 저장하는 구조체 변수

3.12 모듈이름: handling_format2(Statement *, int *)

3.12.1. 기능

format 2 의 statement 를 적절히 handling 하여 obj_code 를 조정한다.

3.12.2. 사용변수

BitsFormat2 bits;	Statement 에 따라서 비트 값 및 flag 값을 저장하는 구조체 변수

3.13 모듈이름: handling_format1(Statement*, int*)

3.13.1. 기능

format 1 의 statement 를 적절히 handling 하여 obj_code 를 조정한다..

3.13.2. 사용변수

없음

3.14 모듈이름: tokenizing_stmt_tokens(Statement*, char*)

3.14.1. 기능

파라미터로 넘어온 char* input 을 토크나이징하고 stmt->tokens 와 stmt->token_cnt 를 조정함

3.14.2. 사용변수

없음

3.15 모듈이름: is_comment_stmt(Statement*)

3.15.1. 기능

Statement 가 주석인지 아닌지 확인함.

3.15.2. 사용변수

없음

3.16 모듈이름: mark_comment_stmt(Statement*)

3.16.1. 기능

Statement 가 주석이라면 구조체의 멤버변수(comment)에 주석이라고 설정을 한다

3.16.2. 사용변수

없음

3.17 모듈이름: is_plus_stmt(Statement *, int)

3.17.1. 기능

+JSUB 과 같이 opcode 앞에 +가 붙었는지 확인

3.17.2. 사용변수

없음

3.18 모듈이름: mark_plus_true_or_false(Statement *, int)

3.18.1. 기능

Statement 가 +JSUB 과 같이 opcode 앞에 +가 붙은 경우에 stmt->plus = true 로 설정함

3.18.2. 사용변수

없음

3.19 모듈이름: update_location_counter_by_format(Statement* ,int *)

3.19.1. 기능

Statement 의 format 에 따라서 적절히 location_counter 값을 증가시킴

3.19.2. 사용변수

없음

3.20 모듈이름: update_location_counter_by_mnemonic_name(Statement*, int*)

3.20.1. 기능

Statement 의 opcode 의 mnemonic 에 따라서 적절히 location_counter 를 증가시킴

3.20.2. 사용변수

Int len, b, cnt	Operand 의 길이 정보와 연관된 정보를 저장함
const char *operand	Operand 정보를 저장함

3.21 모듈이름: update_location_counter_by_plus_and_format(Statement *, int *)

3.21.1. 기능

Statement 가 plus 이면서 format 3/4 인경우에 location_counter 를 1 증가시킴

3.21.2. 사용변수

없음

3.22 모듈이름: is_end_condition(Statement *, FILE *)

3.22.1. 기능

pass 2 를 끝낼 시점인지 아닌지 확인함

3.22.2. 사용변수

없음

3.23 모듈이름: error_handling_pass1or2(…)

3.23.1. 기능

에러일 경우 이 함수가 실행된다.

파일들을 전부 close 하고, 파라미터로 보낸 이름의 파일들을 삭제한다.
에러가 난 Line 을 출력해준다.

3.23.2. 사용변수

Statement *stmt, FILE *fp1, FILE *fp2, FILE *fp3, char *rm_file_name1, char *rm_file_name2, char *rm_file_name3, int line_num	함수 인자

3.24 모듈이름: symbol_handling(OpcodeTable *, Statement *, char *)

3.24.1. 기능

symbol 인지 아닌지에 따라서 적절히 handling 한다.

3.24.1. 사용변수

Opcode* opc	Opcode 를 저장할 변수

3.25 모듈이름: bool read_statement(…)

3.25.1. 기능

파일로 부터 한줄을 읽어서 Statement 변수 에 적절히 초기화하여 저장한다

3.25.2. 사용변수

OpcodeTable *opcode_table, FILE *asm_fp, FILE *tmp_fp, Statement *stmt, bool is_tmp, int *location_counter, int *stmt_size	함수 인자
FILE* fp;	읽을 파일의 파일 포인터

static char raw_input[220]; static char tmp_input[200];	파일의 내용을 저장할 배열
int length = 0, offset, i	길이 변수, offset 변수, 반복문을 돌리기위한 변수
char *op_tok	토큰 값 하나를 저장할 변수

3.26 모듈이름: record_stmt_for_pass1(Statement *, FILE *, int *, int *)

3.26.1. 기능

Statement 의 종류에 따라서 적절히 tmp 파일에 기록한다

3.26.2. 사용변수

없음

3.27 모듈이름: is_end_condition(Statement *, FILE *)

3.27.1. 기능

pass 2 를 끝낼 시점인지 아닌지 확인함

3.27.2. 사용변수

없음

3.28 모듈이름: is_end_condition(Statement *, FILE *)

3.28.1. 기능

pass 2 를 끝낼 시점인지 아닌지 확인함

3.28.2. 사용변수

없음

3.29 모듈이름: Symbol 모듈의 구조체

3.29.1. 기능

3.29.2. 사용변수

typedef struct symbol {
  char label[11];
  int location_counter;
}Symbol;
	Symbol 정보 하나를 저장할 구조체

typedef struct sym_node {
  Symbol* data;
  struct sym_node* prev;
	링크드 리스트 구현을 위한 노드 구조체

struct sym_node* next; }SymNode;	
typedef struct sym_linked_list { SymNode* head; SymNode* tail; int size; }SymLinkedList;	링크드 리스트 구조체
typedef struct symbol_table { SymLinkedList* list[40]; int size; }SymbolTable;	symbol 정보들을 해시테이블 형태로 저장하기 위한 구조체.
char *op_tok	토큰 값 하나를 저장할 변수

3.30 모듈이름: construct_symbol_table()

3.30.1. 기능

Symbol_table 구조체를 할당하고 리턴하는 생성자 함수

3.30.2. 사용변수

typedef struct symbol { char label[11]; int location_counter; }Symbol;	Symbol 정보 하나를 저장할 구조체
SymbolTable* table	할당할 symbol table 변수
Int i	반복문을 위한 변수

3.31 모듈이름: destroy_symbol_table(SymbolTable**)

3.31.1. 기능

Symbol_table 구조체 변수가 할당 받은 메모리를 모두 해제한다. Symbol, SymNode 또한 이 함수에서 모두 해제한다.

3.31.2. 사용변수

SymNode *cur; SymNode *next;	해제하기위해 저장해둘 변수

SymLinkedList** list;	
int i, j	반복문을 위한 변수

3.32 모듈이름: construct_symbol_node()

3.32.1. 기능

SymNode 구조체를 할당하고 리턴하는 생성자 함수이다.

3.32.2. 사용변수

SymNode* node	할당할 SymNode 변수

3.33 모듈이름: construct_symbol()

3.33.1. 기능

Symbol 구조체를 할당하고 리턴하는 생성자 함수

3.33.2. 사용변수

Symbol* symb	할당할 Symbol 변수

3.34 모듈이름: insert_symbol(SymbolTable*, Symbol*)

3.34.1. 기능

SymbolTable 구조체의 해시테이블에 Symbol 을 추가한다.

3.34.2. 사용변수

SymNode *node	Node 구조체 변수
Int hash	Hashing 된 값을 저장하는 변수

3.35 모듈이름: find_symbol_by_name(SymbolTable*, char*)

3.35.1. 기능

SymbolTable 에서 name 으로 Symbol 을 찾는다.

3.35.2. 사용변수

int i;	반복문을 위한 변수
Int hash	Hashing 된 값을 저장하는 변수

3.36 모듈이름: print_symbols(SymbolTable*)

3.36.1. 기능

SymbolTable 에 저장된 Symbol 들을 내림차순으로 출력한다

3.36.2. 사용변수

int size
Symbol *list[1200] = {0};
int i = 0;
int num = 0;
	Table 의 사이즈를 저장하는 변수

Symbol *list[1200]	Symbol 목록을 저장하는 배열
Int I,j	반복문을 위한 변수
Int num	Symbol 개수 저장하는 변수
SymNode** cur Symbol* symb;	반복문을 돌면서 필요한 변수

3.37 모듈이름: symbol_comparator(const void *, const void *)

3.37.1. 기능

Sort 함수에 사용될 comparator 함수

3.37.2. 사용변수

Symbol *left	첫번째 인자를 cast 하여 저장한다
Symbol *right	두번째 인자를 cast 하여 저장한다

3.38 모듈이름: before_dot(char*, int)

3.38.1. 기능

문자열에서 “.” 이전의 문자열 을 찾아서 리턴한다.

예를들어 before_dot(“2_5.asm”) 은 2_5 가 리턴된다.

3.38.2. 사용변수

char *pre	리턴될 값
char* dot	. 의 인덱스를 저장한다

3.39 모듈이름: concat_n(char *, char *, int)

3.39.1. 기능

두 문자열을 합치는 concat 함수

3.39.2. 사용변수

char* res	리턴될 값
ch	. 의 인덱스를 저장한다

before_dot(char *name, int size)
symbol_comparator(const void *a, const void *b)

find_symbol_by_name
destroy_symbol_table(SymbolTable** table)

print_symbols(SymbolTable* table)

4. 전역 변수 정의

전역변수 정의 하지않았음.

5. 코드

5.1 20161631.c

```
#include "20161631.h"

int main() {
    /*
     * state_store 에서는 명령어 히스토리, 가상 메모리 영역, Opcode 정보를
저장한다.
```

```
     */
    State* state_store = construct_state();

    /*
     * 사용자가 quit(q) 명령을 입력하기전까지
     * 쉘을 통해 명령어를 입력, 수행할수있도록 한다.
     */
    command_main(state_store);

    /*
     * 동적 할당 받은 메모리를 모두 해제한다.
     */
    destroy_state(&state_store);

    return 0;
}
```

5.2 20161631.h

```
#ifndef __20161631_H__
#define __20161631_H__

#include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>
#include "command.h"
#include "state.h"
#include "dir.h"

#endif
```

5.3 command.c

```
#include "command.h"

/*
 * 사용자가 quit(q)를 명령을 입력하기 이전까지 쉘을 계속 수행한다.
 */
bool command_main(State* state_store){
    shell_status status;
```

```
    Command user_command;

    while (1){

        render_shell();

        // 사용자로 부터 입력을 받는다.
        status = read_input(&user_command.raw_command);

        // 잘못된 입력이라면 continue 한다.
        if(!exception_check_and_handling(status)) continue;

        // 입력을 지정된 명령어에 있는지 확인하고, 매핑한다.
        status = command_mapping(&user_command);
        if(!exception_check_and_handling(status)) continue;

        // 매핑된 명령어를 수행한다.
        status = command_execute(&user_command, state_store);
        if(check_quit_condition(&user_command)) break;
        if(!exception_check_and_handling(status)) continue;
        if(status == EXECUTE_FAIL) continue;

        // 실행이 완료된 명령어를 입력 그대로 히스토리에 추가한다.
        add_history(state_store, user_command.raw_command);
    }
    return true;
}

/*
 * status 파라미터에 넘어온 내용에 따라서
 * 에러에 해당한다면 적절한 에러문을 출력해주고 false 를 리턴한다.
 * 에러에 해당하지않는다면 true 를 리턴한다.
 *
 * 참고: 사용자에게 입력을 받거나, 토크나이징 하는 등의 함수들은
 *      성공, 실패 여부등에 따라서 shell_status ( enum )을 리턴한다.
 */
bool exception_check_and_handling(shell_status status){
```

```
    switch(status){
        case INPUT_READ_SUCCESS:
            break;
        case TOKENIZING_SUCCESS:
            break;
        case VALID_COMMAND_TYPE:
            break;
        case VALID_PARAMETERS:
            break;
        case EXECUTE_SUCCESS:
            break;
        case TOO_LONG_WRONG_INPUT:
            fprintf(stderr, "[ERROR] Too Long Input₩n");
            return false;
        case TOO_MANY_TOKEN:
            fprintf (stderr, "[ERROR] Too Many Tokens₩n");
            return false;
        case INVALID_COMMAND_TYPE:
            fprintf(stderr, "[ERROR] Invalid Command Type₩n");
            return false;
        case INVALID_INPUT:
            fprintf(stderr, "[ERROR] Invalid Input₩n");
            return false;
        case INVALID_PARAMETERS:
            fprintf(stderr, "[ERROR] Invalid Parameters₩n");
            return false;
        case MISSING_REQUIRE_PARAMETER:
            fprintf(stderr, "[ERROR] Missing Required Parameter₩n");
            return false;
        case EXECUTE_FAIL:
            fprintf(stderr, "[ERROR] Invalid Input₩n");
            break;
        default:
            break;
    }
    return true;
}
```

```
/*
 * quit(q) 명령이 들어왔다면 true 를 리턴하고,
 * 아니라면 false 를 리턴한다.
 */
bool check_quit_condition(Command* user_command){
    if(user_command->type == TYPE_QUIT){
        assert(user_command->token_cnt == 1);
        return true;
    }
    else{
        return false;
    }
}
```

5.4 command.h

```
#ifndef __COMMAND_H__
#define __COMMAND_H__


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <stdbool.h>

#include "command_macro.h"
#include "command_objects.h"
#include "command_shell.h"
#include "command_mapping.h"
#include "command_execute.h"
#include "state.h"

/*
 * 사용자가 quit(q)를 명령을 입력하기 이전까지 쉘을 계속 수행한다.
 */
bool command_main(State* state_store);
```

```
/*
 * status 파라미터에 넘어온 내용에 따라서
 * 에러에 해당한다면 적절한 에러문을 출력해주고 false 를 리턴한다.
 * 에러에 해당하지않는다면 true 를 리턴한다.
 *
 * 참고: 사용자에게 입력을 받거나, 토크나이징 하는 등의 함수들은
 *      성공, 실패 여부등에 따라서 shell_status ( enum )을 리턴한다.
 */
bool exception_check_and_handling(shell_status status);

/*
 * quit(q) 명령이 들어왔다면 true 를 리턴하고,
 * 아니라면 false 를 리턴한다.
 */
bool check_quit_condition(Command* user_command);

#endif
```

