
과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 3>>

서강대학교 컴퓨터공학과

[20161631]

[임동진]

목 차

1. 프로그램 개요	6
2. 프로그램 설명	6
2.1 프로그램 흐름도	6
3. 모듈 정의	8
3.1 모듈 이름 : execute_progaddr(Command *user_command, State *state_store)	8
3.1.1 기능	8
3.1.2 사용 변수	8
3.2 모듈 이름: execute_run(State *state_store)	8
3.2.1 기능	8
3.2.2 사용 변수	8
3.3 모듈이름 execute_bp(Command *user_command, State *state_store)	9
3.3.1 기능	9
3.3.2 사용변수	9
3.4 모듈이름: execute_bp_clear(State *state_store)	9
3.4.1 기능	9
3.4.2 사용변수	9
3.5 모듈이름: execute_loader(Command *user_command, State *state_store)	9
3.5.1 기능	9
3.5.2 사용변수	9
3.6 모듈이름: execute_bp_list(State* state_store)	10
3.6.1 기능	10
3.6.2 사용변수	10
3.7 모듈이름: validate_progaddr_parameters(Command *user_command)	10
3.7.1 기능	10
3.7.2 사용변수	10
3.8 모듈이름: validate_bp_parameters(Command* user_command)	10
3.8.1 기능	10
3.8.2 사용변수	10
3.9 모듈이름: validate_bp_clear_parameters(Command* user_command)	10
3.9.1 기능	10
3.9.2 사용변수	11
3.10 모듈이름: validate_loader_parameters(Command* user_command)	11
3.10.1 기능	11
3.10.2 사용변수	11
3.11 모듈이름: op_format_by_op_num(int op_num)	11
3.11.1 기능	11
3.11.2 사용변수	11
3.12 모듈이름: registers 구조체	11
3.12.1 기능	11
3.12.2 정의 내용	12
3.13 모듈이름: load_info_node 구조체	12
3.13.1 기능	12

3.13.2 정의 내용	12
3.14 모듈이름: load_info_list 구조체	12
3.14.1 기능	12
3.14.2 정의 내용	12
3.15 모듈이름: debugger 구조체	12
3.15.1 기능	12
3.15.2 정의 내용	13
3.16 모듈이름: instruction 구조체	13
3.16.1 기능	13
3.16.2 정의 내용	14
3.17 모듈이름: load_info_type ENUM	15
3.17.1 기능	15
3.17.2 정의 내용	15
3.18 모듈이름: Operator ENUM	15
3.18.1 기능	15
3.18.2 정의 내용	15
3.19 모듈이름: ADDRESSING_MODE ENUM	16
3.19.1 기능	16
3.19.2 정의 내용	16
3.20 모듈이름: construct_debugger()	16
3.20.1 기능	16
3.20.2 사용변수	16
3.21 모듈이름: destroy_debugger(Debugger** debugger)	16
3.21.1 기능	16
3.21.2 사용변수	16
3.22 모듈이름: construct_registers()	16
3.22.1 기능	16
3.22.2 사용변수	16
3.23 모듈이름: destroy_registers(Registers** registers)	17
3.23.1 기능	17
3.23.2 사용변수	17
3.24 모듈이름: reset_registers(Registers* registers)	17
3.24.1 기능	17
3.24.2 사용변수	17
3.25 모듈이름: loader_linker(Debugger *debugger, Memories *memories)	17
3.25.1 기능	17
3.25.2 사용변수	17
3.26 모듈이름: run(Debugger *debugger, Memories *memories)	18
3.26.1 기능	18
3.26.2 사용변수	18
3.27 모듈이름: loader_linker_pass1(Debugger *debugger)	19
3.27.1 기능	19
3.27.2 사용변수	19
3.28 모듈이름: loader_linker_pass1_one(Debugger *debugger, int file_num, int *csaddr)	19
3.28.1 기능	19
3.28.2 사용변수	19
3.29 모듈이름: loader_linker_pass2(Debugger *debugger, Memories *memories)	20

3.29.1 기능	20
3.29.2 사용변수	20
3.30 모듈이름: loader_linker_pass2_one(Debugger *debugger, Memories *memories, int file_num, int *csaddr)	20
3.30.1 기능	20
3.30.2 사용변수	20
3.31 모듈이름: construct_load_info_list()	21
3.31.1 기능	21
3.31.2 사용변수	21
3.32 모듈이름: print_load_infos(LoadInfoList *load_infos)	21
3.32.1 기능	21
3.32.2 사용변수	21
3.33 모듈이름: execute_operator(Debugger *debugger, Memories *memories, Instruction *instruction)	22
3.33.1 기능	22
3.33.2 사용변수	22
3.34 모듈이름: calculate_TA(Instruction* instruction, Registers* registers)	23
3.34.1 기능	23
3.34.2 사용변수	23
3.35 모듈이름: handling_bp(Debugger *debugger, int instruction_size)	23
3.35.1 기능	23
3.35.2 사용변수	23
3.36 모듈이름: calculate_addressing_mode(Instruction* instruction, bool jump_op)	24
3.36.1 기능	24
3.36.2 사용변수	24
3.37 모듈이름: load_from_memory(Debugger *debugger, Memories *memories, Instruction *instruction, uint32_t *value, size_t bytes, bool jump_op)	24
3.37.1 기능	24
3.37.2 사용변수	24
3.38 모듈이름: store_to_memory(Debugger *debugger, Memories *memories, Instruction *instruction, uint32_t value, size_t bytes)	25
3.38.1 기능	25
3.38.2 사용변수	25
3.39 모듈이름: load_from_register(Debugger *debugger, int reg_id, uint32_t *val)	26
3.39.1 기능	26
3.39.2 사용변수	26
3.40 모듈이름: store_to_register(Debugger *debugger, int reg_id, uint32_t val)	26
3.40.1 기능	26
3.40.2 사용변수	26
3.41 모듈이름: get_reg_by_id(Registers *registers, int reg_id)	27
3.41.1 기능	27
3.41.2 사용변수	27
3.42 모듈이름: print_registers(Registers* registers)	27
3.42.1 기능	27
3.42.2 사용변수	27
4. 전역 변수 정의	27

5.	코드	27
5.1	20161631.c	27
5.2	20161631.h	28
5.3	command.c	28
5.4	command.h	30
5.5	command_execute.c	31
5.6	command_execute.h	39
5.7	command_macro.h	42
5.8	command_mapping.c	42
5.9	command_mapping.h	44
5.10	command_objects.h	45
5.11	command_shell.c	46
5.12	command_shell.h	46
5.13	command_validate_util.c	47
5.14	command_validate_util.h	54
5.15	dir.c	55
5.16	dir.h	56
5.17	history.c	56
5.18	history.h	59
5.19	memory.c	60
5.20	memory.h	62
5.21	opcode.c	63
5.22	opcode.h	67
5.23	state.c	70
5.24	state.h	77
5.25	util.c	79
5.26	util.h	81
5.27	symbol.h	82
5.28	symbol.c	84
5.29	assemble.h	87
5.30	assemble.c	92
5.31	debug.c	106
5.32	debug.h	123
5.33	Makefile	124

1. 프로그램 개요

SIC/XE 의 Linking Loader 를 구현한다. 여러 object file 을 linking 하여 메모리에 load 하는 작업을 수행한다. 또한 load 된 프로그램을 실행하고, 브레이크 포인트를 통해 디버깅할 수 있는 디버깅 기능까지 구현되어있다.

