1. 동기/목적

inst_table.txt를 통해 inst table을 초기화

소스 코드 input.txt파일을 입력으로 받아서 token 테이블을 초기화

output.txt로 opcode와 함께 출력

어셈블러의 모든 과정을 이해하기 위해 파싱과 토큰화 과정을 진행하고

pass1의 동작을 간단하게 구현해보는 과제

설계/구현 아이디어

사실 opcode만 출력하는 과제이기 때문에

파싱에서 operator를 찾고 token table의 필요한 부분만 초기화 하면 되지만

최종적으로 어셈블러를 구현하는 과제의 연습이라고 생각해서 token table 초기화에 집중했습니다.

하지만 추후에 다른 input이 들어올 수 있기 때문에 예외처리를 타이트하게 하지 않았습니다.

예시로 현재 input.txt에 나와있는 EXTDEF, EXTREF, LTORG 말고도

op table에 없는 operator가 들어올 수 있다고 생각하여 포맷에 맞다면 순서대로 token을 초기화 합니다.

inst_table input 형식은

str | opcode | format | ops

입니다. 모든 화이트스페이스는 구분자로 적용됩니다.

iniput file의 첫번째 문자가 white space가 아니라면 label이 있다고 가정합니다.

반대로 white space로 시작한다면 label은 NULL이 됩니다.

ex)

comment

space | operator

label | operator | (comment)

label | operator | operands | (comment)

space | operator | operands | (comment)

.

.

이에 따라 operator token을 초기화 하면

1. only comment
2. EXTDEF, EXTREF
3. OP table에 있는 operator
4. OP table에 없는 operator

크게 4가지 경우로 나뉩니다.

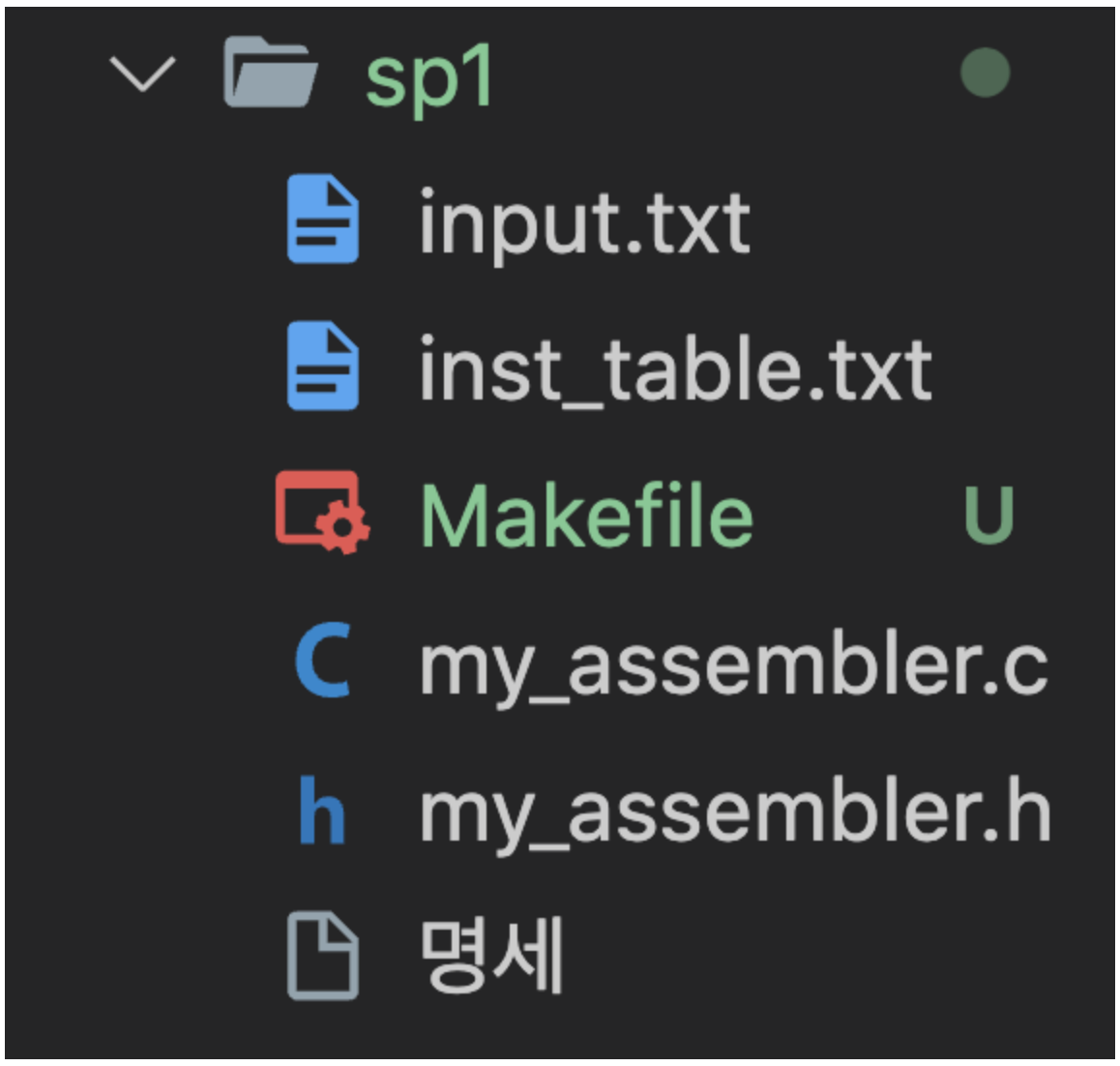
위에서 언급했듯이 현재는 4번의 경우도 초기화를 진행하고 있습니다.