5.5 command_execute.c

```
#include "command_execute.h"

/*
 * 사용자가 입력한 명령어에 따른 실행 함수(execute_***())를 실행한다.
 * 또한 실행된 결과를 리턴해준다.
 */
shell_status command_execute(Command *user_command, State *state_store) {
    assert(user_command);
    assert(state_store);

    switch (user_command->type){
        case TYPE_HELP:
            execute_help();
            return EXECUTE_SUCCESS;
        case TYPE_HISTORY:
            return execute_history(state_store, user_command->raw_command);
        case TYPE_QUIT:
            return execute_quit();
        case TYPE_DIR:
```

```
            return execute_dir();
        case TYPE_EDIT:
            return execute_edit(user_command, state_store->memories_state);
        case TYPE_FILL:
            return execute_fill(user_command, state_store->memories_state);
        case TYPE_RESET:
            return execute_reset(state_store->memories_state);
        case TYPE_OPCODE:
            return execute_opcode(user_command, state_store);
        case TYPE_OPCODELIST:
            return execute_opcodelist(state_store);
        case TYPE_DUMP:
            return execute_dump(user_command, state_store->memories_state);
        case TYPE_ASSEMBLE:
            return execute_assemble(user_command, state_store);
        case TYPE_TYPE:
            return execute_type(user_command);
        case TYPE_SYMBOL:
            return execute_symbol(state_store);
        default:
            break;
    }
    return EXECUTE_SUCCESS;
}

/*
 * history 명령어
 * 실행되었던 명령어 히스토리를 출력한다
 */
shell_status execute_history(State* state_store, char *last_command) {
    assert(state_store);
    assert(last_command);

    print_histories_state(state_store, last_command);

    return EXECUTE_SUCCESS;
}
```

```
/*
 * help 명령어
 * 사용할수있는 명령어들을 화면에 출력해서 보여준다,
 */
void execute_help(){
    fprintf (stdout,"h[elp]₩n"
            "d[ir]₩n"
            "q[uit]₩n"
            "hi[story]₩n"
            "du[mp] [start, end]₩n"
            "e[dit] address, value₩n"
            "f[ill] start, end, value₩n"
            "reset₩n"
            "opcode mnemonic₩n"
            "opcodelist₩n"
            "assemble filename₩n"
            "type filename₩n"
            "symbol₩n");
}

/*
 * quit 명령어
 * QUIT 이라는 shell_status(enum)을 리턴한다.
 *
 * 참고: command_main 함수의 무한 루프는 QUIT 이라는 status 가 들어오면
 *       break 되도록 설계되었음.
 */
shell_status execute_quit(){
    fprintf(stdout, "Bye :)₩n");

    return QUIT;
}

/*
 * dir 명령어
 * 실행 파일이 위치한 폴더에 있는 파일들과 폴더들을 출력한다.
```

```
 */
shell_status execute_dir(){
    if(!print_dir()) return EXECUTE_FAIL;
    return EXECUTE_SUCCESS;
}

/*
 * dump 명령어
 * dump start, end : start~end 까지의 가상 메모리영역을 출력한다.
 * dump start      : start 메모리 부터 10 라인의 영역을 출력한다.
 * dump            : 가장 마지막으로 실행되었던 메모리부터 10 라인의 영역 출력한다.
 */
shell_status execute_dump(Command *user_command, Memories
*memories_state) {
    assert(memories_state);
    assert(user_command);
    assert(user_command->token_cnt < 4);

    size_t token_cnt = user_command->token_cnt;
    int start=0, end=0;

    // 출력할 메모리 영역의 범위를 초기화 한다.
    if(token_cnt == 1){
        // case1. dump

        start = memories_state->last_idx + 1;
        end = start + 159;
    } else if(token_cnt == 2){
        // case2. dump start

        start = (int)strtol(user_command->tokens[1], NULL, 16);
        if(start + 159 >= MEMORIES_SIZE) end = MEMORIES_SIZE - 1;
        else end = start + 159;
    } else if(token_cnt == 3){
        // case3. dump start, end

        start = (int)strtol(user_command->tokens[1], NULL, 16);
```

```
        end = (int)strtol(user_command->tokens[2], NULL, 16);
        if(end >= MEMORIES_SIZE) end = MEMORIES_SIZE - 1;
    }

    // start ~ end 범위의 메모리 영역을 출력한다.
    print_memories(memories_state, start, end);

    // 출력된 마지막 메모리 주소를 저장한다.
    // 만일 마지막 메모리 주소에 1 을 더한 주소가 범위를 벗어났다면 -1 을 저장한다.
    if(end + 1 >= MEMORIES_SIZE)
        memories_state->last_idx = -1;
    else
        memories_state->last_idx = end;

    return EXECUTE_SUCCESS;
}

/*
 * edit 명령어
 * edit addr, value: addr 주소의 값을 value 로 수정한다.
 */
shell_status execute_edit(Command *user_command, Memories *memories_state)
{
    assert(user_command);
    assert(memories_state);
    assert(user_command->token_cnt == 3);

    int addr = (int)strtol(user_command->tokens[1], NULL, 16);
    short value = (short)strtol(user_command->tokens[2], NULL, 16);

    edit_memory(memories_state, addr, value);

    return EXECUTE_SUCCESS;
}

/*
 * fill 명령어
```

```
 * fill start, end, value: start~end 의 메모리 영역의 값들을 value 로 수정한다.
 */
shell_status execute_fill(Command *user_command, Memories *memories_state) {
    assert(user_command);
    assert(memories_state);
    assert(user_command->token_cnt == 4);
    int start = (int)strtol(user_command->tokens[1], NULL, 16);
    int end = (int)strtol(user_command->tokens[2], NULL, 16);
    short value = (short)strtol(user_command->tokens[3], NULL, 16);
    int addr = 0;
    assert(start <= end || start >= 0 || end >= 0 || value >= 0);

    for(addr = start; addr <= end; addr++)
        edit_memory(memories_state, addr, value);

    return EXECUTE_SUCCESS;
}

/*
 * reset 명령어
 * 가상 메모리 영역의 모든 값들을 0 으로 바꾼다.
 */
shell_status execute_reset(Memories *memories_state) {
    assert(memories_state);

    int addr = 0, end = MEMORIES_SIZE - 1;
    for(addr=0;addr<=end;addr++)
        edit_memory(memories_state, addr, 0);

    return EXECUTE_SUCCESS;
}

/*
 * opcode 명령어
 * opcode mnemonic : mnemonic 의 value 를 출력
 * ex) opcode LDF => opcode is 70
 */
```

```
shell_status execute_opcode(Command* user_command, State* state_store){
    assert(user_command->token_cnt == 2);
    Opcode* opc = find_opcode_by_name(state_store->opcode_table_state,
user_command->tokens[1]);

    if(!opc) return EXECUTE_FAIL;

    fprintf(stdout, "opcode is %X₩n", opc->value);

    return EXECUTE_SUCCESS;
}

/*
 * opcodelist 명령어
 * 해시테이블 형태로 저장된 opcode 목록을 출력해준다.
 */
shell_status execute_opcodelist(State* state_store){
    print_opcodes(state_store->opcode_table_state);
    return EXECUTE_SUCCESS;
}

/*
 *  assemble 명령어
 *  ex.
 *      assemble filename
 */
shell_status execute_assemble(Command *user_command, State* state_store){
    assert(user_command->token_cnt == 2);

    if(assemble_file(state_store, user_command->tokens[1]))
        return EXECUTE_SUCCESS;
    else
        return EXECUTE_FAIL;
}

shell_status execute_type(Command* user_command){
    assert(user_command->token_cnt == 2);
```

```
    if(!user_command->tokens[1])
        return EXECUTE_FAIL;
    FILE* fp = fopen(user_command->tokens[1], "rt");
    char buf[10000];
    if(!fp){
        fprintf(stderr, "[ERROR] Can't Open File₩n");
        return EXECUTE_FAIL;
    }
    while (fgets (buf, sizeof(buf), fp))
        fputs (buf, stdout);
    fputs("₩n", stdout);

    fclose(fp);
    return EXECUTE_SUCCESS;
}

shell_status execute_symbol(State *state_store) {
    if(!state_store->is_symbol_table) return EXECUTE_FAIL;

    print_symbols(state_store->symbol_table_state);
    return EXECUTE_SUCCESS;
}
```

5.6 command_execute.h

```
#ifndef __COMMAND_EXECUTE_H__
#define __COMMAND_EXECUTE_H__

#include "command_macro.h"
#include "command_objects.h"
#include "state.h"
#include "dir.h"

/*
 * 사용자가 입력한 명령어에 따른 실행 함수(execute_***())를 실행한다.
 * 또한 실행된 결과를 리턴해준다.
 */
shell_status command_execute(Command *user_command, State *state_store);
```

```
/*
 * help 명령어
 * 사용할수있는 명령어들을 화면에 출력해서 보여준다,
 */
void execute_help();

/*
 * history 명령어
 * 실행되었던 명령어 히스토리를 출력한다
 */
shell_status execute_history(State* state_store, char *last_command);

/*
 * quit 명령어
 * QUIT 이라는 shell_status(enum)을 리턴한다.
 *
 * 참고: command_main 함수의 무한 루프는 QUIT 이라는 status 가 들어오면
 *      break 되도록 설계되었음.
 */
shell_status execute_quit();

/*
 * dir 명령어
 * 실행 파일이 위치한 폴더에 있는 파일들과 폴더들을 출력한다.
 */
shell_status execute_dir();

/*
 * dump 명령어
 * dump start, end : start~end 까지의 가상 메모리영역을 출력한다.
 * dump start      : start 메모리 부터 10 라인의 영역을 출력한다.
 * dump            : 가장 마지막으로 실행되었던 메모리부터 10 라인의 영역 출력한다.
 */
shell_status execute_dump(Command *user_command, Memories
*memories_state);

/*
```

```
 * edit 명령어
 * edit addr, value: addr 주소의 값을 value 로 수정한다.
 */
shell_status execute_edit(Command *user_command, Memories *memories_state);

/*
 * fill 명령어
 * fill start, end, value: start~end 의 메모리 영역의 값들을 value 로 수정한다.
 */
shell_status execute_fill(Command *user_command, Memories *memories_state);

/*
 * reset 명령어
 * 가상 메모리 영역의 모든 값들을 0 으로 바꾼다.
 */
shell_status execute_reset(Memories *memories_state);

/*
 * opcode 명령어
 * opcode mnemonic : mnemonic 의 value 를 출력
 * ex) opcode LDF => opcode is 70
 */
shell_status execute_opcode(Command *user_command, State* state_store);

/*
 * opcodelist 명령어
 * 해시테이블 형태로 저장된 opcode 목록을 출력해준다.
 */
shell_status execute_opcodelist(State* state_store);

/*
 *  assemble 명령어
 */
shell_status execute_assemble(Command *user_command, State* state_store);

/*
 *  type 명령어
```

```
 */
shell_status execute_type(Command* user_command);

/*
 * symbol 명령어
 */
shell_status execute_symbol(State *state_store);

#endif
```

5.7 command_macro.h

```
#ifndef __COMMAND_MACRO_H__
#define __COMMAND_MACRO_H__

#define COMPARE_STRING(T, S) (strcmp ((T), (S)) == 0)
#define COMMAND_MAX_LEN 510
#define TOKEN_MAX_NUM 30

#endif
```

5.8 command_mapping.c

```
#include "command_mapping.h"

/*
 * 사용자의 raw_input 이 적절한 명령어인지 확인하고,
 * 적절하다면 지정된 명령어로 매핑하고, 성공했다는 shell_status 를 리턴한다.
 * 적절하지않다면, 실패했다는 shell_status 를 리턴한다.
 */
shell_status command_mapping(Command* user_command){
    assert(user_command);
    shell_status status;

    status = tokenizing(user_command);
    if(status != TOKENIZING_SUCCESS) return status;

    status = command_mapping_type(user_command);
    if(!validate_tokenizing(user_command->raw_command,
                        (int) user_command->token_cnt,
                        TOKEN_MAX_NUM))
```

```
        return INVALID_INPUT;
    if(status != VALID_COMMAND_TYPE) return status;

    // 파라미터가 해당 명령어에 적절한지 확인한다.
    // 예를들어 dump -1 과 같은 경우를 잡아낸다.
    status = validate_parameters(user_command);
    if(status != VALID_PARAMETERS) return status;

    return status;
}

/*
 * 사용자의 raw_input 을 토크나이징하여 user_command->tokens 에 저장한다.
 * ex) 입력이 dump 1, 2 가 들어왔다면
 *     tokens[0] = "dump"
 *     tokens[1] = "1"
 *     tokens[2] = "2"
 *     형태로 저장된다.
 *
 * @return TOKENIZING_SUCCESS or INVALID_INPUT
 */
shell_status tokenizing(Command* user_command){
    assert(user_command);
    static char raw_command[COMMAND_MAX_LEN];

    strncpy (raw_command, user_command->raw_command,
COMMAND_MAX_LEN);
    user_command->token_cnt = 0;
    user_command->tokens[user_command->token_cnt] = strtok(raw_command,
" ,₩t₩n");
    while (user_command->token_cnt <= TOKEN_MAX_NUM && user_command-
>tokens[user_command->token_cnt])
        user_command->tokens[++user_command->token_cnt] = strtok (NULL,
" ,₩t₩n");

    if(!user_command->tokens[0]) return INVALID_INPUT;
```

```
    return TOKENIZING_SUCCESS;
}

/*
 * user_command->tokens[0]에 저장된 문자열이
 * 적절한 명령어 타입인지 확인하고, user_command->type 에 명령어 타입을
설정해준다.
 *
 * @return VALID_COMMAND_TYPE or INVALID_COMMAND_TYPE
 */
shell_status command_mapping_type(Command *user_command){
    assert(user_command);
    char* first_token = user_command->tokens[0];
    if(COMPARE_STRING(first_token, "h") ||
       COMPARE_STRING(first_token, "help")) {
        user_command->type = TYPE_HELP;
    }else if(COMPARE_STRING(first_token, "d") ||
            COMPARE_STRING(first_token, "dir")) {
        user_command->type = TYPE_DIR;
    }else if(COMPARE_STRING(first_token, "q") ||
            COMPARE_STRING(first_token, "quit")) {
        user_command->type = TYPE_QUIT;
    } else if(COMPARE_STRING(first_token, "hi") ||
             COMPARE_STRING(first_token, "history")){
        user_command->type = TYPE_HISTORY;
    } else if(COMPARE_STRING(first_token, "du") ||
             COMPARE_STRING(first_token, "dump")){
        user_command->type = TYPE_DUMP;
    } else if(COMPARE_STRING(first_token, "e") ||
             COMPARE_STRING(first_token, "edit")){
        user_command->type = TYPE_EDIT;
    } else if(COMPARE_STRING(first_token, "f") ||
             COMPARE_STRING(first_token, "fill")){
        user_command->type = TYPE_FILL;
    } else if(COMPARE_STRING(first_token, "reset")){
        user_command->type = TYPE_RESET;
    } else if(COMPARE_STRING(first_token, "opcode")){
```

```
        user_command->type = TYPE_OPCODE;
    } else if(COMPARE_STRING(first_token, "opcodelist")){
        user_command->type = TYPE_OPCODELIST;
    } else if(COMPARE_STRING(first_token, "assemble")){
        user_command->type = TYPE_ASSEMBLE;
    } else if(COMPARE_STRING(first_token, "type")){
        user_command->type = TYPE_TYPE;
    } else if(COMPARE_STRING(first_token, "symbol")){
        user_command->type = TYPE_SYMBOL;
    } else {
        return INVALID_COMMAND_TYPE;
    }
    return VALID_COMMAND_TYPE;
}
```