2. 프로그램 설명

2.1 프로그램 흐름도

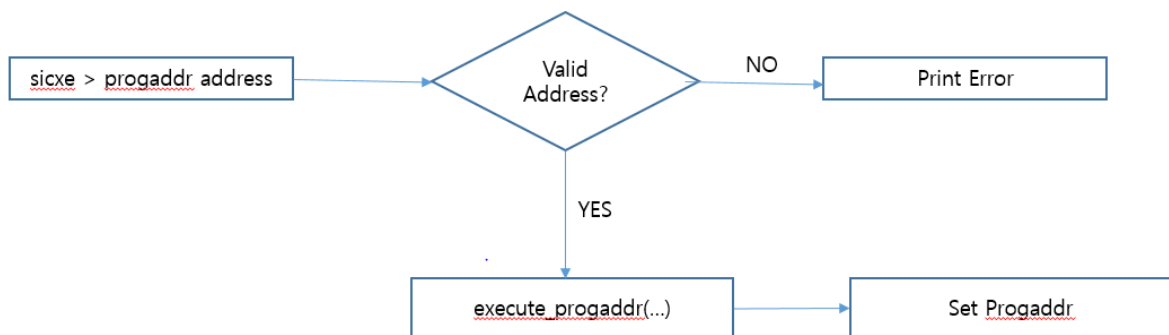


그림 1> progaddr 명령 흐름도

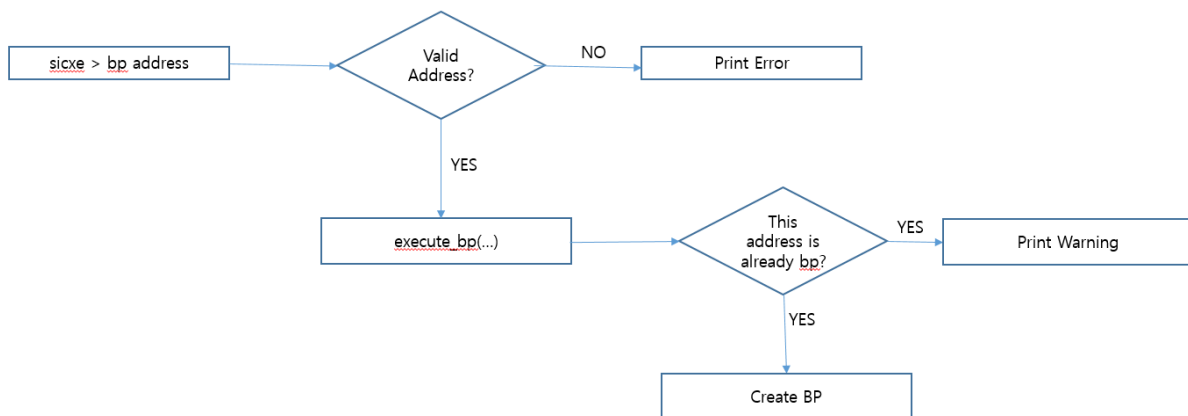


그림 2> bp [address] 명령 흐름도

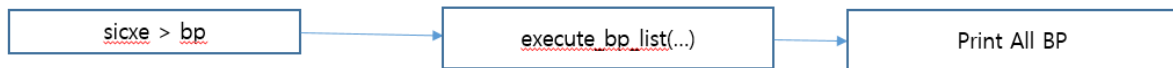


그림 3> bp 명령 흐름도

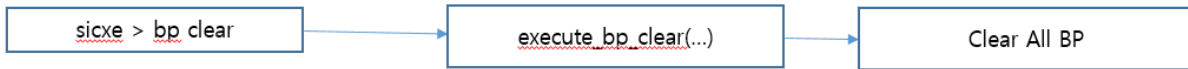


그림 4> bp clear 명령 흐름도

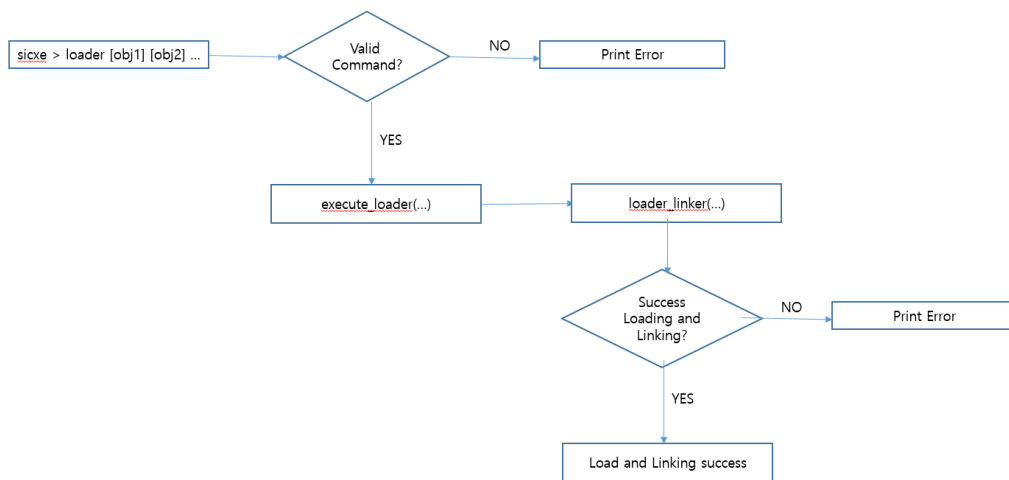


그림 5> loader [obj1] [obj2] ... 명령 흐름도

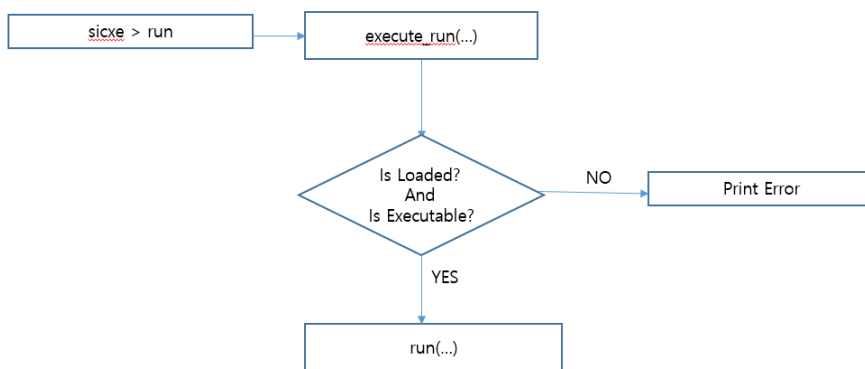


그림 6> run 명령 흐름도

3. 모듈 정의

3.1 모듈 이름 : `execute_progaddr(Command *user_command, State *state_store)`

3.1.1 기능

`progaddr [address]` 명령을 수행한다. 프로그램이 load 될 시작 주소를 지정하는 명령이다. ...

3.1.2 사용 변수

<code>Command *user_command</code>	.사용자가 입력한 명령에 대한 정보를 저장한다.
<code>State *state_store</code>	현재 프로그램의 모든 상태정보를 저장한다.
<code>int addr</code>	<code>Progaddr [address]</code> 에서 <code>address</code> 로 들어온 값을 저장한다.

3.2 모듈 이름: `execute_run(State *state_store)`

3.2.1 기능

현재 프로그램이 `run` 될수 있는지 확인하고, `run` 함수있다면 `run(...)`함수를 호출하고, 아니라면 에러를 출력해준다.. 실질적인 `run` 명령 수행은 `run(...)` 함수에서 수행된다.
`run(..)` 함수 수행 성공 실패 여부에 따라서 적절한 에러 핸들링을 진행한다.

3.2.2 사용 변수

<code>bool status;</code>	<code>run(..)</code> 함수 수행의 성공 실패 여부를 저장한다..
<code>State *state_store</code>	현재 프로그램의 모든 상태정보를 저장한다.

3.3 모듈이름 execute_bp(Command *user_command, State *state_store)

3.3.1 기능

bp [address] 명령을 수행한다. 브레이크 포인트를 설정한다. 이미 존재하는 bp 라면 경고를 출력한다.

3.3.2 사용변수

Command *user_command	.사용자가 입력한 명령에 대한 정보를 저장한다.
State *state_store	현재 프로그램의 모든 상태정보를 저장한다.
int addr	Progaddr [address]에서 address로 들어온 값을 저장한다.

3.4 모듈이름: execute_bp_clear(State *state_store)

3.4.1 기능

bp clear 명령을 수행한다. 모든 브레이크 포인트를 삭제한다..

3.4.2 사용변수

State *state_store	현재 프로그램의 모든 상태정보를 저장한다.
int cnt	반복문 수행을 위한 보조 변수

3.5 모듈이름: execute_loader(Command *user_command, State *state_store)

3.5.1 기능

loader_linker(...) 함수를 호출한다. 실질적인 loader [obj1] [obj2] ... 명령 수행은 loader_linker(...) 함수에서 수행된다. loader_linker(..) 함수 수행 성공 실패 여부에 따라서 적절한 에러 핸들링을 진행한다

3.5.2 사용변수

Command *user_command	.사용자가 입력한 명령에 대한 정보를 저장한다.
State *state_store	현재 프로그램의 모든 상태정보를 저장한다.
bool status	loader_linker(...) 함수 수행의 성공 실패 여부를 저장한다.

3.6 모듈이름: execute_bp_list(State* state_store)

3.6.1 기능

bp 들의 목록을 출력하는 명령인 bp 명령을 수행한다.

3.6.2 사용변수

State *state_store	현재 프로그램의 모든 상태정보를 저장한다.
int cnt	반복문 수행을 위한 보조 변수

3.7 모듈이름: validate_progaddr_parameters(Command *user_command)

3.7.1 기능

progaddr [address] 명령어의 파라미터를 검증한다. 검증 결과에 따라서 VALID_PARAMETERS 또는 INVALID_PARAMETERS 를 반환하여준다.

3.7.2 사용변수

Command *user_command	.사용자가 입력한 명령에 대한 정보를 저장한다.
-----------------------	----------------------------

3.8 모듈이름: validate_bp_parameters(Command* user_command)

3.8.1 기능

bp [address] 명령어의 파라미터를 검증한다. 검증 결과에 따라서 VALID_PARAMETERS 또는 INVALID_PARAMETERS 를 반환하여준다.

3.8.2 사용변수

Command *user_command	.사용자가 입력한 명령에 대한 정보를 저장한다.
-----------------------	----------------------------

3.9 모듈이름: validate_bp_clear_parameters(Command* user_command)

3.9.1 기능

bp clear 명령어의 파라미터를 검증한다. 검증 결과에 따라서 VALID_PARAMETERS 또는 INVALID_PARAMETERS 를 리턴하여준다

3.9.2 사용변수

Command *user_command	.사용자가 입력한 명령에 대한 정보를 저장한다.
-----------------------	----------------------------

3.10 모듈이름: validate_loader_parameters(Command* user_command)

3.10.1 기능

loader [obj1] [obj2] ... 명령어의 파라미터를 검증한다. 검증 결과에 따라서 VALID_PARAMETERS 또는 INVALID_PARAMETERS 를 리턴하여준다

3.10.2 사용변수

Command *user_command	.사용자가 입력한 명령에 대한 정보를 저장한다.
-----------------------	----------------------------

3.11 모듈이름: op_format_by_op_num(int op_num)

3.11.1 기능

Opcode 에 따라서 대응 되는 format(1 또는 2 또는 3/4)을 리턴한다.