그렇게 토큰을 모두 초기화 하면

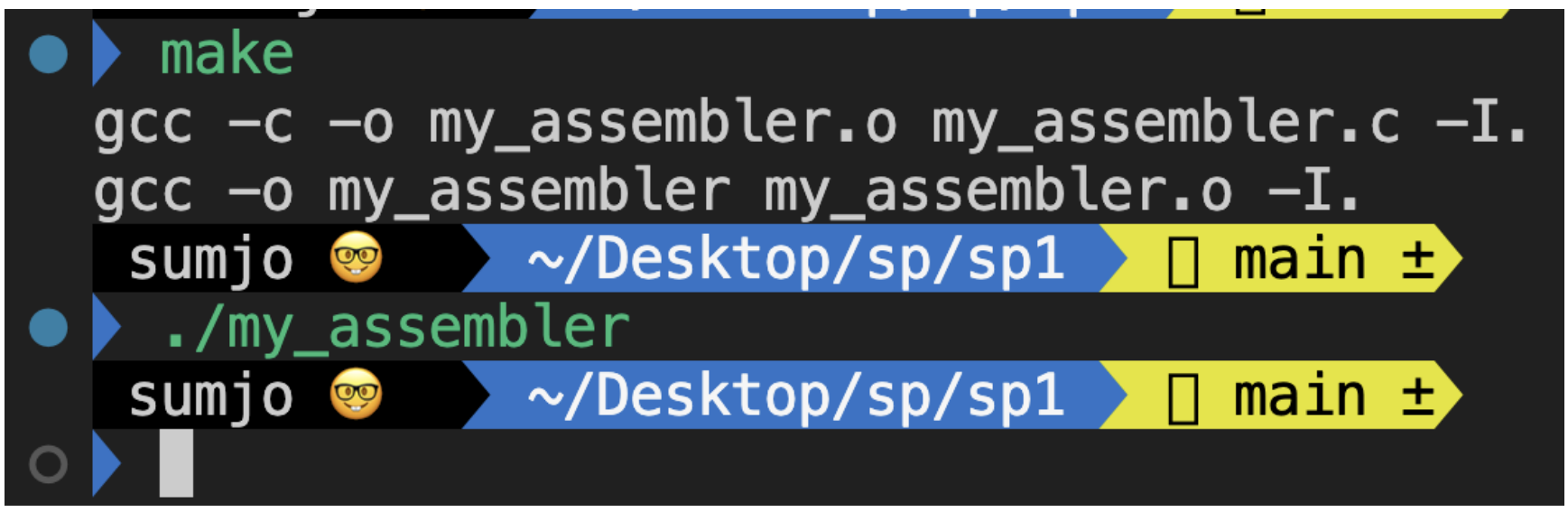
output_file을 만들어주는 과정에서 comment를 제외한 모든 토큰 요소를 출력하고

op table에서 찾을 수 있는 operator는 op code도 함께 출력합니다.