5.9 command_mapping.h

```
#ifndef __COMMAND_MAPPING_H__
#define __COMMAND_MAPPING_H__

#include "command_macro.h"
#include "command_objects.h"
#include "command_validate_util.h"
#include "util.h"
#include <string.h>
#include <stdbool.h>
#include <assert.h>
#include <sys/types.h>
#include <stdlib.h>

/*
 * 사용자의 raw_input 이 적절한 명령어인지 확인하고,
 * 적절하다면 지정된 명령어로 매핑하고, 성공했다는 shell_status 를 리턴한다.
 * 적절하지않다면, 실패했다는 shell_status 를 리턴한다.
 */
shell_status command_mapping(Command* user_command);

/*
 * 사용자의 raw_input 을 토크나이징하여 user_command->tokens 에 저장한다.
```

```
 * ex) 입력이 dump 1, 2 가 들어왔다면
 *     tokens[0] = "dump"
 *     tokens[1] = "1"
 *     tokens[2] = "2"
 *     형태로 저장된다.
 *
 * @return TOKENIZING_SUCCESS or INVALID_INPUT
 */
shell_status tokenizing(Command* user_command);

/*
 * user_command->tokens[0]에 저장된 문자열이
 * 적절한 명령어 타입인지 확인하고, user_command->type 에 명령어 타입을
설정해준다.
 *
 * @return VALID_COMMAND_TYPE or INVALID_COMMAND_TYPE
 */
shell_status command_mapping_type(Command *user_command);

#endif
```

5.10 command_objects.h

```
#ifndef __COMMAND_OBJECTS_H__
#define __COMMAND_OBJECTS_H__
#include <sys/types.h>
#define TOKEN_MAX_NUM 30

// 명령어 타입을 enum 형태로 표현한다.
enum shell_command_type{
    TYPE_HELP, TYPE_DIR, TYPE_QUIT, TYPE_HISTORY,
    TYPE_DUMP, TYPE_EDIT, TYPE_FILL, TYPE_RESET,
    TYPE_OPCODE, TYPE_OPCODELIST
};

// command 를 구조체로 구조화 하여 표현.
typedef struct command {
    char* raw_command;
    char* tokens[TOKEN_MAX_NUM + 5];
```

```
    size_t token_cnt;
    enum shell_command_type type;
} Command;

// shell 의 상태를 표현함.
typedef enum SHELL_STATUS {
    INPUT_READ_SUCCESS, TOO_LONG_WRONG_INPUT,
    TOKENIZING_SUCCESS, TOO_MANY_TOKEN,
    INVALID_INPUT, VALID_COMMAND, INVALID_COMMAND,
    INVALID_COMMAND_TYPE, VALID_COMMAND_TYPE,
    INVALID_PARAMETERS, VALID_PARAMETERS,
    MISSING_REQUIRE_PARAMETER, EXECUTE_SUCCESS, QUIT, EXECUTE_FAIL
} shell_status;

#endif
```

5.11 command_shell.c

```
#include "command_shell.h"

/*
 * 사용자로 부터 입력을 받고 *target 에 저장한다.
 * 입력이 너무 길 경우 에러를 리턴해준다.
 *
 * @return INPUT_READ_SUCCESS or TOO_LONG_WRONG_INPUT
 */
shell_status read_input(char** target){
    static char input[COMMAND_MAX_LEN + 10];
    fgets(input, COMMAND_MAX_LEN + 10, stdin);
    if(strlen(input) >= COMMAND_MAX_LEN) return TOO_LONG_WRONG_INPUT;
    *target = input;
    return INPUT_READ_SUCCESS;
}

/*
 * shell 을 출력함.
 */
void render_shell(){
```

```
    printf("sicsim > ");
}
```

5.12 command_shell.h

```
#ifndef __COMMAND_SHELL_H__
#define __COMMAND_SHELL_H__

#include "command_macro.h"
#include "command_objects.h"
#include <string.h>
#include <stdio.h>

/*
 * 사용자로 부터 입력을 받고 *target 에 저장한다.
 * 입력이 너무 길 경우 에러를 리턴해준다.
 *
 * @return INPUT_READ_SUCCESS or TOO_LONG_WRONG_INPUT
 */
shell_status read_input(char** target);

/*
 * shell 을 출력함.
 */
void render_shell();

#endif
```

5.13 command_validate_util.c

```
#include "command_validate_util.h"

/*
 * 토크나이징이 적절하게 되었는지 검증한다.
 * 예를들어 du , 1 1 과 같이
 * 파라미터 사이에 콤마가 없거나 이상한 위치에 콤마가 있는 등의 문제를 잡아낸다.
 *
 * @return true or false
 */
bool validate_tokenizing(char *str, int token_cnt, int max_token_num) {
    assert(str);
```

```
int length = (int)strlen(str);
int i=0, comma_cnt = 0, flag = 0;
char cm;

// 토큰의 개수를 검증한다.
if(token_cnt > max_token_num)
    return false;
if(token_cnt <= 0)
    return false;

// 콤마의 개수를 계산한다.
for(i=0;i<length; i++){
    if(str[i] == ',')
        comma_cnt++;
}

// 토큰의 개수가 두개이면서 콤마가 없는 경우는 적절한 경우다.
if(token_cnt <= 2 && comma_cnt == 0)
    return true;

// 예시와 같은 경우의 에러를 잡아낸다.
// ex)   du , 1 1 [x]
// ex)   , du
flag = 0;
for(i=0;i<length;i++){
    cm = str[i];
    if(flag == 0){
        // 첫번째 토큰 이전 문자열.
        if(cm == ',') return false;
        if(cm != ' ' && cm != '₩t') flag = 1;
        continue;
    } else if(flag == 1){
        // 첫번째 토큰을 지나가는 중
     if(cm == ',') return false;
        if(cm == ' ' || cm == '₩t') flag = 2;
        continue;
    }
```

```
    // 첫번째 토큰과 두번째 토큰 사이

  if(str[i] == ' ') continue;
    if(str[i] == '₩t') continue;
    if(str[i] != ',') break;
    else return false;
}

// 예시와 같은 경우의 에러를 잡아낸다.
// ex) du 1 1 , [x]
for(i=length-2;i>=0;i--){
    if(str[i] == ' ') continue;
    if(str[i] == '₩t') continue;
    if(str[i] != ',') break;
    else return false;
}

// 토큰의 개수와 콤마의 개수를 비교한다.
if(token_cnt != comma_cnt + 2)
    return false;


// ex) f 1 ,, 2 3 [x]
// , 다음에는 [ ] [₩t]이 나오다가 [ ] [₩t] [,]이 아닌 값이 나와야한다.
flag = 0; // 콤마가 나오면 flag 는 1 로 놓자.
for(i=0;i<length;i++) {
    cm = str[i];
    if (flag == 1) {
        if (cm == ',') return false;
        if (cm == ' ' || cm == '₩t') continue;
        flag = 0;
        continue;
    }
    if (cm == ',') {
        flag = 1;
        continue;
    }
```

```
    }

    return true;
}

/*
 * 사용자가 입력한 파라미터가 적절한 파라미터 값인지 검증한다.
 * (명령어에 따른 파라미터 개수, 크기, 범위 등)
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_parameters(Command *user_command){
    assert(user_command);
    if((user_command->type == TYPE_QUIT ||
        user_command->type == TYPE_HELP ||
        user_command->type == TYPE_HISTORY ||
        user_command->type == TYPE_DIR ||
        user_command->type == TYPE_RESET ||
        user_command->type == TYPE_OPCODELIST ||
        user_command->type == TYPE_SYMBOL
       ) &&
       user_command->token_cnt > 1)
        return INVALID_PARAMETERS;
    if(user_command->type == TYPE_TYPE && user_command->token_cnt != 2)
        return INVALID_PARAMETERS;
    if(user_command->type == TYPE_OPCODE)
        return validate_opcode_parameters(user_command);
    if(user_command->type == TYPE_EDIT)
        return validate_edit_parameters(user_command);
    if(user_command->type == TYPE_FILL)
        return validate_fill_parameters(user_command);
    if(user_command->type == TYPE_DUMP)
        return validate_dump_parameters(user_command);
    if(user_command->type == TYPE_ASSEMBLE)
        return validate_assemble_parameters(user_command);
    return VALID_PARAMETERS;
}
```

```
/*
 * dump 명령어의 파라미터를 검증한다.
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_dump_parameters(Command *user_command){
    assert(user_command);
    assert(user_command->type == TYPE_DUMP);
    int tok1, tok2;
    if(user_command->token_cnt > 1) {

        // 적절한 주소값인지 확인한다.
        if (!is_valid_address(user_command->tokens[1], MB))
            return INVALID_PARAMETERS;
        else if (user_command->token_cnt == 2)
            return VALID_PARAMETERS;

        // 적절한 주소값인지 확인한다.
        if (!is_valid_address(user_command->tokens[2], MB))
            return INVALID_PARAMETERS;

        tok1 = (int) strtol(user_command->tokens[1], NULL, 16);
        tok2 = (int) strtol(user_command->tokens[2], NULL, 16);

        // invalid area
        if (tok1 > tok2) return INVALID_PARAMETERS;
    }

    // dump 1, 2, 3 과 같이 파라미터가 세개 이상이 되는 경우는 에러이다.
    if(user_command->token_cnt > 3)
        return INVALID_PARAMETERS;

    return VALID_PARAMETERS;
}

/*
```

```
 * opcode 명령어의 파라미터를 검증한다.
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_opcode_parameters(Command *user_command){
    assert(user_command);
    assert(user_command->type == TYPE_OPCODE);
    if(user_command->token_cnt != 2) return INVALID_PARAMETERS;
    if(strlen(user_command->tokens[1]) > 14) return INVALID_PARAMETERS;

    return VALID_PARAMETERS;
}

/*
 * edit 명령어의 파라미터를 검증한다.
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_edit_parameters(Command *user_command){
    assert(user_command);
    assert(user_command->type == TYPE_EDIT);
    int value;

    if(user_command->token_cnt != 3)
        return INVALID_PARAMETERS;

    // 적절한 주소값인지 검증
   if(!is_valid_address(user_command->tokens[1], MB))
        return INVALID_PARAMETERS;

    // 적절한 16 진수 값인지 검증한다.
    if(!is_valid_hex(user_command->tokens[2]))
        return INVALID_PARAMETERS;

    value = (int) strtol(user_command->tokens[2], NULL, 16);
    // 범위 확인
   if(!(0 <= value && value <= 0xFF))
```

```
        return INVALID_PARAMETERS;

    return VALID_PARAMETERS;
}

/*
 * fill 명령어의 파라미터를 검증한다.
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_fill_parameters(Command *user_command){
    assert(user_command);
    assert(user_command->type == TYPE_FILL);
    int value, start, end;

    if(user_command->token_cnt != 4)
        return INVALID_PARAMETERS;
    if(!is_valid_address(user_command->tokens[1], MB))
        return INVALID_PARAMETERS;
    if(!is_valid_address(user_command->tokens[2], MB))
        return INVALID_PARAMETERS;

    start = (int) strtol(user_command->tokens[1], NULL, 16);
    end = (int) strtol(user_command->tokens[2], NULL, 16);
    if (start > end) return INVALID_PARAMETERS;

    value = (int) strtol(user_command->tokens[3], NULL, 16);
    if(!is_valid_hex(user_command->tokens[3]))
        return INVALID_PARAMETERS;
    if(!(0 <= value && value <= 0xFF)) return INVALID_PARAMETERS;

    return VALID_PARAMETERS;
}

/*
 * assemble 명령어의 파라미터를 검증한다.
 *
```

```
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_assemble_parameters(Command *user_command){
    assert(user_command);
    assert(user_command->type == TYPE_ASSEMBLE);

    if(user_command->token_cnt != 2)
        return INVALID_PARAMETERS;

    return VALID_PARAMETERS;
}
```

5.14 command_validate_util.h

```
#ifndef __COMMAND_VALIDATE_UTIL_H__
#define __COMMAND_VALIDATE_UTIL_H__

#include <stdbool.h>
#include <string.h>
#include <assert.h>
#include "command_objects.h"
#include "util.h"
#define MB (1024*1024)

/*
 * 토크나이징이 적절하게 되었는지 검증한다.
 * 예를들어 du , 1 1 과 같이
 * 파라미터 사이에 콤마가 없거나 이상한 위치에 콤마가 있는 등의 문제를 잡아낸다.
 *
 * @return true or false
 */
bool validate_tokenizing(char *str, int token_cnt, int max_token_num);

/*
 * 사용자가 입력한 파라미터가 적절한 파라미터 값인지 검증한다.
 * (명령어에 따른 파라미터 개수, 크기, 범위 등)
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
```

```
shell_status validate_parameters(Command *user_command);

/*
 * dump 명령어의 파라미터를 검증한다.
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_dump_parameters(Command *user_command);

/*
 * opcode 명령어의 파라미터를 검증한다.
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_opcode_parameters(Command *user_command);

/*
 * edit 명령어의 파라미터를 검증한다.
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_edit_parameters(Command *user_command);

/*
 * fill 명령어의 파라미터를 검증한다.
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_fill_parameters(Command *user_command);

shell_status validate_assemble_parameters(Command *user_command);
#endif
```

5.15 dir.c

```
#include "dir.h"

/*
 * 실행 파일이 위치한 폴더에 있는 파일들과 폴더들을 출력한다.
```

```
 */
bool print_dir(){
    DIR* dir = opendir(".");
    struct dirent *ent;
    struct stat stat;
    char* ent_dname;
    char* format;
    char path[1025];
    int i = 0;

    if(!dir){
        fprintf(stderr, "[ERROR] Can't open directory");
        return false;
    }
    ent = readdir(dir);
    while (ent){
        ent_dname = ent->d_name;
        lstat(ent_dname, &stat);

        if(S_ISDIR(stat.st_mode)) format = "%s/";
        else if(S_IXUSR & stat.st_mode) format = "%s*";
        else format = "%s ";

        sprintf(path, format, ent->d_name);
        printf("%-20s", path);

        if(++i % 5 == 0) printf("₩n");
        ent = readdir(dir);
    }
    if (i % 5 != 0) printf ("₩n");
    closedir(dir);
    return true;
}
```

5.16 dir.h

```
#ifndef __DIR_H__
#define __DIR_H__
```

```
#include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>

/*
 * 실행 파일이 위치한 폴더에 있는 파일들과 폴더들을 출력한다.
 */
bool print_dir();

#endif
```