3.11.2 사용변수

int op_num	Opcode 정보
bool format3_4[260]	Format3_4에 해당하는 opcode 정보를 true로 하여 저장함
bool format2[260]	Format2에 해당하는 opcode 정보를 true로 하여 저장함

3.12 모듈이름: registers 구조체

3.12.1 기능

레지스터 정보를 저장한다..

3.12.2 정의 내용

```
typedef struct registers {  
    uint32_t A;  
    uint32_t L;  
    uint32_t X;  
    uint32_t PC;  
    uint32_t B;  
    uint32_t S;  
    uint32_t T;  
    uint32_t SW;  
}Registers;
```

3.13 모듈이름: load_info_node 구조체

3.13.1 기능

load_info_list 구조체의 node 구조체 역할을 한다.

3.13.2 정의 내용

```
typedef struct load_info_node {  
    enum load_info_type type;  
    char name[15];  
    int addr;  
    int length;  
}LoadInfoNode;
```

3.14 모듈이름: load_info_list 구조체

3.14.1 기능

loader[obj1][obj2] ...명령의 로딩된 결과를 저장한다.

3.14.2 정의 내용

```
typedef struct load_info_list {  
    LoadInfoNode list[1001];  
    int count;  
} LoadInfoList;;
```

3.15 모듈이름: debugger 구조체

3.15.1 기능

load, run, bp, 레지스터 관리 등의 역할을함. 즉 해당 명령들의 수행을 위해 필요한 정보들을

통합해서 관리한다.

3.15.2 정의 내용

```
typedef struct debugger {  
    uint32_t start_address;  
    int end_address;  
    bool break_points[MAX_BP_NUM];  
    int bp_count;  
    Registers* registers;  
    int run_count;  
    int previous_bp;  
    char* filenames[3];  
    int file_count;  
    SymbolTable* estab;  
    LoadInfoList* load_infos;  
    uint32_t total_length;  
    bool is_running;  
    bool is_loaded;  
} Debugger;
```

3.16 모듈이름: instruction 구조체

3.16.1 기능

Instruction 과 관련한 정보를 저장한다..

3.16.2 정의 내용

```
typedef struct instruction{
    bool extend;
    unsigned char opcode;
    struct {
        union{
            struct{
                uint16_t r2 : 4;
                uint16_t r1 : 4;
                uint16_t opcode : 8;
            } p2;

            struct{
                uint32_t address: 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;
            } p3;

            struct{
                uint32_t address: 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;
            } p4;
            uint32_t val;
        } param;
        bool extend;
    } param;
} Instruction;
```

3.17 모듈이름: load_info_type ENUM

3.17.1 기능

Load 된 정보의 유형을 enum 으로 설정

3.17.2 정의 내용

```
enum load_info_type {  
    INFO_TYPE_CONTROL_SECTION,  
    INFO_TYPE_SYMBOL  
} LoadInfoType;
```

3.18 모듈이름: Operator ENUM

3.18.1 기능

Operator 이름과 opcode 를 매핑한 enum

3.18.2 정의 내용

```
typedef enum {  
    LDA = 0x00,  
    LDB = 0x68,  
    LDT = 0x74,  
    LDX = 0x04,  
    LDCH = 0x50,  
    STA = 0x0C,  
    STL = 0x14,  
    STX = 0x10,  
    STCH = 0x54,  
    JSUB = 0x48,  
    JEQ = 0x30,  
    JGT = 0x34,  
    JLT = 0x38,  
    J = 0x3C,  
    COMP = 0x28,  
    TD = 0xE0,  
    RD = 0xD8,  
    RSUB = 0x4C,  
    WD = 0xDC,  
    CLEAR = 0xB4,  
    COMPR = 0xA0,  
    TIXR = 0xB8  
} Operator;
```

3.19 모듈이름: ADDRESSING_MODE ENUM

3.19.1 기능

Addressing 모드를 enum 으로 매핑함.

3.19.2 정의 내용

```
typedef enum {  
    ENUM_IMMEDIATE_ADDRESSING,  
    ENUM_SIMPLE_ADDRESSING,  
    ENUM_INDIRECT_ADDRESSING,  
    ENUM_ADDRESSING_ERROR  
} ADDRESSING_MODE;
```

3.20 모듈이름: construct_debugger()

3.20.1 기능

Debugger 구조체의 생성자(메모리를 할당받고 초기화를 진행하는) 함수..

3.20.2 사용변수

없음.

3.21 모듈이름: destroy_debugger(Debugger** debugger)

3.21.1 기능

Debugger 구조체와 그와 연관된 변수들에 대한 소멸자(메모리를 해제하는) 함수.

3.21.2 사용변수

Debugger** debugger	해제할 debugger 를 포인팅함
---------------------	---------------------

3.22 모듈이름: construct_registers()

3.22.1 기능

Registers 구조체의 생성자 함수.

3.22.2 사용변수

없음.

3.23 모듈이름: **destroy_registers(Registers** registers)**

3.23.1 기능

Registers 구조체의 소멸자 함수.

3.23.2 사용변수

Registers** registers	해제할 registers 를 포인팅함
-----------------------	----------------------

3.24 모듈이름: **reset_registers(Registers* registers)**

3.24.1 기능

Registers 구조체를 리셋함..

3.24.2 사용변수

Registers* registers	레지스터 정보를 저장함.
----------------------	---------------

3.25 모듈이름: **loader_linker(Debugger *debugger, Memories *memories)**

3.25.1 기능

loader [obj1] [obj2].. 명령을 실질적으로 수행함. Pass1 을 수행하는 함수와 pass2 를 수행하는 함수를 호출하고 성공적으로 load 되었다면 load 된정보를 출력해서 보여주는 함수를 호출한다.

3.25.2 사용변수

Debugger *debugger	디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함..
Memories *memories	메모리 정보를 저장함

3.26 모듈이름: run(Debugger *debugger, Memories *memories)

3.26.1 기능

run 명령을 실질적으로 수행함. Load 된 프로그램을 실행하고 중간에 브레이크포인트 가 있으면 이를 적절히 핸들링한다.

3.26.2 사용변수

Debugger *debugger	디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함..
Memories *memories	메모리 정보를 저장함
Registers* registers	레지스터 정보를 저장함.
uint32_t tmp	Tmp 변수
bool is_continue	Continue 할지 여부를 저장함
uint8_t opcode	Opcode 를 저장함
uint32_t instruction_val	Instruction의 값을 저장함.
uint8_t memory_val	메모리 값을 저장함
int instruction_size	Instruction 의 크기를 저장함

3.27 모듈이름: loader_linker_pass1(Debugger *debugger)

3.27.1 기능

Loader 의 Pass1 과정을 수행한다..

3.27.2 사용변수

Debugger *debugger	디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함..
int csaddr	CSADDR 정보를 저장한다.

3.28 모듈이름: loader_linker_pass1_one(Debugger *debugger, int file_num, int *csaddr)

3.28.1 기능

Loader 의 Pass1 과정중에 파일 하나에 대한 작업을 수행한다..

3.28.2 사용변수

Debugger *debugger	디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함..
int* csaddr	CSADDR 정보를 저장한다.
Int file_num	파일 번호를 저장한다.
FILE *fp	파일 포인터
char buf[1010]	버퍼 정보를 저장함
bool is_header	헤더가 있는지 여부를 저장함
int base_address	base가 되는 주소를 저장함. 원래의 csaddr 주소

3.29 모듈이름: loader_linker_pass2(Debugger *debugger, Memories *memories)

3.29.1 기능

Loader 의 Pass2 과정을 수행한다...

3.29.2 사용변수

Debugger *debugger	디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함..
int csaddr	CSADDR 정보를 저장한다.
Memories *memories	메모리 정보를 저장한다.

3.30 모듈이름: loader_linker_pass2_one(Debugger *debugger, Memories *memories, int file_num , int *csaddr)

3.30.1 기능

Loader 의 Pass2 과정중 파일 하나에 대한 작업을 수행한다...