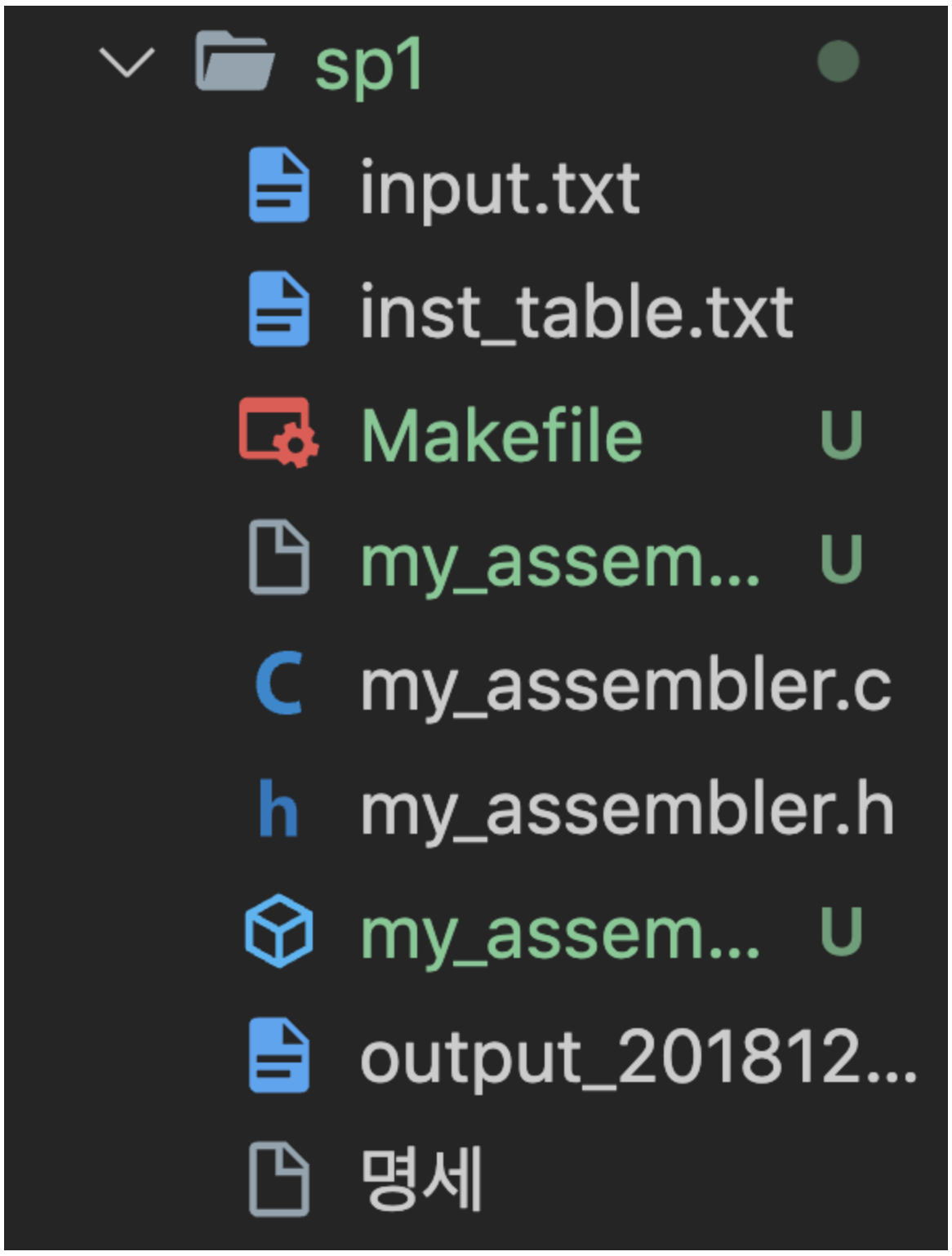
수행 결과



Directory



Makefile을 통해 컴파일 후 실행



output_20181259 파일 생성됨

sp1 > output_20181259.txt

```
1   COPY        START     0
2               EXTDEF    BUFFER,BUFEND,LENGTH
3               EXTREF    RDREC,WRREC
4   FIRST       STL       RETADR          14
5   CLOOP       +JSUB     RDREC           48
6               LDA       LENGTH          00
7               COMP      #0              28
8               JEQ       ENDFIL          30
9               +JSUB     WRREC           48
10              J         CLOOP           3C
11  ENDFIL      LDA       =C'EOF'         00
12              STA       BUFFER          0C
13              LDA       #3              00
14              STA       LENGTH          0C
15              +JSUB     WRREC           48
16              J         @RETADR         3C
17  RETADR      RESW      1
18  LENGTH      RESW      1
19              LTORG
20  BUFFER      RESB      4096
21  BUFEND      EQU       *
22  MAXLEN      EQU       BUFEND-BUFFER
23  RDREC       CSECT
24  .
25  . SUBROUTINE TO READ RECORD INTO BUFFER
26  .
27              EXTREF    BUFFER,LENGTH,BUFEND
28              CLEAR     X               B4
29              CLEAR     A               B4
30              CLEAR     S               B4
31              LDT       MAXLEN          74
32  RLOOP       TD        INPUT           E0
33              JEQ       RLOOP           30
34              RD        INPUT           D8
35              COMPR     A,S             A0
36              JEQ       EXIT            30