5.17 history.c

```
#include "history.h"

/*
 * Histories 구조체를 생성(할당)하고 HistoryList 도 생성(할당)한다.
 */
Histories* construct_histories(){
    Histories* hists = (Histories*)malloc(sizeof(*hists));

    hists->list = (HistoryList*)malloc(sizeof(HistoryList));
    hists->list->head = (HistoryNode*)malloc(sizeof(HistoryNode));
    hists->list->tail = (HistoryNode*)malloc(sizeof(HistoryNode));
    hists->list->head->prev = NULL;
    hists->list->head->next = hists->list->tail;
    hists->list->tail->prev = hists->list->head;
    hists->list->tail->next = NULL;

    hists->list->head->data = construct_history();
    hists->list->tail->data = construct_history();

    hists->size = 0;
    return hists;
}
```

```
/*
 * History 구조체를 생성(할당)한다.
 */
History* construct_history(){
    History* hist = (History*)malloc(sizeof(*hist));

    return hist;
}

/*
 * Histories 구조체를 생성하면서 할당했던 모든 메모리를 해제한다.
 */
bool destroy_histories(Histories **histories_state){
    HistoryNode *cur;
    int i;
    cur = ((*histories_state)->list->head);
    for(i=0;i<(*histories_state)->size + 1;i++){
        cur = (*histories_state)->list->head;
        (*histories_state)->list->head = (*histories_state)->list->head->next;
        free(cur->data);
        free(cur);
    }

    free((*histories_state)->list->tail->data);
    free((*histories_state)->list->tail);

    free((*histories_state)->list);

    free((*histories_state));

    return true;
}

/*
 * History 구조체를 생성(할당)하고 value 에 str 문자열을 저장한다.
 */
```

```
History* construct_history_with_string(char* str){
    History* hist = construct_history();
    strncpy(hist->value, str, HISTORY_MAX_LEN);

    return hist;
}

/*
 * Histories 에 target History 구조체를 저장한다.
 */
bool push_history(Histories* histories_store, History* target){
    assert(histories_store);
    assert(target);
    assert(histories_store->list);
    assert(histories_store->list->head);
    assert(histories_store->list->tail);

    HistoryNode* hist_node = (HistoryNode*)malloc(sizeof(HistoryNode));
    hist_node->data = target;

    hist_node->prev = histories_store->list->tail->prev;
    hist_node->next = histories_store->list->tail;
    histories_store->list->tail->prev->next = hist_node;
    histories_store->list->tail->prev = hist_node;
    histories_store->list->size += 1;

    histories_store->size += 1;
    return true;
}

/*
 * Histories 에 저장된 명령어 히스토리를 출력한다.
 */
void print_history(Histories *histories_store, char *last_command) {
    assert(histories_store);
    assert(last_command);
    assert(histories_store->list);
```

```
    assert(histories_store->list->head);
    assert(histories_store->list->tail);

    HistoryNode** cur = &histories_store->list->head;
    int i = 0;
    for(i=0;i<histories_store->size + 1;i++){
        if(i == 0){
            cur = &((*cur)->next);
            continue;
        }
        printf("%-4d %s", i, (char*)(*cur)->data);
        cur = &((*cur)->next);

    }
    printf("%-4d %s", i, last_command);
    printf("₩n");
}
```

5.18 history.h

```
#ifndef __HISTORY_H__
#define __HISTORY_H__
#define HISTORY_MAX_LEN 501
#include "util.h"
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>

/*
 * 히스토리에 저장될 하나의 명령어 문자열을
 * history 구조체에 wrapping 해서 저장하게된다.
 */
typedef struct history {
    char value[HISTORY_MAX_LEN];
} History;

/*
```

```
 * history_list 의 Node 역활을 한다.
 */
typedef struct history_node{
    History* data;
    struct history_node* prev;
    struct history_node* next;
} HistoryNode;

/*
 * 명령어 히스토리를 링크드 리스트 형태로 저장하는 구조체.
 */
typedef struct history_list {
    struct history_node* head;
    struct history_node* tail;
    int size;
} HistoryList;

/*
 * 히스토리에 담긴 명령어의 개수를 저장 하고
 * 실질적으로 히스토리를 저장하는 history_list* 를 멤버로 갖고있다.
 * 일종의 wrapper struct 이다.
 */
typedef struct histories {
    int size;
    HistoryList* list;
} Histories;

/*
 * Histories 구조체를 생성(할당)하고 HistoryList 도 생성(할당)한다.
 */
Histories* construct_histories();

/*
 * History 구조체를 생성(할당)한다.
 */
History* construct_history();
```

```
/*
 * History 구조체를 생성(할당)하고 value 에 str 문자열을 저장한다.
 */
History* construct_history_with_string(char* str);

/*
 * Histories 에 target History 구조체를 저장한다.
 */
bool push_history(Histories* histories_store, History* target);

/*
 * Histories 에 저장된 명령어 히스토리를 출력한다.
 */
void print_history(Histories *histories_store, char *last_command);

/*
 * Histories 구조체를 생성하면서 할당했던 모든 메모리를 해제한다.
 */
bool destroy_histories(Histories **histories_state);

#endif
```

5.19 memory.c

```
#include "memory.h"

/*
 * Memories 구조체를 생성(할당)한다.
 */
Memories* construct_memories(){
    Memories* virtual_memories = (Memories*)malloc(sizeof(*virtual_memories));
    virtual_memories->last_idx = -1;

    return virtual_memories;
}

/*
 * Memories 구조체를 생성하면서 할당했던 메모리를 해제한다.
```

```
 */
bool destroy_memories(Memories** memories_state){
    free(*memories_state);
    return true;
}

/*
 * start ~ end 메모리 영역을 출력한다.
 * start 와 end 는 10 진수로 변환한 값이다.
 */
void print_memories(Memories* memories_state, int start, int end){
    assert(memories_state);
    assert(start >= 0);
    assert(end >= 0);
    assert(start <= end);
    assert(start < MEMORIES_SIZE);
    assert(end < MEMORIES_SIZE);

    int start_row = (start / 16)*16;
    // ex, [dump 11] 에서 start 는 17 이 되고, start_row 는 16 이 됨.
    // ex, [dump AA] 에서 start 는 170 이 되고, start_row 는 160 이 됨. ( 10 ==
0xA0)

    int end_row = (end / 16)*16;
    int y, x, n;
    short value;
    for(y=start_row;y<=end_row;y+=16){
        printf("%05X ", y);
        for(x=0;x<16;x++){
            n = y + x;
            if(n < start || n > end) printf("   ");
            else printf("%02X ", memories_state->data[n].value);
        }
        printf("; ");
        for(x=0;x<16;x++){
            n = y + x;
            if(n < start || n > end) printf(".");
```

```
            else {
                value = memories_state->data[n].value;
                if(0x20 <= value && value <= 0x7E) printf("%c", value);
                else printf(".");
            }
        }
        printf("₩n");
    }
}

/*
 * 메모리 영역의 address 주소의 값을 value 로 수정한다.
 */
void edit_memory(Memories* memories_state, int address, short value){
    assert(address < MEMORIES_SIZE && address >= 0);
    assert(value >= 0 && value <= 0xFF);

    memories_state->data[address].value = value;
}
```

5.20 memory.h

```
#ifndef __MEMORY_H__
#define __MEMORY_H__

#define MEMORIES_SIZE (1024 * 1024) // 1MB
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <stdbool.h>

/*
 * 메모리 값 하나를 의미하는 wrapper struct
 */
typedef struct memory {
    short value;
} Memory;
```

```
/*
 * 메모리 영역을 저장하는 구조체
 */
typedef struct memories {
    Memory data[MEMORIES_SIZE];
    int last_idx;
} Memories;

/*
 * Memories 구조체를 생성(할당)한다.
 */
Memories* construct_memories();

/*
 * start ~ end 메모리 영역을 출력한다.
 * start 와 end 는 10 진수로 변환한 값이다.
 */
void print_memories(Memories* memories_state, int start, int end);

/*
 * 메모리 영역의 address 주소의 값을 value 로 수정한다.
 */
void edit_memory(Memories* memories_state, int address, short value);

/*
 * Memories 구조체를 생성하면서 할당했던 메모리를 해제한다.
 */
bool destroy_memories(Memories** memories_state);

#endif
```

5.21 opcode.c

```
#include "opcode.h"

/*
 * Opcode 구조체를 생성(할당)한다.
 */
Opcode* construct_opcode(){
```

```
    Opcode* op = (Opcode*)malloc(sizeof(Opcode));
    op->value = -1;
    return op;
}

/*
 * op_node 를 생성(할당)한다.
 */
struct op_node* construct_opnode(){
    struct op_node* node = (struct op_node*)malloc(sizeof(struct op_node));

    return node;
}

/*
 * OpcodeTable 을 생성(할당)한다.
 */
OpcodeTable* construct_opcode_table(){
    OpcodeTable* table = (OpcodeTable*)malloc(sizeof(OpcodeTable));
    table->list = (OpLinkedList**)malloc(sizeof(OpLinkedList*)*20);
    for(int i = 0; i<20;i++) {
        table->list[i] = (OpLinkedList*)malloc(sizeof(OpLinkedList));
        table->list[i]->head = construct_opnode();
        table->list[i]->tail = construct_opnode();

        table->list[i]->head->prev = NULL;
        table->list[i]->head->next = table->list[i]->tail;
        table->list[i]->tail->prev = table->list[i]->head;
        table->list[i]->tail->next = NULL;

        table->list[i]->head->data = construct_opcode();
        table->list[i]->tail->data = construct_opcode();
        table->list[i]->size = 0;
    }
    table->size = 20;
    return table;
}
```

```
/*
 * opcode 파일을 읽어서 OpcodeTable 에 적절히 저장한다
 */
bool build_opcode_table(OpcodeTable* table){
    FILE* fp = fopen("opcode.txt", "rt");
    if(!fp) {
        fprintf(stderr, "[ERROR] opcode file not exists₩n");
        return false;
    }
    char format_name[20];
    char name[20];
    unsigned int value;

    while (fscanf(fp, "%X %6s %5s",
            &value, name, format_name) != EOF){
        Opcode* opc = construct_opcode();

        strncpy(opc->mnemonic_name, name, 10);

        opc->value = value;
        if(opc->value > 0xFF) {
            fprintf(stderr, "[ERROR] %X %6s %5s is Invalid Input!₩n", value, name,
format_name);
            continue;
        }
//        printf("%s %d₩n", opc->mnemonic_name,opc->value);
        if(COMPARE_STRING(format_name, "1")){
            opc->format = OP_FORMAT_1;
        }else if(COMPARE_STRING(format_name, "2")){
            if(COMPARE_STRING(opc->mnemonic_name, "CLEAR") ||
                COMPARE_STRING(opc->mnemonic_name, "TIXR"))
                opc->format = OP_FORMAT_2_ONE_REG;
            else if(COMPARE_STRING(opc->mnemonic_name, "SHIFTL") ||
                    COMPARE_STRING(opc->mnemonic_name, "SHIFTR"))
                opc->format = OP_FORMAT_2_REG_N;
            else if(COMPARE_STRING(opc->mnemonic_name, "SVC"))
```

```
            opc->format = OP_FORMAT_2_ONE_N;
        else
            opc->format = OP_FORMAT_2_GEN;
    } else if(COMPARE_STRING(format_name, "3/4")){
        if(COMPARE_STRING(opc->mnemonic_name, "RSUB"))
            opc->format = OP_FORMAT_3_4_NO_OPERAND;
        else
            opc->format = OP_FORMAT_3_4_GEN;
    }
    insert_opcode(table, opc);
  }
  fclose(fp);

  return true;
}

/*
 * OpcodeTable 을 해제한다.
 */
bool destroy_opcode_table(OpcodeTable** table){
  assert(table);
  OpNode *cur;
  OpNode *next;
  OpLinkedList** list;
  int i, j;

  list = (*table)->list;
  for(i=0;i<(*table)->size;i++){
    cur = list[i]->head;

    for(j=0;j<list[i]->size+1;j++){
      next = cur->next;
      free(cur->data);
      free(cur);
      cur = next;
    }
    free(list[i]);
```

```
    }
    free(*table);

    return true;
}

/*
 * OpcodeTable 에 Opcode 를 추가한다.
 */
bool insert_opcode(OpcodeTable* table, Opcode* opc){
    OpNode* op_node = construct_opnode();

    op_node->data = opc;
    int hash = (int)hash_string(opc->mnemonic_name, 20);

    op_node->prev = table->list[hash]->tail->prev;
    op_node->next = table->list[hash]->tail;
    table->list[hash]->tail->prev->next = op_node;
    table->list[hash]->tail->prev = op_node;
    table->list[hash]->size += 1;

//    assert(opc->value >= 0);
//    assert(table->list[hash]->tail->prev->data->value >= 0);
    return true;
}

/*
 * OpcodeTable 에서 name 문자열 이름의 mnemonic 을 가진
 * Opcode 를 찾고 리턴한다.
 */
Opcode* find_opcode_by_name(OpcodeTable* table, char* name){
    assert(strlen(name) <= 14);
    assert(table);

    int hash = (int)hash_string(name, table->size);
    int i = 0;
    OpNode** cur = &(table->list[hash]->head);
```

```
    Opcode* opc;
//    printf("hash: %d₩n", hash);
    for(i=0;i<(table->list[hash]->size)+1;i++){
        opc = (*cur)->data;
//        printf("op-name: %s₩n", opc->mnemonic_name);
        if(opc->value == -1){
            cur = &((*cur)->next);
            continue;
        }
        if(COMPARE_STRING(name, opc->mnemonic_name)){
            return opc;
        }
        cur = &((*cur)->next);
    }
    return NULL;
}

/*
 * OpcodeTable 에 저장된 opcode 목록을 출력한다.
 */
void print_opcodes(OpcodeTable* table){
    int size = table->size;
    for(int i = 0;i < size;i++){
        printf("%2d : ", i);
        OpNode** cur = &(table->list[i]->head);
        Opcode* opc;
        for(int j=0;j<table->list[i]->size+1;j++){
            opc = (*cur)->data;
//            if(opc->value == -1) cnt -= 1;
            if(opc->value == -1){
                cur=&((*cur)->next);
                continue;
            }
            printf("[%s,%02X] ",opc->mnemonic_name,opc->value);
//            printf("gogo %s %d₩n", opc->mnemonic_name, opc->value);
//            cnt += 1;
            if(j != table->list[i]->size)
```

```
            printf(" -> ");
        cur = &((*cur)->next);
    }
    printf("₩n");
  }
}
```