3.30.2 사용변수

Debugger *debugger	디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함..
int* csaddr	CSADDR 정보를 저장한다.
Int file_num	파일 번호를 저장한다.
Memories *memories	메모리 정보를 저장한다.
FILE* fp	파일 포인터

3.31 모듈이름: **construct_load_info_list()**

3.31.1 기능

Load_info_list 구조체의 생성자 함수

3.31.2 사용변수

없음

3.32 모듈이름: **print_load_infos(LoadInfoList *load_infos)**

3.32.1 기능

Load 된 정보를 출력해줌

3.32.2 사용변수

LoadInfoList* load_infos	로드된 정보를 저장함
Unsigned int total_length	총 할당된 길이를 저장
Const int count	로드된 정보의 개수를 저장함

3.33 모듈이름: execute_operator(Debugger *debugger, Memories *memories, Instruction *instruction)

3.33.1 기능

Instruction 에 대응되는 Operator(LDA, LDB, 등등)를 실행함

3.33.2 사용변수

Debugger *debugger	디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함..
Memories *memories	메모리 정보를 저장한다.
Instruction *instruction	Instruction 정보를 저장함.
Registers* registers	레지스터 정보를 저장한다.
uint32_t value,val1,val2	값 정보를 저장하기 위한 변수
static size_t device_input_idx	Device Read Write 구현을 위한 변수
char inputDevice[]	가상 device 변수

3.34 모듈이름: calculate_TA(Instruction* instruction, Registers* registers)

3.34.1 기능

Target Address 를 계산하여 리턴함.

3.34.2 사용변수

Instruction *instruction	Instruction 정보를 저장함.
Registers* registers	레지스터 정보를 저장한다.
uint32_t TA	Target address 정보를 저장한다.
uint32_t b,p,x	Instruction의 B,P,X 값을 저장함
uint32_t address	주소를 저장함.

3.35 모듈이름: handling_bp(Debugger *debugger, int instruction_size)

3.35.1 기능

브레이크 포인트를 핸들링함..

3.35.2 사용변수

Debugger *debugger	디버거 정보를 저장함
Int instruction_size	Instruction 크기를 저장함
Bool is_break	브레이크 포인트가 있는지 여부를 저장함
Unsigned int bp	Bp의 주소를 저장함

3.36 모듈이름: calculate_addressing_mode(Instruction* instruction, bool jump_op)

3.36.1 기능

Addressing 모드를 계산하여 리턴함. (immediate, simple, indirect). .

3.36.2 사용변수

Instruction* instruction	Instruction 정보를 저장함
Bool jump_op	해당 instruction 이 jump를 하는 명령인지 여부 저장
uint32_t n, i	Instruction의 N, I 값을 저장함

3.37 모듈이름: load_from_memory(Debugger *debugger, Memories *memories, Instruction *instruction, uint32_t *value, size_t bytes, bool jump_op)

3.37.1 기능

메모리로부터 값을 가져옴

3.37.2 사용변수

Debugger *debugger	디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함..
Memories *memories	메모리 정보를 저장한다.
Instruction *instruction	Instruction 정보를 저장함.
uint32_t *value	메모리로부터 가져온 값을 저장할 변수
Size_t bytes	크기
Bool jump_op	점프인지 여부

Uint32_t TA	Target Address 정보 저장
ADDRESSING_MODE addr_mode	어드레싱 모드 정보 저장

3.38 모듈이름: store_to_memory(Debugger *debugger, Memories *memories, Instruction *instruction, uint32_t value, size_t bytes)

3.38.1 기능

메모리에 값을 저장함

3.38.2 사용변수

Debugger *debugger	디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함..
Memories *memories	메모리 정보를 저장한다.
Instruction *instruction	Instruction 정보를 저장함.
uint32 *value	저장할 메모리 주소
Size_t bytes	크기
Uint32_t TA	Target Address 정보 저장
ADDRESSING_MODE addr_mode	어드레싱 모드 정보 저장

3.39 모듈이름: load_from_register(Debugger *debugger, int reg_id, uint32_t *val)

3.39.1 기능

레지스터로부터 값을 가져옴

3.39.2 사용변수

Debugger *debugger	디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함..
uint32_t *val	레지스터로부터 가져온 값을 저장할 변수
Int reg_id	레지스터 번호
Uint32_t TA	Target Address 정보 저장
ADDRESSING_MODE addr_mode	어드레싱 모드 정보 저장
uint32_t* reg	레지스터 번호에 대응되는 레지스터 포인터

3.40 모듈이름: store_to_register(Debugger *debugger, int reg_id, uint32_t val)

3.40.1 기능

레지스터에 값을 저장함

3.40.2 사용변수

Debugger *debugger	디버깅, load, run 등의 명령 수행을 위한 정보들을 저장함..
uint32_t val	레지스터에 저장할 값
Int reg_id	레지스터 번호

uint32_t* reg	레지스터 번호에 대응되는 레지스터 포인터
---------------	------------------------

3.41 모듈이름: get_reg_by_id(Registers *registers, int reg_id)

3.41.1 기능

레지스터 번호로 레지스터를 찾고 대응되는 포인터를 리턴함

3.41.2 사용변수

Registers* registers	레지스터들을 저장하는 변수
Int reg_id	레지스터 번호
uint32_t* regs[10]	레지스터 번호에 대응되는 레지스터 포인터를 매핑한 배열

3.42 모듈이름: print_registers(Registers* registers)

3.42.1 기능

현재의 레지스터 상태를 출력함.

3.42.2 사용변수

Registers* registers	레지스터들을 저장하는 변수
----------------------	----------------

4. 전역 변수 정의

전역변수 선언,정의 하지 않았음.

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 입력하기 전까지
 * 셸을 통해 명령어를 입력, 수행할 수 있도록 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 셸을 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, 매핑 한다.
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, 토큰나이징 하는 등의 함수들은
* 성공, 실패 여부에 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 셸을 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, 토큰나이징 하는 등의 함수들은
*   성공, 실패 여부에 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);
    case TYPE_PROGADDR:
        return execute_progaddr(user_command, state_store);
    case TYPE_RUN:
        return execute_run(state_store);
    case TYPE_BP:
        return execute_bp(user_command, state_store);
    case TYPE_BP_CLEAR:
        return execute_bp_clear(state_store);
    case TYPE_BP_LIST:
        return execute_bp_list(state_store);
    case TYPE_LOADER:
        return execute_loader(user_command, 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"
        "opodelist\n"
        "assemble filename\n"
        "type filename\n"
        "symbol\n"
        "loader [filename1] [filename2] ...\n"
        "run\n"
        "bp\n"
        "progaddr [address]\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 수정 한다.
*/
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;
}
```

/*

* opodelist 명령어

```

* 해시 테이블 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;
}

/*
* type 명령어
*/
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;
}

/*
* symbol 명령어
*/
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;
}

/*
 * progaddr 명령어
 */
shell_status execute_progaddr(Command *user_command, State *state_store) {
    int addr = (int)strtol(user_command->tokens[1], NULL, 16);

    state_store->debugger_state->start_address = (uint32_t)addr;
    fprintf(stdout, "Program starting address set to 0x%04X.\n", addr);
    state_store->debugger_state->is_loaded = false;
    return EXECUTE_SUCCESS;
}

/*
 * run 명령어
 */
shell_status execute_run(State *state_store){
    bool status;
    if(!state_store->debugger_state->is_loaded){
        fprintf(stderr, "[ERROR] Not Loaded\n");
        return EXECUTE_FAIL;
    }
    if(state_store->debugger_state->file_count != 1){
        fprintf(stderr, "[ERROR] Only Support One Object Run\n");
        fprintf(stderr, "[ERROR] Must Re-Load\n");
        return EXECUTE_FAIL;
    }
    if(!state_store->debugger_state->is_running){
        state_store->debugger_state->registers->PC = state_store->debugger_state->start_address;
    }

    status = run(state_store->debugger_state, state_store->memories_state);
    if(!status){
        state_store->debugger_state->is_running = false;
        state_store->debugger_state->is_loaded = false;
        return EXECUTE_FAIL;
    }

    return EXECUTE_SUCCESS;
}

/*
 * bp 명령어
 */
shell_status execute_bp(Command *user_command, State *state_store) {
    int addr = (int)strtol(user_command->tokens[1], NULL, 16);

```

```

    if(state_store->debugger_state->break_points[addr] == true){
        fprintf(stdout, "[warning] breakpoint already at %04X\n", addr);
        return EXECUTE_SUCCESS;
    }

    state_store->debugger_state->break_points[addr] = true;
    state_store->debugger_state->bp_count += 1;
    fprintf(stdout, "[ok] create breakpoint %04X\n", addr);
    return EXECUTE_SUCCESS;
}

/*
 * bp clear 명령어
 */
shell_status execute_bp_clear(State *state_store){
    int cnt = 0;
    for(int i = 0; i < MAX_BP_NUM; i++){
        if(state_store->debugger_state->bp_count == cnt)
            break;
        if(state_store->debugger_state->break_points[i] == true){
            state_store->debugger_state->break_points[i] = false;
            cnt++;
        }
    }
    state_store->debugger_state->bp_count = 0;

    fprintf(stdout, "[ok] clear all breakpoints\n");
    return EXECUTE_SUCCESS;
}

/*
 * loader 명령어
 */
shell_status execute_loader(Command *user_command, State *state_store) {
    // printf("loader linker execute\n");
    bool status;
    for(int i = 0; i < 3; i++)
        state_store->debugger_state->filenames[i] = NULL;

    for(size_t i = 1; i < user_command->token_cnt; i++)
        state_store->debugger_state->filenames[i - 1] = user_command->tokens[i];

    state_store->debugger_state->file_count = user_command->token_cnt - 1;
    status = loader_linker(state_store->debugger_state, state_store->memories_state);
    if(!status){
        for(int i = 0; i < 3; i++) state_store->debugger_state->filenames[i] = NULL;
        state_store->debugger_state->file_count = 0;
        return EXECUTE_FAIL;
    }

    state_store->debugger_state->run_count = 0;

```

```

state_store->debugger_state->previous_bp = -1;
// state_store->debugger_state->end_address 설정 해주기.
state_store->debugger_state->end_address = MAX_BP_NUM - 1;
state_store->debugger_state->is_loaded = true;

return EXECUTE_SUCCESS;
}

shell_status execute_bp_list(State* state_store){
    int cnt = 0;
    if(state_store->debugger_state->bp_count == 0){
        fprintf(stdout, "no breakpoints set.\n");
        return EXECUTE_SUCCESS;
    }
    fprintf(stdout, "breakpoint\n");
    fprintf(stdout, "-----\n");
    for(int i = 0; i < MAX_BP_NUM; i++){
        if(state_store->debugger_state->bp_count == cnt)
            break;
        if(state_store->debugger_state->break_points[i] == true) {
            fprintf(stdout, "%04X\n", i);
            cnt++;
        }
    }