37              +STCH     BUFFER,X        54
38              TIXR      T               B8
39              JLT       RLOOP           38
40  EXIT        +STX      LENGTH          10
41              RSUB                      4C
42  INPUT       BYTE      X'F1'
43  MAXLEN      WORD      BUFEND-BUFFER
44  WRREC       CSECT
45  .
46  . SUBROUTINE TO WRITE RECORD FROM BUFFER
47  .
48              EXTREF    LENGTH,BUFFER
49              CLEAR     X               B4
50              +LDT      LENGTH          74
51  WLOOP       TD        =X'05'          E0
52              JEQ       WLOOP           30
53              +LDCH     BUFFER,X        50
54              WD        =X'05'          DC
55              TIXR      T               B8
56              JLT       WLOOP           38
57              RSUB                      4C
58              END       FIRST
59
```

output_20181259.txt

```
❯ ./my_assembler
token_parsing: operand가 없는 input JEQ 있습니다.
assem_pass1: 토큰 파싱에 실패했습니다.
assem_pass1: 패스1 과정에서 실패했습니다. (error_code: -1)
```

error

```
❯ ./my_assembler
warning: optable에 없는 명령어 HELLO가 들어왔습니다.
```

error

결론 및 보충할 점

단순히 opcode와 함께 출력하는 과제라기 보다는

추후에 2 way assembler 구현을 위해 1pass의 진행과정을 이해하기 위한 과제라고 생각이 들었습니다.

하지만 파싱 과정에서 아직 충분히 많은 예외사항이 있을 것이라고 생각합니다.

input 포맷의 형식이 조금 더 디테일하게 명시된다면 그에 맞추어 파싱할 수 있을 것 같습니다.

특히 op table에 없는 operator(혹은 지시어)의 범위가 아직 정확하지 않은 것 같습니다.

이를테면 현재 input에 있는 LTORG, EXTDEF등등을 제외하고 opcode에 없는 명령어는 예외처리를 해야할 지, 혹은 교재를 참고하여 모든 경우를 처리해야 하는 것인지 고민해봐야 할 것 같습니다.

또한 함수의 역할 분리가 미흡했습니다. Token 부분이 다소 길게 작성되었습니다. 재사용 가능한 부분은 분리하여 함수를 작성해야 할 것 같습니다.