5.22 opcode.h

```
#ifndef __OPCODE_H__
#define __OPCODE_H__

#include <stdbool.h>
#include "util.h"

/*
 * opcode 의 format 을 enum 으로 표현한다. (구현중)
 */
enum op_format {
   OP_FORMAT_1,

   OP_FORMAT_2_ONE_REG, OP_FORMAT_2_REG_N,
   OP_FORMAT_2_ONE_N, OP_FORMAT_2_GEN,

   OP_FORMAT_3_4_NO_OPERAND, OP_FORMAT_3_4_GEN
};

/*
 *  opcode 의 mnemonic 을 enum 으로 표현한다. (구현중)
 */
enum op_mnemonic {
   OP_ADD, OP_ADDR
};

/*
 * opcode 정보 하나를 나타낸다.
 */
typedef struct opcode {
   enum op_format format;
```

```
    enum op_mnemonic mnemonic;
    char mnemonic_name[10];
    int value;
} Opcode;

/*
 * op_linked_list 의 Node 역활
 */
typedef struct op_node{
    Opcode* data;
    struct op_node* prev;
    struct op_node* next;
} OpNode;

/*
 * 링크드 리스트
 */
typedef struct op_linked_list {
    struct op_node* head;
    struct op_node* tail;
    int size;
}OpLinkedList;

/*
 * opcode 정보들을 해시테이블 형태로 저장하기 위한 구조체.
 */
typedef struct opcode_table {
    OpLinkedList** list;
    int size;
} OpcodeTable;

/*
 * Opcode 구조체를 생성(할당)한다.
 */
Opcode* construct_opcode();

/*
```

```
 * op_node 를 생성(할당)한다.
 */
struct op_node* construct_opnode();

/*
 * OpcodeTable 을 생성(할당)한다.
 */
OpcodeTable* construct_opcode_table();

/*
 * opcode 파일을 읽어서 OpcodeTable 에 적절히 저장한다
 */
bool build_opcode_table(OpcodeTable* table);

/*
 * OpcodeTable 을 해제한다.
 */
bool destroy_opcode_table(OpcodeTable** table);

/*
 * OpcodeTable 에 Opcode 를 추가한다.
 */
bool insert_opcode(OpcodeTable* table, Opcode* opc);

/*
 * OpcodeTable 에서 name 문자열 이름의 mnemonic 을 가진
 * Opcode 를 찾고 리턴한다.
 */
Opcode* find_opcode_by_name(OpcodeTable* table, char* name);

/*
 * OpcodeTable 에 저장된 opcode 목록을 출력한다.
 */
void print_opcodes(OpcodeTable* table);

#endif
```

5.23 state.c

```
#include "state.h"
#include "util.h"
#include "assemble.h"

/*
 * History, 가상 Memory, Opcode 정보가 초기화(및 저장)된 State* 을 리턴한다.
 */
State* construct_state(){
   State* state_obj = (State*)malloc(sizeof(*state_obj));

   state_obj->histories_state = construct_histories();
   state_obj->memories_state = construct_memories();

   state_obj->opcode_table_state = construct_opcode_table();
   build_opcode_table(state_obj->opcode_table_state);

   state_obj->symbol_table_state = construct_symbol_table();
   state_obj->is_symbol_table = false;
   return state_obj;
}

/*
 * state_store 가 동적 할당한 모든 메모리를 해제한다.
 */
bool destroy_state(State **state_store){

   destroy_histories(&((*state_store)->histories_state));
   destroy_memories(&((*state_store)->memories_state));
   destroy_opcode_table(&(*state_store)->opcode_table_state);
   destroy_symbol_table(&((*state_store)->symbol_table_state));

   free(*state_store);

   return true;
}

/*
 * history_str 문자열을 명령어 히스토리에 기록한다.
 */
bool add_history(State *state_store, char* history_str){
   return push_history(state_store->histories_state,
         construct_history_with_string(history_str));
}

/*
 * 명령어 히스토리를 출력한다.
 */
void print_histories_state(State* state_store, char* last_command){
   print_history(state_store->histories_state, last_command);
```

```
}

/*
 * file 을 assemble 하여 state 변경 및 성공 오류 여부 리턴
 */
bool assemble_file(State *state_store, char *asm_file_name){
   assert(strlen(asm_file_name) < MAX_ASM_FILENAME_LENGTH);
   free(state_store->symbol_table_state);
   state_store->symbol_table_state = construct_symbol_table();

   if(!assemble_pass1(state_store, asm_file_name)) {
      state_store->is_symbol_table = false;
      return false;
   }

   if(!assemble_pass2(state_store, asm_file_name)){
      state_store->is_symbol_table = false;
      return false;
   }

   state_store->is_symbol_table = true;
   return true;
}

/*
 * assembler 의 pass1 과정을 구현하였다.
 */
bool assemble_pass1(State *state_store, char *asm_file_name) {
   FILE* asm_fp = fopen(asm_file_name, "r");
   char* tmp_file_name, *prefix, *lst_file_name, *obj_file_name;
   int location_counter = 0, stmt_size = 0, line_num = -1;
   Statement stmt;
   FILE* tmp_fp = NULL;

   prefix = before_dot(asm_file_name, MAX_ASM_FILENAME_LENGTH);

   tmp_file_name = concat_n(prefix, ".tmp", MAX_ASM_FILENAME_LENGTH);
   lst_file_name = concat_n(prefix, ".lst", MAX_ASM_FILENAME_LENGTH);
   obj_file_name = concat_n(prefix, ".obj", MAX_ASM_FILENAME_LENGTH);

   if(!tmp_file_name){
      fprintf(stderr, "[ERROR] Invalid File Format\n");
      error_handling_pass1or2(NULL, asm_fp, tmp_fp, NULL, tmp_file_name, lst_file_name, obj_file_name,
line_num);
      return false;
   }

   tmp_fp = fopen(tmp_file_name, "wt");
   if(!tmp_fp){
      fprintf(stderr, "[ERROR] Can't Create tmp file\n");
      error_handling_pass1or2(NULL, asm_fp, tmp_fp, NULL, tmp_file_name, lst_file_name, obj_file_name,
```

```
line_num);
      return false;
   }

   if(!asm_fp){
      fprintf(stderr, "[ERROR] Can't Open File\n");
      error_handling_pass1or2(NULL, asm_fp, tmp_fp, NULL, tmp_file_name, lst_file_name, obj_file_name,
line_num);
      return false;
   }
   line_num = 5;

   if(!read_statement(state_store->opcode_table_state,
                  asm_fp, tmp_fp,
                  &stmt,
                  false,
                  &location_counter,
                  &stmt_size)) {
      error_handling_pass1or2(&stmt, asm_fp, tmp_fp, NULL, tmp_file_name, lst_file_name, obj_file_name,
line_num);
      return false;
   }

   if(!stmt.comment && COMPARE_STRING(stmt.opcode->mnemonic_name, "START")){
      location_counter = strtol (stmt.tokens[0], NULL, 16);
      fprintf (tmp_fp, "%04X\t0\t%s\n", location_counter, stmt.raw_input);

      if (!read_statement(state_store->opcode_table_state,
                     asm_fp,
                     tmp_fp,
                     &stmt,
                     false,
                     &location_counter,
                     &stmt_size)){
         error_handling_pass1or2(&stmt, asm_fp, tmp_fp, NULL, tmp_file_name, lst_file_name, obj_file_name,
line_num);
         return false;
      }
      line_num += 5;
   }

   while (1){
      if (is_comment_stmt(&stmt)){
         record_stmt_for_pass1(&stmt, tmp_fp, &location_counter, NULL);
         if(is_end_condition(&stmt, asm_fp)) break;

         if (!read_statement(state_store->opcode_table_state, asm_fp, tmp_fp, &stmt, false, NULL, NULL)) {
            error_handling_pass1or2(&stmt, asm_fp, tmp_fp, NULL, tmp_file_name, lst_file_name, obj_file_name,
line_num);
            return false;
         }
```

```
            line_num += 5;
            continue;
        }
        int old_location_counter = location_counter;

        if (stmt.raw_symbol){
            if(find_symbol_by_name(state_store->symbol_table_state, stmt.raw_symbol)) {
                error_handling_pass1or2(&stmt, asm_fp, tmp_fp, NULL, tmp_file_name, lst_file_name, obj_file_name,
line_num);
                return false;
            }
            Symbol* symbol = construct_symbol();
            strncpy (symbol->label, stmt.raw_symbol, 10);

            symbol->location_counter = location_counter;

            // [TODO] symbol insert 테스트 필요
            insert_symbol(state_store->symbol_table_state, symbol);
        }
        update_location_counter_by_format(&stmt, &location_counter);

        if(!update_location_counter_by_mnemonic_name(&stmt, &location_counter)){
            error_handling_pass1or2(&stmt, asm_fp, tmp_fp, NULL, tmp_file_name, lst_file_name, obj_file_name,
line_num);
            return false;
        }

        if(!update_location_counter_by_plus_and_format(&stmt, &location_counter)){
            error_handling_pass1or2(&stmt, asm_fp, tmp_fp, NULL, tmp_file_name, lst_file_name, obj_file_name,
line_num);
            return false;
        }

//        fprintf (tmp_fp, "%04X\t%X\t%s\n",
//                (unsigned int) old_location_counter,
//                (unsigned int)(location_counter - old_location_counter),
//                stmt.raw_input);
        record_stmt_for_pass1(&stmt, tmp_fp, &location_counter, &old_location_counter);
        if(is_end_condition(&stmt, asm_fp)) break;

        if (!read_statement(state_store->opcode_table_state, asm_fp, tmp_fp, &stmt, false, NULL, NULL)) {
            error_handling_pass1or2(&stmt, asm_fp, tmp_fp, NULL, tmp_file_name, lst_file_name, obj_file_name,
line_num);
            return false;
        }
        line_num += 5;
    }

    fclose(asm_fp);
    fclose(tmp_fp);
    free(lst_file_name);
```

```
    free(tmp_file_name);
    free(obj_file_name);
//    print_symbols(state_store->symbol_table_state);
    return true;
}

bool assemble_pass2(State *state_store, char *asm_file_name) {
    FILE *tmp_fp, *lst_fp, *obj_fp;
    char *tmp_file_name, *lst_file_name, *obj_file_name;
    char* prefix = before_dot(asm_file_name, MAX_ASM_FILENAME_LENGTH);
    int line_num = -1;
    Statement stmt;
    int location_counter = 0, stmt_size = 0, obj_code, r_lc, start_lc;
    int* location_counters = malloc(sizeof(int)*1001);
    char* b_buf = malloc(sizeof(char)*1001);
    int location_counter_cnt = 0;
    bool is_base = false;
    int base;
    char symb[11] = {0, };
    char* obj_buf = malloc(sizeof(char)*1001);
    char* rec_head = malloc(sizeof(char)*31);

    tmp_file_name = concat_n(prefix,
                        ".tmp",
                        MAX_ASM_FILENAME_LENGTH);
    lst_file_name = concat_n(prefix,
                        ".lst",
                        MAX_ASM_FILENAME_LENGTH);
    obj_file_name = concat_n(prefix,
                        ".obj",
                        MAX_ASM_FILENAME_LENGTH);

    tmp_fp = fopen(tmp_file_name, "rt");
    lst_fp = fopen(lst_file_name, "wt");
    obj_fp = fopen(obj_file_name, "wt");

    if(!tmp_fp || !lst_fp || !obj_fp) {
        fprintf(stderr, "[ERROR] Can't Open Files \n");
        error_handling_pass1or2(NULL, lst_fp, tmp_fp, obj_fp, tmp_file_name, lst_file_name, obj_file_name,
line_num);
        return false;
    }

    start_lc = r_lc = location_counter;
    obj_buf[0] = '\0';
    line_num = 5;
    if(!read_statement(state_store->opcode_table_state,
                    NULL, tmp_fp,
                    &stmt,
                    true,
                    &location_counter,
```

```
                &stmt_size)) {
        error_handling_pass1or2(&stmt, lst_fp, tmp_fp, obj_fp, tmp_file_name, lst_file_name, obj_file_name,
line_num);
        return false;
    }
    if(!stmt.comment && COMPARE_STRING(stmt.opcode->mnemonic_name,"START")){
        if(stmt.raw_symbol)
            strncpy(symb, stmt.raw_symbol, 11);

        fprintf (lst_fp, "%d\t%04X%s\n", line_num, location_counter, stmt.raw_input);
        fprintf (obj_fp, "%19s\n", "");

        if (!read_statement(state_store->opcode_table_state,
                        NULL,
                        tmp_fp,
                        &stmt,
                        true,
                        &location_counter,
                        &stmt_size)){
            error_handling_pass1or2(&stmt, lst_fp, tmp_fp, obj_fp, tmp_file_name, lst_file_name, obj_file_name,
                            line_num);
            return false;
        }
        line_num += 5;
    }

    while (1){
        if(stmt.comment){
            fprintf (lst_fp, "%d\t%s\n", line_num, stmt.raw_input);
            if(is_end_condition(&stmt, tmp_fp)) break;
            if(!read_statement(state_store->opcode_table_state,
                        NULL, tmp_fp,
                        &stmt,
                        true,
                        &location_counter,
                        &stmt_size)) {
                error_handling_pass1or2(&stmt, lst_fp, tmp_fp, obj_fp, tmp_file_name, lst_file_name, obj_file_name,
                                line_num);
                return false;
            }
            line_num += 5;
            continue;
        }
        if(is_format(&stmt, 1) && !handling_format1(&stmt, &obj_code)) {
            error_handling_pass1or2(&stmt, lst_fp, tmp_fp, obj_fp, tmp_file_name, lst_file_name, obj_file_name,
                            line_num);
            return false;
        }
        else if(is_format(&stmt, 2) && !handling_format2(&stmt, &obj_code)){
            error_handling_pass1or2(&stmt, lst_fp, tmp_fp, obj_fp, tmp_file_name, lst_file_name, obj_file_name,
                            line_num);
```

```
        return false;
    }else if(is_format(&stmt, 3) &&
            !handling_format3(state_store->symbol_table_state,
                        &stmt,
                        &obj_code,
                        &location_counter,
                        &location_counters,
                        &location_counter_cnt,
                        stmt_size,
                        &is_base,
                        &base)) {
        error_handling_pass1or2(&stmt, lst_fp, tmp_fp, obj_fp, tmp_file_name, lst_file_name, obj_file_name,
                        line_num);
        return false;
    }
    else if(is_format(&stmt, 0) && !handling_format_default(state_store->symbol_table_state,
                                            &stmt,
                                            &obj_code,
                                            &is_base,
                                            &base,
                                            &b_buf)) {
        error_handling_pass1or2(&stmt, lst_fp, tmp_fp, obj_fp, tmp_file_name, lst_file_name, obj_file_name,
                        line_num);
        return false;
    }
    record_stmt_for_pass2(&stmt,
                    &obj_code,
                    &location_counter,
                    &r_lc,
                    &line_num,
                    lst_fp,
                    obj_fp,
                    &obj_buf,
                    &b_buf,
                    &rec_head);
    if(is_end_condition(&stmt, tmp_fp)) break;
    if(!read_statement(state_store->opcode_table_state,
                NULL, tmp_fp,
                &stmt,
                true,
                &location_counter,
                &stmt_size)) {
        error_handling_pass1or2(&stmt, lst_fp, tmp_fp, obj_fp, tmp_file_name, lst_file_name, obj_file_name,
                        line_num);
        return false;
    }

    line_num += 5;
}

location_counter += stmt_size;
```

```
    if(location_counter + 30 > r_lc + 30){
        snprintf(rec_head, 30, "T%06X%02X", r_lc, (uint8_t)strlen(obj_buf) / 2);
        fprintf (obj_fp, "%s%s\n", rec_head, obj_buf);
        r_lc = location_counter;
        obj_buf[0] = '\0';
    }

    for (int i = 0; i < location_counter_cnt; ++i){
        snprintf (rec_head, 30, "M%06X05", location_counters[i]);
        fprintf (obj_fp, "%s\n", rec_head);
    }

    snprintf (rec_head, 30, "E%06X", start_lc);
    fprintf (obj_fp, "%s\n", rec_head);
    snprintf (rec_head,
            30,
            "H%-6s%06X%06X",
            symb,
            start_lc,
            location_counter - start_lc);
    fseek (obj_fp, 0, SEEK_SET);
    fprintf (obj_fp, "%s\n", rec_head);

    if(tmp_fp) fclose(tmp_fp);
    if(lst_fp) fclose(lst_fp);
    if(obj_fp) fclose(obj_fp);

    remove(tmp_file_name);

    free(tmp_file_name);
    free(location_counters);
    free(b_buf);
    free(obj_buf);
    free(rec_head);

    fprintf(stdout, "\toutput file : [%s], [%s]\n", lst_file_name, obj_file_name);

    free(lst_file_name);
    free(obj_file_name);

    return true;
}
```