    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);

/*
* opodelist 명령어
* 해시 테이블 형태로 저장된 opcode 목록을 출력해준다.
*/
shell_status execute_opodelist(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);

/*
* progaddr 명령어
*/
shell_status execute_progaddr(Command *user_command, State *state_store);

/*
* run 명령어
*/
shell_status execute_run(State *state_store);

/*
* bp <주소> 명령어
*/
shell_status execute_bp(Command *user_command, State *state_store);

/*
* bp clear 명령어
*/
shell_status execute_bp_clear(State *state_store);

/*
* bp list 출력해주는 명령어
*/
shell_status execute_bp_list(State* state_store);

```

```

/*
 * loader_linker 명령어
 */
shell_status execute_loader(Command *user_command, 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(user_command->type != TYPE_LOADER
        && !validate_tokenizing(user_command->raw_command,
                               (int) user_command->token_cnt,
                               TOKEN_MAX_NUM))
        return INVALID_INPUT;
    else if(user_command->type == TYPE_LOADER
        && !validate_tokenizing_for_not_comma(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 토큰나이징하여 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, "opodelist")){
    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 if(COMPARE_STRING(first_token, "progaddr")){
    user_command->type = TYPE_PROGADDR;
} else if(COMPARE_STRING(first_token, "bp")){
    if(user_command->token_cnt >= 2){
        char* second_token = user_command->tokens[1];
        if(COMPARE_STRING(second_token, "clear"))
            user_command->type = TYPE_BP_CLEAR;
        else
            user_command->type = TYPE_BP;
    } else {
        user_command->type = TYPE_BP_LIST;
    }
} else if(COMPARE_STRING(first_token, "run")){
    user_command->type = TYPE_RUN;
} else if(COMPARE_STRING(first_token, "loader")){
    user_command->type = TYPE_LOADER;
} 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 토큰나이징하여 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, TYPE_ASSEMBLE,
    TYPE_TYPE, TYPE_SYMBOL, TYPE_PROGADDR, TYPE_RUN, TYPE_BP,

```

```

    TYPE_BP_CLEAR, TYPE_LOADER, TYPE_BP_LIST
};

// 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"

/*
 * 사용자로부터 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>

/*
 * 사용자로부터 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"

/*
 * 토큰나이징이 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;
}

bool validate_tokenizing_for_not_comma(char *str, int token_cnt, int max_token_num){
    assert(str);

    int length = (int)strlen(str);
    int comma_cnt = 0;

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

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

    if(comma_cnt != 0)
        return false;

    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->type == TYPE_RUN ||
    user_command->type == TYPE_BP_LIST
) &&
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);
if(user_command->type == TYPE_PROGADDR)
return validate_progaddr_parameters(user_command);
if(user_command->type == TYPE_BP)
return validate_bp_parameters(user_command);
if(user_command->type == TYPE_BP_CLEAR)
return validate_bp_clear_parameters(user_command);
if(user_command->type == TYPE_LOADER)
return validate_loader_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;
}

```

```

/*
 * progaddr 명령어의 파라미터를 검증한다
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_progaddr_parameters(Command *user_command){
    assert(user_command);
    assert(user_command->type == TYPE_PROGADDR);

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

    // 적절한 주소 값인지 확인한다.
    if(!is_valid_address(user_command->tokens[1], MB))
        return INVALID_PARAMETERS;

    return VALID_PARAMETERS;
}

/*
 * bp 명령어의 파라미터를 검증한다
 *
 * @return VALID_PARAMETERS or INVALID_PARAMETERS
 */
shell_status validate_bp_parameters(Command* user_command){
    assert(user_command);
    assert(user_command->type == TYPE_BP);

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

    // 적절한 주소 값인지 확인한다.
    if(!is_valid_address(user_command->tokens[1], MB))
        return INVALID_PARAMETERS;

    return VALID_PARAMETERS;
}

shell_status validate_bp_clear_parameters(Command* user_command){
    assert(user_command);
    assert(user_command->type == TYPE_BP_CLEAR);

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

    return VALID_PARAMETERS;
}

```

```

shell_status validate_loader_parameters(Command* user_command){
    assert(user_command);
    assert(user_command->type == TYPE_LOADER);

    if(user_command->token_cnt > 4 || user_command->token_cnt == 1)
        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)

/*
 * 토큰나이프가 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);

bool validate_tokenizing_for_not_comma(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);

shell_status validate_progaddr_parameters(Command *user_command);

shell_status validate_bp_parameters(Command* user_command);

shell_status validate_bp_clear_parameters(Command* user_command);

shell_status validate_loader_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 역할을 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 해제 한다.
 */
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;

// DEFAULT OPCODE only just hard coding
if(COMPARE_STRING(name, "START")
    || COMPARE_STRING(name, "END")
    || COMPARE_STRING(name, "BYTE")
    || COMPARE_STRING(name, "WORD")
    || COMPARE_STRING(name, "RESB")
    || COMPARE_STRING(name, "RESW")
    || COMPARE_STRING(name, "BASE")
    || COMPARE_STRING(name, "NOBASE")){
    opc = construct_opcode();
    strncpy(opc->mnemonic_name, name, 10);
    opc->value = 0;
    opc->format = OP_DEFAULT;
    return opc;
}

for(i=0;i<(table->list[hash]->size)+1;i++){
    opc = (*cur)->data;
    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");
  }
}

```

// format 을 매핑 하는 함수

```

enum op_format op_format_by_op_num(int op_num){
  bool format3_4[260] = {
    [0x00] = true,
    [0x68] = true,
    [0x74] = true,
    [0x04] = true,
    [0x50] = true,
    [0x0C] = true,
    [0x14] = true,
    [0x10] = true,
    [0x54] = true,
    [0x48] = true,
    [0x30] = true,
    [0x34] = true,
    [0x38] = true,
    [0x3C] = true,
    [0x28] = true,
    [0xE0] = true,
    [0xD8] = true,
    [0x4C] = true,
    [0xDC] = true
  };
  bool format2[260] = {
    [0xB4] = true,
    [0xA0] = true,
    [0xB8] = true
  };
  if(format3_4[op_num]) return OP_FORMAT_3_4_GEN;
  if(format2[op_num]) return OP_FORMAT_2_GEN;
  return OP_FORMAT_1;
}

```

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,

    OP_DEFAULT
};

/*
/* 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 역할
*/
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* op);

/*
 * 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);

// format 을 매핑하는 함수
enum op_format op_format_by_op_num(int op_num);

```

```
#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 "debug.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;

    // run, bp, progaddr 등의 정보 저장
    Debugger* debugger_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 = strchr (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 해시 테이블 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 해시 테이블에 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_sym_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 해시 테이블에 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 토큰나이징하여 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 인 경우에 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(OpcodesTable *opcode_table,
        Statement *stmt,
        char *name);

/*
 * 파일로부터 한줄을 읽어서 Statement 변수 에 적절히 초기화하여 저장한다.
 */
bool read_statement(OpcodesTable *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 조정 한다.

*/

```

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 조정 한다.
*/
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 조정 한다.
 */
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 조정 한다.
*/
bool handling_format1(Statement* stmt, int* obj_code){
    if (stmt->token_cnt != 0) return false;
    *obj_code = stmt->opcode->value;

    return true;
}

/*
* input 을 토큰나이징하여 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] != '\0') 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] != '\0') 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;
}

```

5.31 debug.c

```

#include "debug.h"
#include <stdint.h>

```

```

/*
 * Struct or Enum Declarations and Definition
 */

```

// Instruction 정보를 저장함.