5.24 state.h

```
#ifndef __STATE_H__
#define __STATE_H__

#include "history.h"
```

```
#include "memory.h"
#include "opcode.h"
#include "symbol.h"
#include "assemble.h"
#include "util.h"
#include <stdlib.h>
#include <stdint.h>

#define MAX_ASM_FILENAME_LENGTH 300

/*
 * state 구조체에서는 명령어 히스토리, 가상 메모리 영역, Opcode 정보를 저장하는
 * 구조체 포인터들을 멤버 변수로 갖고있다.
 */
typedef struct state {
    // 명령어 히스토리
    Histories* histories_state;

    // 가상 메모리 영역
    Memories* memories_state;

    // opcode 파일을 읽어 들인 내용들
    OpcodeTable* opcode_table_state;

    // symbol 정보 저장
    SymbolTable* symbol_table_state;

    bool is_symbol_table;
} State;

/*
 * History, 가상 Memory, Opcode 정보가 초기화(및 저장)된 State* 을 리턴한다.
 */
State* construct_state();

/*
 * state_store 가 동적 할당한 모든 메모리를 해제한다.
```

```
 */
bool destroy_state(State **state_store);

/*
 * history_str 문자열을 명령어 히스토리에 기록한다.
 */
bool add_history(State *state_store, char* history_str);

/*
 * 명령어 히스토리를 출력한다.
 */
void print_histories_state(State* state_store, char* last_command);

/*
 * file 을 assemble 하여 state 변경 및 성공 오류 여부 리턴
 */
bool assemble_file(State *state_store, char *asm_file_name);

/*
 * assembler 의 pass1 과정을 구현하였다.
 */
bool assemble_pass1(State *state_store, char *asm_file_name);

/*
 * assembler 의 pass2 과정을 구현하였다.
 */
bool assemble_pass2(State *state_store, char *asm_file_name);

#endif
```

5.25 util.c

```
#include "state.h"
#include "util.h"

/*
 * hashtable 구현을 위한 hash function
 */
```

```
size_t hash_string (char *str, int hash_size){
    int32_t hash = 2829;
    int32_t c;
    size_t res;
    while((c = *str++)){
        hash = (hash * 615) + c;
    }
    res = (size_t)hash % hash_size;
    return res;
}

/*
 * 문자열 str 이 0 으로 변환될수있는지 확인한다.
 * ex, is_zero_str("0000") => return true
 * ex, is_zero_str("A00") => return false
 */
bool is_zero_str(char* str){
    assert(str);
    int len = (int)strlen(str);
    int i;
    for(i=0;i<len;i++)
        if(str[i] != '0')
            return false;
    return true;
}

/*
 * 문자열 str 이 16 진수인지 확인한다.
 * ex, is_valid_hex("00F1") => return true
 * ex2, is_valid_hex("FZ") => return false
 */
bool is_valid_hex(char* str){
    assert(str);
    int l = (int)strlen(str), i;
    for(i=0;i<l;i++) {
        if ('0' <= str[i] &&
            str[i] <= '9')
```

```
            continue;
        if('A' <= str[i] &&
           str[i] <= 'F')
            continue;
        if('a' <= str[i] &&
           str[i] <= 'f')
            continue;
        return false;
    }
    return true;
}


/*
 * 문자열 str 이 [0 ~ max_size-1] 범위의 적절한 주소값인지 확인한다.
 * ex, is_zero_str("00F", 100) => return true
 * ex, is_zero_str("FG", 100000) => return false
 */
bool is_valid_address(char *str, int max_size) {
    assert(str);
    int target = (int)strtol(str, NULL, 16);

    if(target < 0) return false; // 0 보다 큰지 검증
    if(target == 0 && !is_zero_str(str)) return false; // 올바른 hex 값인지 검증
    if(target >= max_size) return false; // 범위 내에 있는지 검증
    if(!is_valid_hex(str)) return false; // 올바른 hex 값인지 검증

    return true;
}

/*
 * 문자열에서  . 이전의 문자열 을 찾아서 리턴한다.
 * 예를들어 before_dot(2_5.asm) 은 2_5 가 리턴된다.
 */
char *before_dot(char *name, int size) {
    char *pre;
    char* dot;
```

```
    pre = malloc(sizeof(char)*size);

    strncpy(pre, name, size);

    dot = strrchr (pre,'.');

    if(dot == NULL){
        return NULL;
    }
    *dot = '₩0';

    return pre;
}

char *concat_n(char *name, char *name2, int max_size) {
    char* res;
    res = malloc(sizeof(char)*max_size);

    snprintf (res, max_size, "%s%s", name, name2);

    return res;
}
```

5.26 util.h

```
#ifndef __UTIL_H__
#define __UTIL_H__
#include <stdlib.h>
#include <assert.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

#define COMPARE_STRING(T, S) (strcmp ((T), (S)) == 0)

/*
 * hashtable 구현을 위한 hash function
 */
size_t hash_string (char *str, int hash_size);
```

```
/*
 * 문자열 str 이 0 으로 변환될수있는지 확인한다.
 * ex, is_zero_str("0000") => return true
 * ex, is_zero_str("A00") => return false
 */
bool is_zero_str(char* str);

/*
 * 문자열 str 이 16 진수인지 확인한다.
 * ex, is_valid_hex("00F1") => return true
 * ex2, is_valid_hex("FZ") => return false
 */
bool is_valid_hex(char* str);

/*
 * 문자열 str 이 [0 ~ max_size-1] 범위의 적절한 주소값인지 확인한다.
 * ex, is_zero_str("00F", 100) => return true
 * ex, is_zero_str("FG", 100000) => return false
 */
bool is_valid_address(char *str, int max_size);

/*
 * 문자열에서  . 이전의 문자열 을 찾아서 리턴한다.
 * 예를들어 before_dot(2_5.asm) 은 2_5 가 리턴된다.
 */
char *before_dot(char *name, int size);

/*
 * 두 문자열을 합치는 concat 함수
 */
char *concat_n(char *name, char *name2, int max_size);

#endif
```

5.27 symbol.h

```
#ifndef __SYMBOL_H__
#define __SYMBOL_H__

#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>

typedef struct symbol {
    char label[11];
    int location_counter;
}Symbol;

/*
 * 링크드 리스트 구현을 위한 노드
 */
typedef struct sym_node {
    Symbol* data;
    struct sym_node* prev;
    struct sym_node* next;
}SymNode;

/*
 * 링크드 리스트
 */
typedef struct sym_linked_list {
    SymNode* head;
    SymNode* tail;
    int size;
}SymLinkedList;

/*
 * symbol 정보들을 해시테이블 형태로 저장하기 위한 구조체.
 */
typedef struct symbol_table {
    SymLinkedList* list[40];
    int size;
```

```
}SymbolTable;

/*
 * Symbol table 생성자 함수
 */
SymbolTable* construct_symbol_table();

/*
 * Symbol table 소멸자 함수
 */
bool destroy_symbol_table(SymbolTable** table);

/*
 * SymNode 생성자 함수
 */
SymNode* construct_symbol_node();

/*
 * Symbol 생성자 함수
 */
Symbol* construct_symbol();

/*
 * SymbolTable 구조체의 해시테이블에 Symbol 을 추가한다.
 */
bool insert_symbol(SymbolTable* table, Symbol* symbol);

/*
 * Symbol 을 찾는 함수
 */
Symbol* find_symbol_by_name(SymbolTable *table, char *name);

/*
* SymbolTable 에 저장된 Symbol 들을 내림차순으로 출력한다
*/
void print_symbols(SymbolTable* table);
```

```
/*
 * Sort Comparator 함수
 */
int symbol_comparator(const void *a, const void *b);

#endif
```

5.28 symbol.c

```
#include "symbol.h"
#include "util.h"

/*
 * Symbol table 생성자 함수
 */
SymbolTable* construct_symbol_table(){
    SymbolTable* table = malloc(sizeof(*table));
    int i;
    for(i = 0; i < 40;i++){
        table->list[i] = (SymLinkedList*)malloc(sizeof(SymLinkedList));
        table->list[i]->head = construct_symbol_node();
        table->list[i]->tail = construct_symbol_node();

        table->list[i]->head->prev = NULL;
        table->list[i]->head->next = table->list[i]->tail;
        table->list[i]->tail->prev = table->list[i]->head;
        table->list[i]->tail->next = NULL;

        table->list[i]->head->data = construct_symbol();
        table->list[i]->tail->data = construct_symbol();
        table->list[i]->size = 0;
    }

    table->size = 40;
    return table;
}

/*
```

```
 * Symbol table 소멸자 함수
 */
bool destroy_symbol_table(SymbolTable** table){
    SymNode *cur;
    SymNode *next;
    SymLinkedList** list;
    int i, j;

    list = (*table)->list;
    for(i=0;i<(*table)->size;i++){
        cur = list[i]->head;

        for(j=0;j<list[i]->size+1;j++){
            next = cur->next;
            free(cur->data);
            free(cur);
            cur = next;
        }
        free(list[i]);
    }
    free(*table);

    return true;
}

/*
 * SymNode 생성자 함수
 */
SymNode* construct_symbol_node(){
    SymNode* node = (SymNode*)malloc(sizeof(SymNode));

    return node;
}

/*
 * Symbol 생성자 함수
 */
```

```
Symbol* construct_symbol(){
    Symbol* symb = (Symbol*)malloc(sizeof(Symbol));
    strncpy(symb->label, "---nono--", 11);

    return symb;
}

/*
 * SymbolTable 구조체의 해시테이블에 Symbol 을 추가한다.
 */
bool insert_symbol(SymbolTable* table, Symbol* symbol){
    SymNode *node = malloc(sizeof(SymNode));
    int hash = (int)hash_string(symbol->label, 40);

    node->data = symbol;

    node->prev = table->list[hash]->tail->prev;
    node->next = table->list[hash]->tail;
    table->list[hash]->tail->prev->next = node;
    table->list[hash]->tail->prev = node;
//    fprintf(stdout, "[SUCCESS?] insert symbol %s₩n", table->list[hash]->tail-
>prev->data->label);
    table->list[hash]->size += 1;

//    fprintf(stdout, "[SUCCESS] insert symbol %s₩n", symbol->label);

    return true;
}

/*
 * Symbol 을 찾는 함수
 */
Symbol * find_symbol_by_name(SymbolTable *table, char *name){
    int hash = (int)hash_string(name, 40);
    int i;

    SymNode** cur = &(table->list[hash]->head);
```

```
    Symbol* symb;

    for(i=0;i<(table->list[hash]->size)+1;i++){
        symb = (*cur)->data;
        if(COMPARE_STRING(symb->label, "---nono--")){
            cur = &((*cur)->next);
            continue;
        }

        if(COMPARE_STRING(symb->label, name)){
            return symb;
        }
        cur = &((*cur)->next);
    }
    return NULL;
}

/*
* SymbolTable 에 저장된 Symbol 들을 내림차순으로 출력한다
*/
void print_symbols(SymbolTable* table){
    int size = table->size;
    Symbol *list[1200] = {0};
    int i = 0;
    int num = 0;
    for(i = 0;i < size;i++){
        SymNode** cur = &(table->list[i]->head);
        Symbol* symb;
        for(int j=0;j<table->list[i]->size+1;j++){
            symb = (*cur)->data;

            if(COMPARE_STRING(symb->label, "---nono--")){
                cur=&((*cur)->next);
                continue;
            }
            list[num++] = symb;
            cur = &((*cur)->next);
```

```
        }
    }
//    symbol_comparator(NULL, NULL);
    qsort(list, num, sizeof(Symbol *), symbol_comparator);

    for (int k = 0; k < num; ++k){
        if(!list[k])
            return;
        printf ("₩t%-5s₩t%04X₩n", list[k]->label, list[k]->location_counter);
    }
}

/*
 * Sort Comparator 함수
 */
int symbol_comparator(const void *a, const void *b){
    if(!a || !b) return 0;

    Symbol *left = *(Symbol **) a;
    Symbol *right = *(Symbol **) b;

    if(!left) return 0;
    if(!right) return 0;

    return -1*strcmp(left->label, right->label);
}
```

5.29 assemble.h

```
#ifndef __ASSEMBLE_H__
#define __ASSEMBLE_H__

#include <stdlib.h>
#include <stdbool.h>
#include "opcode.h"
#include "symbol.h"
#include <stdint.h>
#include <stdio.h>
```

```
#define MAX_TOKENS_LENGTH 8

/*
 * 어셈블리 코드 한줄 Statement 라고 정의한다.
 * 코드의 정보를 이 구조체에 저장한다.
 */
typedef struct statement {
    char* raw_input;
    int token_cnt;
    char* tokens[MAX_TOKENS_LENGTH];
    bool comment;
    Opcode* opcode;
    char* raw_symbol;
    bool tmp_bool;
    bool plus;
}Statement;

/*
 * format 2 인 Statement 일때,
 * 비트 값으로 저장하기 위한 변수이다.
 */
typedef union bits_format2{
    struct
    {
        uint16_t r2     : 4;
        uint16_t r1     : 4;
        uint16_t opcode : 8;
    } bits;
    uint16_t res;
}BitsFormat2;

/*
 * format 3 인 Statement 일때,
 * 비트 값으로 저장하기 위한 변수이다.
 */
typedef union bits_format3{
```

```
    struct{
        uint32_t disp   : 12;
        uint32_t e      : 1;
        uint32_t p      : 1;
        uint32_t b      : 1;
        uint32_t x      : 1;
        uint32_t i      : 1;
        uint32_t n      : 1;
        uint32_t opcode : 6;
    } bits;
    uint32_t res;
}BitsFormat3;

/*
 * format 4 인 Statement 일때,
 * 비트 값으로 저장하기 위한 변수이다.
 */
typedef union bits_format4{
    struct{
        uint32_t addr: 20;
        uint32_t e      : 1;
        uint32_t p      : 1;
        uint32_t b      : 1;
        uint32_t x      : 1;
        uint32_t i      : 1;
        uint32_t n      : 1;
        uint32_t opcode : 6;
    } bits;
    uint32_t res;
}BitsFormat4;

/*
 * register mnemonic 을 숫자값으로 변환한다
 */
int reg_mnemonic_num (char *reg_mnemonic);

/*
```

```
 * Statement 의 format 을 검사한다.
 * ex)
 *      is_format(&stmt, 0) : END, BYTE, WORD, RESB 등인지 검사
 *      is_format(&stmt, 1) : format 1 인지 검사
 *      is_format(&stmt, 2) : format 2 인지 검사
 *      is_format(&stmt, 3) : format 3/4 인지 검사
 */
bool is_format(Statement* stmt, int num);


/*
 * Statement 의 format 에 따라서 적절히
 * obj_code, location_counter, line_num 등을 .obj, .list 파일에 기록한다
 */
bool record_stmt_for_pass2(Statement *stmt,
                           const int *obj_code,
                           const int *location_counter,
                           int *r_lc,
                           const int *line_num,
                           FILE *lst_fp,
                           FILE *obj_fp,
                           char **obj_buf,
                           char **byte_buf,
                           char **rec_head);

/*
 * Default format (WORD, BASE 등등) 의 statement 를
 * 적절히 handling 하여 is_base, obj_code, base, b_buf 등의 값을 조정한다.
 */
bool handling_format_default(SymbolTable *symbol_table,
                             Statement *stmt,
                             int *obj_code,
                             bool *is_base,
                             int *base,
                             char **b_buf);

/*
```

```
 * format 3/4 의 statement 를
 * 적절히 handling 하여 obj_code 를 조정한다.
 */
bool handling_format3(SymbolTable *symbol_table,
                      Statement *stmt,
                      int *obj_code,
                      const int *location_counter,
                      int **location_counters,
                      int *location_counter_cnt,
                      int stmt_size,
                      const bool *is_base,
                      const int *base);

/*
 * format 2 의 statement 를
 * 적절히 handling 하여 obj_code 를 조정한다.
 */
bool handling_format2(Statement *stmt, int *obj_code);

/*
 * format 1 의 statement 를
 * 적절히 handling 하여 obj_code 를 조정한다.
 */
bool handling_format1(Statement *stmt, int* obj_code);

/*
 * input 을 토크나이징하여 stmt->tokens 와 stmt->token_cnt 를 조정함
 */
bool tokenizing_stmt_tokens(Statement* stmt, char* input);

/*
 * Statement 가 주석인지 아닌지 확인한다
 */
bool is_comment_stmt(Statement* stmt);

/*
 * Statement 구조체에 이 statement 가 주석이라는 표시를 한다.
```

```
 */
bool mark_comment_stmt(Statement* stmt);

/*
 * +JSUB 과 같이 opcode 앞에 +가 붙었는지 확인
 */
bool is_plus_stmt(Statement *stmt, int str_idx);

/*
 * Statement 가 +JSUB 과 같이 opcode 앞에 +가 붙은 경우에
 * stmt->plus = true 로 설정함
 */
bool mark_plus_true_or_false(Statement *stmt, int str_idx);

/*
 * Statement 의 format 에 따라서 적절히 location_counter 값을 증가시킴
 */
void update_location_counter_by_format(Statement *stmt,
        int *location_counter);

/*
 * Statement 의 opcode 의 mnemonic 에 따라서 적절히 location_counter 를
증가시킴
 */
bool update_location_counter_by_mnemonic_name(Statement *stmt,
        int *location_counter);

/*
 * Statement 가 plus 이면서 format 3/4 인경우에 location_counter 를 1 증가 시킴
 */
bool update_location_counter_by_plus_and_format(Statement *stmt, int
*location_counter);

/*
 * pass 2 를 끝낼 시점인지 아닌지 확인함
 */
bool is_end_condition(Statement *stmt, FILE *fp);
```

```
/*
 * 에러일 경우 이 함수가 실행된다.
 * 파일들을 전부 close 하고, 파라미터로 보낸 이름의 파일들을 삭제한다.
 * 에러가 난 Line 을 출력해준다.
 */
bool error_handling_pass1or2(Statement *stmt,
                             FILE *fp1,
                             FILE *fp2,
                             FILE *fp3,
                             char *rm_file_name1,
                             char *rm_file_name2,
                             char *rm_file_name3,
                             int line_num);

/*
 * symbol 인지 아닌지에 따라서 적절히 handling 한다.
 */
bool symbol_handling(OpcodeTable *opcode_table,
                     Statement *stmt,
                     char *name);

/*
 * 파일로 부터 한줄을 읽어서 Statement 변수 에 적절히 초기화하여 저장한다.
 */
bool read_statement(OpcodeTable *opcode_table,
                    FILE *asm_fp,
                    FILE *tmp_fp,
                    Statement *stmt,
                    bool is_tmp,
                    int *location_counter,
                    int *stmt_size);

/*
 * Statement 의 종류에 따라서 적절히 tmp 파일에 기록한다
 */
bool record_stmt_for_pass1(Statement *stmt, FILE *fp, int *location_counter, int
```

```
*old_location_counter);

#endif
```