```

typedef struct instruction{
    bool extend;
    unsigned char opcode;
    struct {
        union{
            struct{
                uint16_t r2 : 4;
                uint16_t r1 : 4;
                uint16_t opcode : 8;
            } p2;

            struct{
                uint32_t address: 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;
            } p3;

            struct{
                uint32_t address: 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;
    } p4;
    uint32_t val;
    } param;
    bool extend;
    } param;
} Instruction;

```

// Operator 이름과 opcode 를 매핑

```

typedef enum {
    LDA = 0x00,
    LDB = 0x68,
    LDT = 0x74,
    LDX = 0x04,
    LDCH = 0x50,
    STA = 0x0C,
    STL = 0x14,
    STX = 0x10,
    STCH = 0x54,
    JSUB = 0x48,
    JEQ = 0x30,
    JGT = 0x34,
    JLT = 0x38,
    J = 0x3C,
    COMP = 0x28,
    TD = 0xE0,
    RD = 0xD8,
    RSUB = 0x4C,
    WD = 0xDC,
    CLEAR = 0xB4,
    COMPR = 0xA0,
    TIXR = 0xB8
} Operator;

```

// Addressing mode

```

typedef enum {
    ENUM_IMMEDIATE_ADDRESSING,
    ENUM_SIMPLE_ADDRESSING,
    ENUM_INDIRECT_ADDRESSING,
    ENUM_ADDRESSING_ERROR
} ADDRESSING_MODE;

```

```

/*
 * Static Function Declarations
 */

```

/* loader_linker 함수 구현을 위한 함수들 */

// pass1

```

static bool loader_linker_pass1(Debugger *debugger);

// pass2
static bool loader_linker_pass2(Debugger *debugger, Memories *memories);

// pass1 sub function
static bool loader_linker_pass1_one(Debugger *debugger, int file_num, int *csaddr);

// pass2 sub function
static bool loader_linker_pass2_one(Debugger *debugger, Memories *memories, int file_num , int *csaddr);

// 생성자 함수
static LoadInfoList* construct_load_info_list();

// load 된 정보를 출력함.
static void print_load_infos(LoadInfoList *load_infos);

/* run 함수를 구현을 위한 함수들 */

// 레지스터 상태를 출력함.
static void print_registers(Registers* registers);

// 명령을 실행함.
static bool execute_operator(Debugger *debugger, Memories *memories, Instruction *instruction);

// TA 계산함.
static uint32_t calculate_TA(Instruction* instruction, Registers* registers);

// Addressing Mode 계산함
static ADDRESSING_MODE calculate_addressing_mode(Instruction* instruction, bool jump_op);

// reg_id 로 레지스터를 찾아서 리턴함.
static uint32_t *get_reg_by_id(Registers *registers, int reg_id);

// BP 를 핸들링함.
static bool handling_bp(Debugger *debugger, int instruction_size);

// 메모리로부터 값을 가져옴.
static bool load_from_memory(Debugger *debugger, Memories *memories, Instruction *instruction, uint32_t
*value,
                        size_t bytes, bool jump_op);

// 메모리에 값을 저장함.
static bool store_to_memory(Debugger *debugger, Memories *memories, Instruction *instruction, uint32_t value,
                        size_t bytes);

// 레지터로부터 값을 가져옴.
static bool load_from_register(Debugger *debugger, int reg_id, uint32_t *val);

// 레지스터에 값을 저장함.

```

```
static bool store_to_register(Debugger *debugger, int reg_id, uint32_t val);
```

```
/*
```

```
 * 함수 상세 설명(주석)은 declaration 파트 참고
```

```
 * Function Definitions (public)
```

```
*/
```

```
Debugger* construct_debugger(){  
    Debugger* debugger = (Debugger*)malloc(sizeof(*debugger));
```

```
    debugger->start_address = 0; // default: 0  
    debugger->bp_count = 0;      // default: 0  
    debugger->run_count = 0; // default: 0  
    debugger->registers = construct_registers();  
    debugger->previous_bp = -1;  
    debugger->end_address = MAX_BP_NUM - 1;  
    return debugger;  
}
```

```
bool destroy_debugger(Debugger** debugger){  
    destroy_registers(&((*debugger)->registers));  
    free(*debugger);  
    return true;  
}
```

```
Registers* construct_registers(){  
    Registers* registers = (Registers*)malloc(sizeof(*registers));  
    registers->A = 0;  
    registers->B = 0;  
    registers->L = 0;  
    registers->PC = 0;  
    registers->S = 0;  
    registers->T = 0;  
    registers->X = 0;  
    registers->SW = 0;  
  
    return registers;  
}
```

```
bool destroy_registers(Registers** registers){  
    free(*registers);  
    return true;  
}
```

```
void reset_registers(Registers* registers){  
    memset(registers, 0, sizeof(*registers));  
    registers->L = 0x00FFFFFF;  
}
```

```
bool loader_linker(Debugger *debugger, Memories *memories){  
    assert(debugger->filenames[0]);  
    assert(debugger->file_count > 0);
```

```

bool status;

status = loader_linker_pass1(debugger);
if(!status) return false;

status = loader_linker_pass2(debugger, memories);
if(!status) return false;

print_load_infos(debugger->load_infos);

destroy_symbol_table(&(debugger->estab));

return true;
}

bool run(Debugger *debugger, Memories *memories){
    Registers* registers = debugger->registers;
    uint32_t tmp;
    bool is_continue = debugger->is_running;

    if(!debugger->is_running){
        tmp = registers->PC;
        reset_registers(registers);
        registers->PC = tmp;
    }

    debugger->is_running = true;

    while (registers->PC != 0x00FFFFFF){
        uint8_t opcode;
        uint32_t instruction_val = 0;
        uint8_t memory_val;
        int instruction_size;

        enum op_format format;
        memory_val = (uint8_t)memories->data[registers->PC].value;

        instruction_val = memory_val;
        opcode = memory_val & 0xFC;
        format = op_format_by_op_num(opcode);

        switch (format){
            case OP_FORMAT_1:
                instruction_size = 1;
                break;
            case OP_FORMAT_2_GEN:
                memory_val = (unsigned char)memories->data[registers->PC + 1].value;
                instruction_size = 2;
                instruction_val = (instruction_val << 8) + memory_val;
                break;
        }
    }
}

```

```

case OP_FORMAT_3_4_GEN:
    memory_val = (unsigned char)memories->data[registers->PC + 1].value;
    instruction_val = (instruction_val << 8) + memory_val;
    memory_val = (unsigned char)memories->data[registers->PC + 2].value;
    instruction_val = (instruction_val << 8) + memory_val;
    instruction_size = 3;
    if (instruction_val & (1 << 12)){
        memory_val = (unsigned char)memories->data[registers->PC + 3].value;
        instruction_val = (instruction_val << 8) + memory_val;
        ++instruction_size;
    }
    break;
default:
    return false;
}

if (!is_continue){
    bool is_break = handling_bp(debugger, instruction_size);
    if(is_break) return true;
}

Instruction instruction;
instruction.opcode = opcode;
instruction.extend = (instruction_size == 4);
instruction.param.param.val = instruction_val;

bool status;
registers->PC += instruction_size;

status = execute_operator(debugger, memories, &instruction);
if(!status) return false;
is_continue = false;
}

registers->PC = debugger->start_address + (uint32_t)debugger->load_infos->list[0].length;
print_registers(registers);

debugger->previous_bp = -1;
printf("End program\n");

debugger->is_running = false;
return true;
}

/*
 * 함수 상세 설명(주석)은 declaration 파트 참고
 * Static Function Definitions
 */
static bool loader_linker_pass1(Debugger *debugger) {
    debugger->estab = construct_symbol_table();

```

```

debugger->load_infos = construct_load_info_list();

int csaddr = (int)debugger->start_address;

for(int i = 0; i < debugger->file_count; i++){
    int status;
    status = loader_linker_pass1_one(debugger, i, &csaddr);
    if(!status) return false;
}

return true;
}

static bool loader_linker_pass1_one(Debugger *debugger, int file_num, int *csaddr) {
    assert(file_num >= 0 && file_num <= 2);
    assert(debugger->filenames[file_num]);

    FILE *fp;
    char buf[1010];
    bool is_header = false;
    int base_address = *csaddr;

    fp = fopen (debugger->filenames[file_num], "rt");
    if(!fp){
        fprintf(stderr, "[ERROR] Can't file open\n");
        return false;
    }

    while (fgets (buf, 1000, fp) != NULL){
        if(buf[0] != 'H' && buf[0] != 'D') continue;
        if (buf[0] == 'H'){

            if (is_header) return false;
            else is_header = true;

            int tmp;

            LoadInfoList* load_infos = debugger->load_infos;
            load_infos->list[load_infos->count].type = INFO_TYPE_CONTROL_SECTION;

            sscanf (buf, "H%6s%06X%06X",
                    load_infos->list[load_infos->count].name,
                    &tmp, &load_infos->list[load_infos->count].length);
            load_infos->list[load_infos->count].addr = base_address;