5.30 assemble.c

```
#include "assemble.h"

/*
 * register mnemonic 을 숫자값으로 변환한다
 */
int reg_mnemonic_num (char *reg_mnemonic){
    if (COMPARE_STRING(reg_mnemonic, "A")) return 0;
    if (COMPARE_STRING(reg_mnemonic, "X")) return 1;
    if (COMPARE_STRING(reg_mnemonic, "L")) return 2;
    if (COMPARE_STRING(reg_mnemonic, "B")) return 3;
    if (COMPARE_STRING(reg_mnemonic, "S")) return 4;
    if (COMPARE_STRING(reg_mnemonic, "T")) return 5;
    if (COMPARE_STRING(reg_mnemonic, "F")) return 6;
    if (COMPARE_STRING(reg_mnemonic, "PC")) return 8;
    if (COMPARE_STRING(reg_mnemonic, "SW")) return 9;
    return -1;
}

/*
 * Statement 의 format 을 검사한다.
 * ex)
 *      is_format(&stmt, 0) : END, BYTE, WORD, RESB 등인지 검사 ( DEFAULT
format 이라 부르겠다)
 *      is_format(&stmt, 1) : format 1 인지 검사
 *      is_format(&stmt, 2) : format 2 인지 검사
 *      is_format(&stmt, 3) : format 3/4 인지 검사
 */
bool is_format(Statement* stmt, int num){
    if(num == 1) {
        return stmt->opcode->format == OP_FORMAT_1;
    }else if(num == 2){
        return (stmt->opcode->format == OP_FORMAT_2_ONE_N ||
```

```
            stmt->opcode->format == OP_FORMAT_2_REG_N ||
            stmt->opcode->format == OP_FORMAT_2_ONE_REG ||
            stmt->opcode->format == OP_FORMAT_2_GEN
        );
    }else if(num == 3){
        return (stmt->opcode->format == OP_FORMAT_3_4_GEN ||
            stmt->opcode->format == OP_FORMAT_3_4_NO_OPERAND
        );
    }else if(num == 0){
        return stmt->opcode->format == OP_DEFAULT;
    }else{
        return false;
    }
}

/*
 * Statement 의 format 에 따라서 적절히
 * obj_code, location_counter, line_num 등을 .obj, .list 파일에 기록한다
 */
bool record_stmt_for_pass2(Statement *stmt,
                        const int *obj_code,
                        const int *location_counter,
                        int *r_lc,
                        const int *line_num,
                        FILE *lst_fp,
                        FILE *obj_fp,
                        char **obj_buf,
                        char **byte_buf,
                        char **rec_head) {
    const char *format;
    if (is_format(stmt, 1)){
        format = "%d₩t%04X%-30s%02X₩n";
        if(*location_counter + 1 > *r_lc + 30){
            snprintf(*rec_head, 30, "T%06X%02X", *r_lc, (uint8_t) strlen (*obj_buf) /
2);
            fprintf (obj_fp, "%s%s₩n", *rec_head, *obj_buf);
            *r_lc = *location_counter;
```

```
            (*obj_buf)[0] = '₩0';
        }
        sprintf ((*obj_buf) + strlen (*obj_buf), "%02X", (*obj_code));
    }else if (is_format(stmt, 2)){
        format = "%d₩t%04X%-30s%04X₩n";
        if(*location_counter + 2 > *r_lc + 30){
            snprintf(*rec_head, 30, "T%06X%02X", *r_lc, (uint8_t) strlen (*obj_buf) /
2);
            fprintf (obj_fp, "%s%s₩n", *rec_head, *obj_buf);
            *r_lc = *location_counter;
            (*obj_buf)[0] = '₩0';
        }
        sprintf ((*obj_buf) + strlen (*obj_buf), "%04X", (*obj_code));
    }else if (is_format(stmt, 3)){
        if (stmt->plus){
            format = "%d₩t%04X%-30s%08X₩n";
            if(*location_counter + 4 > *r_lc + 30){
                snprintf(*rec_head, 30, "T%06X%02X", *r_lc, (uint8_t) strlen (*obj_buf)
/ 2);
                fprintf (obj_fp, "%s%s₩n", *rec_head, *obj_buf);
                *r_lc = *location_counter;
                (*obj_buf)[0] = '₩0';
            }
            sprintf ((*obj_buf) + strlen (*obj_buf), "%08X", (*obj_code));
        }
        else{
            format = "%d₩t%04X%-30s%06X₩n";
            if(*location_counter + 3 > *r_lc + 30){
                snprintf(*rec_head, 30, "T%06X%02X", *r_lc, (uint8_t) strlen (*obj_buf)
/ 2);
                fprintf (obj_fp, "%s%s₩n", *rec_head, *obj_buf);
                *r_lc = *location_counter;
                (*obj_buf)[0] = '₩0';
            }
            sprintf ((*obj_buf) + strlen (*obj_buf), "%06X", (*obj_code));
        }
```

```
    }else if (COMPARE_STRING(stmt->opcode->mnemonic_name,"BYTE")){
        fprintf (lst_fp, "%d₩t%04X%-30s%s₩n", (*line_num), (*location_counter),
stmt->raw_input, (*byte_buf));
        format = NULL;
        if(*location_counter + (int)strlen(*byte_buf) > *r_lc + 30){
            snprintf(*rec_head, 30, "T%06X%02X", *r_lc, (uint8_t) strlen (*obj_buf) /
2);
            fprintf (obj_fp, "%s%s₩n", *rec_head, *obj_buf);
            *r_lc = *location_counter;
            (*obj_buf)[0] = '₩0';
        }
        sprintf ((*obj_buf) + strlen (*obj_buf), "%s", (*byte_buf));
    }else if (COMPARE_STRING(stmt->opcode->mnemonic_name, "WORD")){
        format = "%d₩t%04X%-30s%06X₩n";
        if(*location_counter + 3 > *r_lc + 30){
            snprintf(*rec_head, 30, "T%06X%02X", *r_lc, (uint8_t) strlen (*obj_buf) /
2);
            fprintf (obj_fp, "%s%s₩n", *rec_head, *obj_buf);
            *r_lc = *location_counter;
            (*obj_buf)[0] = '₩0';
        }
        sprintf ((*obj_buf) + strlen (*obj_buf), "%06X", (*obj_code));
    }else{
        fprintf (lst_fp, "%d₩t%s₩n",(*line_num), stmt->raw_input);
        format = NULL;
    }

    if (format)
        fprintf (lst_fp, format, (*line_num), (*location_counter), stmt->raw_input,
(*obj_code));
    return true;
}

/*
 * Default format (WORD, BASE 등등) 의 statement 를
 * 적절히 handling 하여 is_base, obj_code, base, b_buf 등의 값을 조정한다.
```

```
 */
bool handling_format_default(SymbolTable *symbol_table,
                             Statement *stmt,
                             int *obj_code,
                             bool *is_base,
                             int *base,
                             char **b_buf) {
    if (COMPARE_STRING(stmt->opcode->mnemonic_name, "BASE")){
        if(stmt->token_cnt != 1) return false;
        Symbol* symb = find_symbol_by_name(symbol_table, stmt->tokens[0]);
        if(!symb) return false;
        *is_base = true;
        *base = symb->location_counter;
        return true;
    }
    if (COMPARE_STRING(stmt->opcode->mnemonic_name, "NOBASE")){
        (*is_base) = false;
        return true;
    }
    if (COMPARE_STRING(stmt->opcode->mnemonic_name, "BYTE")){
        if(stmt->token_cnt != 1) return false;

        const char *operand = stmt->tokens[0];
        int len = strlen (operand);

        if (len > 500) return false;

        if (operand[0] == 'C'){
            int idx = 0;
            for (int i = 2; i < len-1; ++i){
                unsigned char ch = operand[i];
                uint8_t val[2] = { ch / 16 , ch % 16 };
                for (int j = 0; j < 2; ++j, ++idx){
                    if (val[j] <= 9)
                        (*b_buf)[idx] = val[j] + '0';
                    else
```

```
                    (*b_buf)[idx] = val[j] - 10 + 'A';
                }
            }
            (*b_buf)[idx] = '₩0';
        }
        else if (operand[0] == 'X'){
            int i;
            for (i = 2; i < len-1; ++i) (*b_buf)[i-2] = operand[i];
            (*b_buf)[i-2] = '₩0';
        }
        else {
            return false;
        }
        return true;
    }else if (COMPARE_STRING(stmt->opcode->mnemonic_name, "WORD")){
        if(stmt->token_cnt != 1) return false;
        int val = strtol (stmt->tokens[0], NULL, 10);
        *obj_code = val;

        return true;
    }

    return true;
}

/*
 * format 3/4 의 statement 를
 * 적절히 handling 하여 obj_code 를 조정한다.
 */
bool handling_format3(SymbolTable *symbol_table,
                      Statement *stmt,
                      int *obj_code,
                      const int *location_counter,
                      int **location_counters,
                      int *location_counter_cnt,
                      int stmt_size,
```

```
                const bool *is_base,
                const int *base) {
BitsFormat3 bitsFormat3; bitsFormat3.res = 0;
BitsFormat4 bitsFormat4;

if(stmt->plus){
    bitsFormat4.bits.opcode = (stmt->opcode->value >> 2);
    bitsFormat4.bits.e = stmt->plus;
}else{
    bitsFormat3.bits.opcode = (stmt->opcode->value >> 2);
    bitsFormat3.bits.e = stmt->plus;
}

if (stmt->opcode->format == OP_FORMAT_3_4_NO_OPERAND){
    if (stmt->token_cnt != 0) return false;
    if(stmt->plus){
        bitsFormat4.bits.n = 1;
        bitsFormat4.bits.i = 1;
        bitsFormat4.bits.x = 0;
        bitsFormat4.bits.b = 0;
        bitsFormat4.bits.p = 0;
        bitsFormat4.bits.addr = 0;
    }else {
        bitsFormat3.bits.n = 1;
        bitsFormat3.bits.i = 1;
        bitsFormat3.bits.x = 0;
        bitsFormat3.bits.b = 0;
        bitsFormat3.bits.p = 0;
        bitsFormat3.bits.disp = 0;
    }
} else {
    if (stmt->token_cnt > 2 || stmt->token_cnt < 1) return false;

    /* Index Mode */
    if (stmt->token_cnt == 2){
        if (strcmp (stmt->tokens[1], "X") != 0) return false;
```

```
    if(stmt->plus) bitsFormat4.bits.x = 1;
    else bitsFormat3.bits.x = 1;
}else{
    if(stmt->plus) bitsFormat4.bits.x = 0;
    else bitsFormat3.bits.x = 0;
}

const char *operand = stmt->tokens[0];

/* Addressing mode */
bool operand_is_constant = false;

// Immediate addressing
if (operand[0] == '#'){
    if(stmt->plus){
        bitsFormat4.bits.n = 0;
        bitsFormat4.bits.i = 1;
    }else{
        bitsFormat3.bits.n = 0;
        bitsFormat3.bits.i = 1;
    }
    if ('0' <= operand[1] && operand[1] <= '9')
        operand_is_constant = true;
    ++operand;
}
// Indirect addressing
else if (operand[0] == '@')
{
    if(stmt->plus){
        bitsFormat4.bits.n = 1;
        bitsFormat4.bits.i = 0;
    }else{
        bitsFormat3.bits.n = 1;
        bitsFormat3.bits.i = 0;
    }
    ++operand;
```

```
        }
        // simple addressing
        else{
            if(stmt->plus){
                bitsFormat4.bits.n = 1;
                bitsFormat4.bits.i = 1;
            }else{
                bitsFormat3.bits.n = 1;
                bitsFormat3.bits.i = 1;
            }
        }

        uint32_t operand_value;

        if (operand_is_constant){
            operand_value = strtol (operand, NULL, 10);
        }
        else{
            const Symbol *symb = find_symbol_by_name(
                    symbol_table,
                    (char*)operand);
            if(!symb){
                return false;
            }
            operand_value = symb->location_counter;
        }

        if (stmt->plus){
            bitsFormat4.bits.b = 0;
            bitsFormat4.bits.p = 0;
            bitsFormat4.bits.addr = operand_value;
            if (!operand_is_constant)
                (*location_counters)[(*location_counter_cnt)++] =
(*location_counter)+1;
        } else if (operand_is_constant){
            bitsFormat3.bits.b = 0;
```

```
    bitsFormat3.bits.p = 0;
    bitsFormat3.bits.disp = operand_value;
} else {
    /* Displacement */
    int32_t disp;

    /* PC relative */
    const size_t PC = (*location_counter) + stmt_size;

    disp = operand_value - PC;

    if (-(1 << 11) <= disp && disp < (1 << 11)){
        // PC relative O case
        bitsFormat3.bits.b = 0;
        bitsFormat3.bits.p = 1;
        bitsFormat3.bits.disp = disp;
    }
    else{
        // PC relative X case

        /* Base relative check */
        if ((*is_base) == false){
            // BASE X => error
            return false;
        }

        disp = operand_value - (*base);

        if (0 <= disp && disp < (1 << 12)){
            // BASE relative O
            bitsFormat3.bits.b = 1;
            bitsFormat3.bits.p = 0;
            bitsFormat3.bits.disp = disp;
        }
        else{
            // Base relative X
```

```
                return false;
            }

        }
    }
}

if (stmt->plus)
    *obj_code = bitsFormat4.res;
else
    *obj_code = bitsFormat3.res;

return true;
}

/*
 * format 3/4 의 statement 를
 * 적절히 handling 하여 obj_code 를 조정한다.
 */
bool handling_format2(Statement *stmt, int *obj_code) {
    BitsFormat2 bits;
    if(stmt->opcode->format == OP_FORMAT_2_GEN){
        if(stmt->token_cnt != 2) return false;
        int reg_no_1, reg_no_2;

        reg_no_1 = reg_mnemonic_num (stmt->tokens[0]);
        reg_no_2 = reg_mnemonic_num (stmt->tokens[1]);

        if(reg_no_1 == -1 || reg_no_2 == -1) return false;

        bits.bits.opcode = stmt->opcode->value;
        bits.bits.r1 = reg_no_1;
        bits.bits.r2 = reg_no_2;

    } else if(stmt->opcode->format == OP_FORMAT_2_ONE_REG){
        if(stmt->token_cnt != 1) return false;
```

```
    int reg_no = reg_mnemonic_num (stmt->tokens[0]);
    if(reg_no == -1) return false;

    bits.bits.opcode = stmt->opcode->value;
    bits.bits.r1 = reg_no;
    bits.bits.r2 = 0;

} else if(stmt->opcode->format == OP_FORMAT_2_REG_N){
    if(stmt->token_cnt != 2) return false;

    int reg_no = reg_mnemonic_num(stmt->tokens[0]);
    char *endptr;
    long int n = strtol (stmt->tokens[1], &endptr, 16);

    if(reg_no == -1 || *endptr != '₩0' || n > 0xF || n < 0)
        return false;
    bits.bits.opcode = stmt->opcode->value;
    bits.bits.r1 = reg_no;
    bits.bits.r2 = n;
} else if(stmt->opcode->format == OP_FORMAT_2_ONE_N){

    if(stmt->token_cnt != 1) return false;
    char *endptr;
    long int n = strtol (stmt->tokens[0], &endptr, 16);

    if (*endptr != '₩0' || n > 0xF || n < 0) return false;

    bits.bits.opcode = stmt->opcode->value;
    bits.bits.r1 = n;
    bits.bits.r2 = 0;
} else{
    assert(false);
}
*obj_code = bits.res;
```

```
    return true;
}

/*
 * format 1 의 statement 를
 * 적절히 handling 하여 obj_code 를 조정한다.
 */
bool handling_format1(Statement* stmt, int* obj_code){
    if (stmt->token_cnt != 0) return false;
    *obj_code = stmt->opcode->value;

    return true;
}

/*
 * input 을 토크나이징하여 stmt->tokens 와 stmt->token_cnt 를 조정함
 */
bool tokenizing_stmt_tokens(Statement* stmt, char* input){
    stmt->token_cnt = 0;
    stmt->tokens[stmt->token_cnt] = strtok (input, " ,₩t₩n");
    while (stmt->token_cnt <= 15 && stmt->tokens[stmt->token_cnt])
        stmt->tokens[++stmt->token_cnt] = strtok (NULL, " ,₩t₩n");

    return true;
}

/*
 * Statement 가 주석인지 아닌지 확인한다
 */
bool is_comment_stmt(Statement* stmt){
    if(stmt->tokens[0][0] != '.') return false;
    if(stmt->comment) return true;
    return true;
}

/*
```

```
 * Statement 구조체에 이 statement 가 주석이라는 표시를 한다.
 */
bool mark_comment_stmt(Statement* stmt){
    assert(is_comment_stmt(stmt));

    stmt->comment = true;
    stmt->opcode = NULL;

    return true;
}

/*
 * +JSUB 과 같이 opcode 앞에 +가 붙었는지 확인
 */
bool is_plus_stmt(Statement *stmt, int str_idx) {
    if(stmt->tokens[str_idx][0] != '+') return false;

    return true;
}

/*
 * Statement 가 +JSUB 과 같이 opcode 앞에 +가 붙은 경우에
 * stmt->plus = true 로 설정함
 */
bool mark_plus_true_or_false(Statement *stmt, int str_idx) {
    if(is_plus_stmt(stmt, str_idx))
        stmt->plus = true;
    else
        stmt->plus = false;

    return true;
}

/*
 * Statement 의 format 에 따라서 적절히 location_counter 값을 증가시킴
 */
```

```
void update_location_counter_by_format(Statement *stmt, int *location_counter) {
    if (is_format(stmt, 1)){
        *location_counter += 1;
        return;
    }
    else if (is_format(stmt, 2)){
        *location_counter += 2;
        return;
    }
    else if (is_format(stmt, 3)){
        *location_counter += 3;
    }
}

/*
 * Statement 의 opcode 의 mnemonic 에 따라서 적절히 location_counter 를
증가시킴
 */
bool update_location_counter_by_mnemonic_name(Statement *stmt,
        int *location_counter){
    int len, b;
    if (COMPARE_STRING(stmt->opcode->mnemonic_name, "BYTE")){
        if (stmt->token_cnt != 1) return false;

        const char *operand = stmt->tokens[0];

        if (operand[1] != '₩'') return false;

        len = strlen (operand);

        if (operand[0] == 'C') b = len - 3;
        else if (operand[0] == 'X') b = (len - 3) / 2;
        else return false;

        if (operand[len-1] != '₩'') return false;
```

```
        *location_counter += b;
    } else if (COMPARE_STRING(stmt->opcode->mnemonic_name, "WORD")){
        if (stmt->token_cnt != 1) return false;
        *location_counter += 3;
    } else if (COMPARE_STRING(stmt->opcode->mnemonic_name,"RESB")){
        if (stmt->token_cnt != 1) return false;
        int cnt = strtol (stmt->tokens[0], NULL, 10);
        *location_counter += cnt;
    } else if (COMPARE_STRING(stmt->opcode->mnemonic_name, "RESW")){
        if (stmt->token_cnt != 1) return false;
        int cnt = strtol (stmt->tokens[0], NULL, 10);
        *location_counter += cnt * 3;
    }

    return true;
}

/*
 * Statement 가 plus 이면서 format 3/4 인경우에 location_counter 를 1 증가 시킴
 */
bool update_location_counter_by_plus_and_format(Statement *stmt, int
*location_counter){
    if (stmt->plus){
        if (is_format(stmt, 3)) ++(*location_counter);
        else return false;
    }

    return true;
}

/*
 * pass 2 를 끝낼 시점인지 아닌지 확인함
 */
bool is_end_condition(Statement *stmt, FILE *fp) {
    if (feof (fp) != 0)
        return true;
```

```
    else if (!stmt->comment && COMPARE_STRING(stmt->opcode->mnemonic_name,"END"))
        return true;

    return false;
}

/*
 * 에러일 경우 이 함수가 실행된다.
 * 파일들을 전부 close 하고, 파라미터로 보낸 이름의 파일들을 삭제한다.
 * 에러가 난 Line 을 출력해준다.
 */
bool error_handling_pass1or2(Statement *stmt,
                        FILE *fp1,
                        FILE *fp2,
                        FILE *fp3,
                        char *rm_file_name1,
                        char *rm_file_name2,
                        char *rm_file_name3,
                        int line_num){
    if(fp1) fclose(fp1);
    if(fp2) fclose(fp2);
    if(fp3) fclose(fp3);

    if(rm_file_name1) remove(rm_file_name1);
    if(rm_file_name2) remove(rm_file_name2);
    if(rm_file_name3) remove(rm_file_name3);

    if(line_num != -1 && stmt && stmt->raw_input) {
        fprintf(stderr, "[ERROR] Line %d: %s ₩n", line_num, stmt->raw_input);
    }

    return true;
}

/*
```

```
 * symbol 인지 아닌지에 따라서 적절히 handling 한다.
 */
bool symbol_handling(OpcodeTable *opcode_table,
                     Statement *stmt,
                     char *name) {

    Opcode* opc = find_opcode_by_name(opcode_table, name);

    int offset;

    if(opc){
        stmt->raw_symbol = NULL;
        stmt->opcode = opc;
        offset = 1;
    } else {
        if (stmt->token_cnt <= 1) return false;

        mark_plus_true_or_false(stmt, 1);
        if(stmt->plus) name = &stmt->tokens[1][1];
        else name = stmt->tokens[1];

        opc = find_opcode_by_name (opcode_table, name);
        if (!opc) return false;

        offset = 2;
        stmt->opcode = opc;
        stmt->raw_symbol = stmt->tokens[0];
    }
    for (size_t i = offset; i < (size_t)stmt->token_cnt; ++i)
        stmt->tokens[i - offset] = stmt->tokens[i];

    stmt->token_cnt -= offset;

    return true;
}
```

```
/*
 * 파일로 부터 한줄을 읽어서 Statement 변수 에 적절히 초기화하여 저장한다.
 */
bool read_statement(OpcodeTable *opcode_table,
                    FILE *asm_fp,
                    FILE *tmp_fp,
                    Statement *stmt,
                    bool is_tmp,
                    int *location_counter,
                    int *stmt_size) {
    FILE* fp;
    static char raw_input[220];
    static char tmp_input[200];
    int length = 0;
    char *op_tok;
    int offset, i;

    if(is_tmp) fp = tmp_fp;
    else fp = asm_fp;

    if(!fgets(raw_input, 220, fp)){
//        fprintf(stderr,"[DEBUG]");
        return false;
    }

    length = strlen(raw_input);
    if(!feof(fp) && raw_input[length - 1] != '₩n') return false;

    raw_input[length - 1] = '₩0';

//    printf("%s₩n", raw_input);

    if(is_tmp){
        sscanf (raw_input, "%X₩t%X%n", location_counter, stmt_size, &offset);
        for (i = 0; raw_input[offset + i]; i++)
            raw_input[i] = raw_input[offset + i];
```

```
        raw_input[i] = 0;
    }

    strncpy (tmp_input, raw_input, 200);
    stmt->raw_input = raw_input;

    tokenizing_stmt_tokens(stmt, tmp_input);

    if (is_comment_stmt(stmt) && mark_comment_stmt(stmt)) return true;

    stmt->comment = false;

    if (stmt->token_cnt == 0 || stmt->token_cnt > MAX_TOKENS_LENGTH)
        return false;

    mark_plus_true_or_false(stmt, 0);

    if(stmt->plus) op_tok = &stmt->tokens[0][1];
    else op_tok = stmt->tokens[0];

    if(!symbol_handling(opcode_table,
                        stmt,
                        op_tok)) return false;

    return true;
}

/*
 * Statement 의 종류에 따라서 적절히 tmp 파일에 기록한다
 */
bool record_stmt_for_pass1(Statement *stmt,
                           FILE *fp,
                           int *location_counter,
                           int *old_location_counter) {
    if(is_comment_stmt(stmt)){
        fprintf(fp, "%04X₩t0₩t%s₩n", *location_counter, stmt->raw_input);
```

```
        return true;
    }
    fprintf (fp, "%04X₩t%X₩t%s₩n",
            (unsigned int) *old_location_counter,
            (unsigned int)(*location_counter - *old_location_counter),
            stmt->raw_input);
    return true;
}
```