            *csaddr += load_infos->list[load_infos->count].length;
            load_infos->count += 1;

            continue;
        }
        if (buf[0] == 'D'){

```



```

int rec_len = strlen(buf);
rec_len -= 2;
if (rec_len % 12 != 0) return false;
LoadInfoList* load_infos = debugger->load_infos;

for (int i = 0; i < rec_len / 12; i++){
    int offset;
    Symbol* symbol = construct_symbol();

    load_infos->list[load_infos->count].type = INFO_TYPE_SYMBOL;

    sscanf (buf + 1 + i * 12,
            "%6s%06X",
            load_infos->list[load_infos->count].name,
            &offset);
    load_infos->list[load_infos->count].addr = base_address + offset;

    strncpy (symbol->label,
            load_infos->list[load_infos->count].name,
            10);

    symbol->location_counter = load_infos->list[load_infos->count].addr;

    insert_symbol(debugger->estab, symbol);

    load_infos->count += 1;
}
continue;
}
}

if(!is_header){
    fprintf(stderr, "[ERROR] Object file is wrong\n");
    return false;
}

fclose(fp);

return true;
}

static bool loader_linker_pass2(Debugger *debugger, Memories *memories){
    int csaddr = (int)debugger->start_address;

    for(int i = 0; i < debugger->file_count; i++){
        int status;
        status = loader_linker_pass2_one(debugger, memories, i, &csaddr);
        if(!status) return false;
    }

    return true;
}

```

```

}

static bool loader_linker_pass2_one(Debugger *debugger, Memories *memories, int file_num, int *csaddr){
    assert(file_num >= 0 && file_num <= 2);
    assert(debugger->filenames[file_num]);

    FILE *fp;

    fp = fopen (debugger->filenames[file_num], "rt");
    if(!fp){
        fprintf(stderr, "[ERROR] Can't file open\n");
        return false;
    }

    char ch_stat[128] = {};
    ch_stat['H'] = 0;
    ch_stat['R'] = 0;
    ch_stat['T'] = 1;
    ch_stat['M'] = 2;

    char buf[1001];
    int state = 0;
    int num_to_address[256];
    bool num_validation[256] = { false, };
    int base_addr = *csaddr;

    num_validation[1] = true;
    num_to_address[1] = base_addr;

    while (fgets (buf, 1000, fp) != NULL){
        if(buf[0] != 'H' && buf[0] != 'R' && buf[0] != 'T' && buf[0] != 'M')
            continue;

        if(state > ch_stat[(unsigned char)buf[0]]) return false;
        state = ch_stat[(unsigned char)buf[0]];

        if (buf[0] == 'H'){
            char name[8];
            unsigned int hex, length;

            sscanf (buf, "H%6s%06X%06X", name, &hex, &length);
            *csaddr += length;
            continue;
        }
        if (buf[0] == 'R'){
            int num;
            char label[8];
            int cnt;
            char *ptr = buf + 1;

            while (sscanf (ptr, "%02X%6s%n", &num, label, &cnt) > 0){

```

```

    ptr += cnt;

    const Symbol *symbol = find_symbol_by_name (debugger->estab, label);
    if (!symbol) return false;

    num_to_address[num] = symbol->location_counter;
    num_validation[num] = true;
}
continue;
}
if (buf[0] == 'T'){

    int start_offset;
    int length;

    sscanf (buf, "T%06X%02X", &start_offset, &length);
    for (int i = 0; i < length; ++i){
        unsigned int val_c;

        sscanf (buf + 9 + i * 2, "%02X", &val_c);
        edit_memory (memories, base_addr + start_offset + i, val_c);
    }
    continue;
}
if (buf[0] == 'M'){

    int offset;
    int length, num = 0xFFFFFFFF;
    char op_char = 0;

    sscanf (buf, "M%06X%02X%c%02X", &offset, &length, &op_char, &num);

    if ((unsigned)num == 0xFFFFFFFF){
        op_char = '+';
        num = 0x01;
    }

    if (!num_validation[num]) return false;

    const unsigned int old_value = num_to_address[num];
    unsigned int new_value = 0;

    for (int i = 0; i < 3; ++i){
        new_value <<= 8;
        new_value += (unsigned char)memories->data[base_addr+offset+i].value;
    }

    const int sum_new_old_value = new_value + old_value;
    const int sub_new_old_value = new_value - old_value;

    if (length == 5){

```

```

        if (op_char == '+')
            new_value = (sum_new_old_value & 0xFFFFF) + ((new_value >> 20) << 20);
        else if (op_char == '-')
            new_value = (sub_new_old_value & 0xFFFFF) + ((new_value >> 20) << 20);
        else return false;
    }else if (length == 6){
        if (op_char == '+') new_value = sum_new_old_value;
        else if (op_char == '-') new_value = sub_new_old_value;
        else return false;
    }

    edit_memory (memories, base_addr + offset, (unsigned char)(new_value >> 16));
    edit_memory (memories, base_addr + offset + 1, (unsigned char)(new_value >> 8));
    edit_memory (memories, base_addr + offset + 2, (unsigned char)new_value);
}
}

fclose(fp);
return true;
}

static LoadInfoList* construct_load_info_list(){
    LoadInfoList* loadInfoList;
    loadInfoList = (LoadInfoList*)malloc(sizeof(LoadInfoList));
    loadInfoList->count = 0;
    return loadInfoList;
}

static void print_load_infos(LoadInfoList *load_infos){
    unsigned int total_length = 0;
    const int count = load_infos->count;

    printf ("%s %s %s %s\n", "Control", "Symbol", "Address", "Length");
    printf ("%s %s %s %s\n", "Section", "Name", "", "");
    printf ("-----\n");
    for(int i=0; i<count; i++){
        const LoadInfoNode node = load_infos->list[i];
        if (node.type == INFO_TYPE_CONTROL_SECTION){
            printf ("%s %c %04X%11c %04X%11c\n",
                    node.name,
                    '\'', node.addr,
                    '\'', node.length, '\');
            total_length += node.length;
            continue;
        }
        if (node.type == INFO_TYPE_SYMBOL){
            printf ("%c %s %04X%11c %s\n", '\'',
                    node.name, node.addr,
                    '\'', '\');
        }
    }
}

```

```

printf("-----\n");
printf("%-15c %-15c %-15s %04X%11c\n", ' ', ' ', "Total Length", total_length, ' ');
}

static bool execute_operator(Debugger *debugger, Memories *memories, Instruction *instruction) {
    assert(debugger);
    assert(memories);
    assert(instruction);

    Registers* registers = debugger->registers;
    uint32_t value;
    uint32_t val1, val2;
    static size_t device_input_idx = 0;
    char inputDevice[] = " I am Test Device\0\0"; // virtual input device

    switch (instruction->opcode){
        case LDA:
            load_from_memory(debugger, memories, instruction, &registers->A, 3, false);
            break;
        case LDB:
            load_from_memory(debugger, memories, instruction, &registers->B, 3, false);
            break;
        case LDT:
            load_from_memory(debugger, memories, instruction, &registers->T, 3, false);
            break;
        case LDX:
            load_from_memory(debugger, memories, instruction, &registers->X, 3, false);
            break;
        case LDCH:
            load_from_memory(debugger, memories, instruction, &value, 1, false);
            registers->A = (registers->A & 0xFFFFF00) + (value & 0xFF);
            break;
        case STA:
            store_to_memory(debugger, memories, instruction, registers->A, 3);
            break;
        case STL:
            store_to_memory(debugger, memories, instruction, registers->L, 3);
            break;
        case STX:
            store_to_memory(debugger, memories, instruction, registers->X, 3);
            break;
        case STCH:
            store_to_memory(debugger, memories, instruction, registers->A & 0xFF, 1);
            break;
        case JSUB:
            load_from_memory(debugger, memories, instruction, &value, 3, true);
            registers->L = registers->PC;
            registers->PC = value;
            break;
        case JEQ:
            if (registers->SW == 0) load_from_memory(debugger, memories, instruction, &registers->PC, 3, true);
    }
}

```

```

    break;
case JGT:
    if ((int) registers->SW > 0) load_from_memory(debugger, memories, instruction, &registers->PC, 3, true);
    break;
case JLT:
    if ((int) registers->SW < 0) load_from_memory(debugger, memories, instruction, &registers->PC, 3, true);
    break;
case J:
    load_from_memory(debugger, memories, instruction, &registers->PC, 3, true);
    break;
case COMP:
    load_from_memory(debugger, memories, instruction, &value, 3, false);
    if (registers->A > value)
        registers->SW = 1;
    else if (registers->A < value)
        registers->SW = -1;
    else
        registers->SW = 0;
    break;
case TD:
    registers->SW = 1;
    break;
case RD:
    registers->A = (registers->A & 0FFFFFF00) + (unsigned char)inputDevice[device_input_idx++];
    if (device_input_idx >= sizeof(inputDevice) / sizeof(char))
        device_input_idx = 0;
    break;
case RSUB:
    registers->PC = registers->L;
    break;
case WD:
    // is done!
    break;
case CLEAR:
    store_to_register(debugger, instruction->param.param.p2.r1, 0);
    break;
case COMPR:
    load_from_register(debugger, instruction->param.param.p2.r1, &val1);
    load_from_register(debugger, instruction->param.param.p2.r2, &val2);
    if (val1 > val2) registers->SW = 1;
    else if (val1 < val2) registers->SW = -1;
    else registers->SW = 0;
    break;
case TIXR:
    load_from_register(debugger, instruction->param.param.p2.r1, &value);
    ++registers->X;

    if (registers->X > value) registers->SW = 1;
    else if (registers->X < value) registers->SW = -1;
    else registers->SW = 0;
    break;

```

```

        default:
            return false;
    }

    return true;
}

static uint32_t calculate_TA(Instruction* instruction, Registers* registers){
    uint32_t TA;

    uint32_t b = (instruction->extend ? instruction->param.param.p4.b : instruction->param.param.p3.b);
    uint32_t p = (instruction->extend ? instruction->param.param.p4.p : instruction->param.param.p3.p);
    uint32_t address = (instruction->extend ? instruction->param.param.p4.address : instruction-
>param.param.p3.address);
    uint32_t x = (instruction->extend ? instruction->param.param.p4.x : instruction->param.param.p3.x);

    if (b == 1 && p == 0){
        // Base Relative
        TA = address + registers->B;
    } else if (b == 0 && p == 1){
        // PC Relative
        int32_t val;
        uint32_t bound;

        if(instruction->extend) bound = (1 << 19);
        else bound = (1 << 11);

        if(address >= bound) val = address - (bound << 1);
        else val = address;

        TA = registers->PC + val;
    } else {
        TA = address;
    }

    if (x == 1){
        TA += registers->X;
    }

    return TA;
}

static bool handling_bp(Debugger *debugger, int instruction_size) {
    bool is_break = false;
    unsigned int bp;

    for(unsigned int i = debugger->registers->PC; i < debugger->registers->PC + instruction_size; i++){
        if(debugger->break_points[i]){
            if(debugger->previous_bp != -1 && i == (unsigned int)debugger->previous_bp)
                continue;
            is_break = true;
            bp = i;
            debugger->previous_bp = bp;
        }
    }
}

```

```

        break;
    }
}
if(is_break){
    print_registers(debugger->registers);
    printf("Stop at checkpoint[%04X]\n", bp);
    return true;
}
return false;
}

static ADDRESSING_MODE calculate_addressing_mode(Instruction* instruction, bool jump_op){
    uint32_t n = (instruction->extend ? instruction->param.param.p4.n : instruction->param.param.p3.n);
    uint32_t i = (instruction->extend ? instruction->param.param.p4.i : instruction->param.param.p3.i);

    if(jump_op){
        if(n == 1 && i == 1) return ENUM_IMMEDIATE_ADDRESSING;
        if(n == 1 && i == 0) return ENUM_SIMPLE_ADDRESSING;
        return ENUM_ADDRESSING_ERROR;
    }

    if(n == 0 && i == 1) return ENUM_IMMEDIATE_ADDRESSING;
    if(n == 1 && i == 1) return ENUM_SIMPLE_ADDRESSING;
    if(n == 1 && i == 0) return ENUM_INDIRECT_ADDRESSING;

    return ENUM_ADDRESSING_ERROR;
}

static bool load_from_memory(Debugger *debugger, Memories *memories, Instruction *instruction, uint32_t
*value,
                        size_t bytes, bool jump_op){
    uint32_t TA;
    ADDRESSING_MODE addr_mode;

    TA = calculate_TA(instruction, debugger->registers);
    addr_mode = calculate_addressing_mode(instruction, jump_op);

    if(addr_mode == ENUM_IMMEDIATE_ADDRESSING){
        *value = TA;
        return true;
    }

    if(addr_mode == ENUM_SIMPLE_ADDRESSING){
        uint8_t memory_val;
        *value = 0;
        for(size_t k = 0; k < bytes; ++k){
            memory_val = (uint8_t)memories->data[TA + k].value;

            *value = (*value << 8) + memory_val;
        }
        return true;
    }
}

```



```

if (addr_mode == ENUM_INDIRECT_ADDRESSING) {
    uint32_t indirect_address = 0;
    uint8_t memory_val;

    for (int k = 0; k < 3; ++k){
        memory_val = (uint8_t)memories->data[TA + k].value;
        indirect_address = (indirect_address << 8) + memory_val;
    }

    *value = 0;
    for (size_t k = 0; k < bytes; ++k){
        memory_val = (uint8_t)memories->data[indirect_address + k].value;
        *value = (*value << 8) + memory_val;
    }
    return true;
}

return false;
}

static bool store_to_memory(Debugger *debugger, Memories *memories, Instruction *instruction, uint32_t value,
                           size_t bytes){
    uint32_t TA;
    ADDRESSING_MODE addr_mode;

    TA = calculate_TA(instruction, debugger->registers);
    addr_mode = calculate_addressing_mode(instruction, false);

    if(addr_mode == ENUM_IMMEDIATE_ADDRESSING){
        return false;
    }

    if(addr_mode == ENUM_SIMPLE_ADDRESSING){
        for (int k = bytes-1; k >= 0; --k){
            edit_memory(memories, TA + k, (uint8_t)value);
            value >>= 8;
        }
        return true;
    }

    if(addr_mode == ENUM_INDIRECT_ADDRESSING){
        uint8_t mem_val;
        uint32_t addr = 0;
        for (int k = 0; k < 3; ++k){
            mem_val = (uint8_t)memories->data[TA + k].value;
            addr = (addr << 8) + mem_val;
        }
        for (int k = 2; k >= 0; --k){
            edit_memory(memories, addr + k, (uint8_t)value);
            value >>= 8;
        }
    }
}

```

```

    }
    return true;
}

return false;
}

static bool load_from_register(Debugger *debugger, int reg_id, uint32_t *val){
    uint32_t* reg = get_reg_by_id(debugger->registers, reg_id);
    if(reg == NULL) return false;

    *val = *reg;

    return true;
}

static uint32_t *get_reg_by_id(Registers *registers, int reg_id) {
    if(reg_id == 6 || reg_id == 7) return NULL;
    if(reg_id < 0 || reg_id > 9) return NULL;

    uint32_t* regs[10] = {
        [0] = &registers->A,
        [1] = &registers->X,
        [2] = &registers->L,
        [3] = &registers->B,
        [4] = &registers->S,
        [5] = &registers->T,
        [6] = NULL,
        [7] = NULL,
        [8] = &registers->PC,
        [9] = &registers->SW
    };

    return regs[reg_id];
}

static bool store_to_register(Debugger *debugger, int reg_id, uint32_t val){
    uint32_t* reg = get_reg_by_id(debugger->registers, reg_id);
    if(reg == NULL) return false;

    *reg = val;
    return true;
}

static void print_registers(Registers* registers){
    printf (
        "A : %06X X : %06X \n"
        "L : %06X PC: %06X \n"
        "B : %06X S : %06X \n"
        "T : %06X      \n",
        registers->A, registers->X,
        registers->L, registers->PC,
        registers->B, registers->S,
        registers->T
    );
}

```

```
);  
}
```

5.32 debug.h

```
#ifndef __DEBUG_H__  
#define __DEBUG_H__  
  
#include <stdlib.h>  
#include <stdbool.h>  
#include <stdint.h>  
#include "memory.h"  
#include "symbol.h"  
#include "opcode.h"  
  
#include <string.h>  
#define MAX_BP_NUM (1024 * 1024) // 1MB  
  
// 레지스터 값을 저장함.  
typedef struct registers {  
    uint32_t A;  
    uint32_t L;  
    uint32_t X;  
    uint32_t PC;  
    uint32_t B;  
    uint32_t S;  
    uint32_t T;  
    uint32_t SW;  
} Registers;  
  
// load info 정보의 유형  
enum load_info_type {  
    INFO_TYPE_CONTROL_SECTION,  
    INFO_TYPE_SYMBOL  
} LoadInfoType;  
  
// load info 정보 한 단위  
typedef struct load_info_node {  
    enum load_info_type type;  
    char name[15];  
    int addr;  
    int length;  
} LoadInfoNode;  
  
// load 되는 정보를 저장함.  
typedef struct load_info_list {  
    LoadInfoNode list[1001];  
    int count;  
} LoadInfoList;
```

```

// load, run, bp, 레지스터 관리 등의 역할을 함.
typedef struct debugger {
    uint32_t start_address; // 10 진수로 변환하여 저장함.
    int end_address;
    bool break_points[MAX_BP_NUM]; // 해당 셀에 브레이크포인트가 걸렸는지 안걸렸는지 여부 저장
    int bp_count; // 브레이크 포인트 개수
    Registers* registers; // 레지스터 정보
    int run_count; // run 실행 횟수.
    int previous_bp; // 이전 bp 주소

    char* filenames[3]; // 파일 이름들
    int file_count; // 파일 개수

    SymbolTable* estab;
    LoadInfoList* load_infos;
    uint32_t total_length;
    bool is_running;
    bool is_loaded;
} Debugger;

// debugger 의 생성자 함수
Debugger* construct_debugger();

// debugger 의 소멸자 함수
bool destroy_debugger(Debugger** debugger);

// registers 의 생성자 함수
Registers* construct_registers();

// registers 의 소멸자 함수
bool destroy_registers(Registers** registers);

// register 를 리셋함.
void reset_registers(Registers* registers);

// loader 명령을 실행하는 함수
bool loader_linker(Debugger *debugger, Memories *memories);

// run 명령을 실행하는 함수
bool run(Debugger *debugger, Memories *memories);

#endif

```

5.33 Makefile

SOURCES=20161631.c command.c command_mapping.c command_shell.c command_execute.c history.c
memory.c state.c util.c dir.c command_validate_util.c opcode.c symbol.c assemble.c debug.c

TARGET=20161631.out

all: \$(TARGET)

\$(TARGET): \$(SOURCES)
gcc -std=gnu99 -W -Wall \$(SOURCES) -o \$(TARGET)

clean:
rm -f \$(TARGET)