디버깅


변수
Locals
err: -1
temp: 0x00000001309041e0
len: 4
input: 0x00000001308062c0 "\tJEQ\tENDFIL\tEXIT IF EOF FOUND\n"
token: 0x00000001309041a0
inst_table: 0x000000016fdfe598
inst_table_length: 59
Registers
Other Registers
CPU
IEEE Single
IEEE Double
조사식
token->operator: 0x0000000130904210 "JEQ"
*$26: 74 'J'
token->operand: [3]
[0]: 0x0000000130904220 "ENDFIL"
*[0]: 69 'E'
[1]: 0x0000000000000000
[2]: 0x0000000000000000
token->label: 0x0000000000000000
*$29: ??

```
변수
  Locals
    err: -1
  > temp: 0x00000001309043f0
    len: 2
  > input: 0x0000000130806390 "\t+JSUB\tWRREC\tWRITE EOF\n"
  > token: 0x00000001309043b0
  > inst_table: 0x000000016fdfe598
    inst_table_length: 59
  Registers
  > Other Registers
  > CPU
  > IEEE Single
  > IEEE Double

조사식
  token->operator: 0x0000000130904410 "+JSUB"
    *$64: 43 '+'
  token->operand: [3]
    [0]: 0x0000000130904420 "WRREC"
      *[0]: 87 'W'
    [1]: 0x0000000000000000
    [2]: 0x0000000000000000
  token->label: 0x0000000000000000
    *$67: ??
  token->comment: 0x0000000130904450 "WRITE EOF"
    *$68: 87 'W'
```

소스코드

init_input.c

```
int init_input(char **input, int *input_length, const char *input_dir) {
    FILE *fp;
    int err;

    err = 0;
    fp = fopen(input_dir, "r");
    if (fp == NULL) {
        fprintf(stderr, "init_input: 소스코드 파일을 열 수 없습니다.\n");
        err = -1;
        return err;
    }

    int i = 0;
    char line[MAX_LINES];
    while (fgets(line, MAX_LINES, fp) != NULL) {
        input[i] = (char *)malloc(strlen(line) + 1);
        if (input[i] == NULL) {
            fprintf(stderr, "init_input: 메모리 할당에 실패했습니다.\n");
            exit(1);
        }
        strcpy(input[i], line);
        i++;
    }
    *input_length = i;
    return err;
}
```

token_parsing.c

```
int token_parsing(const char *input, token *token, const inst **inst_table, int inst_tabl

    token->label = NULL;
    token->operator = NULL;
    token->operand[0] = NULL;
    token->comment = NULL;

    int err ;
    char **temp = split((char *)input, &err);

    if(temp == NULL) {
        fprintf(stderr, "token_parsing: split 함수에서 에러가 발생했습니다.\n");
        return err;
    }

    err = -1;
    //라인의 길이
    int len = strings_len(temp);
    if(len == 0) {
        token->label = NULL;
        token->operator = NULL;
        token->operand[0] = NULL;
        token->comment = NULL;
        return 0;
    }

    // 라인의 첫 글자가 '.'이면 주석
    if (input[0] == '.') {
```

```
        token->label = NULL;
        token->operator = NULL;
        token->operand[0] = NULL;
        token->comment = (char *)malloc(calc_totallen(temp));
        if (token->comment == NULL) {
            fprintf(stderr, "token_parsing: 메모리 할당에 실패했습니다.\n");
            return -1;
        }

        for(int i = 0; i < len; i++) {
            strcat(token->comment, temp[i]);
            if(i != len - 1)
                strcat(token->comment, " ");
        }
    }
    else if (is_white_space(input[0])) {
    // 라인의 첫 글자가 공백이면 레이블이 없는 경우
            token->label = NULL;
            token->operator = temp[0];
                int opcode_index = search_opcode(temp[0], inst_table, inst_table_length);
                if (opcode_index == -1) {
                    // EXTDEF, EXTREF, LTORG
                    if(strcmp(token->operator, "EXTDEF") == 0 || strcmp(token->operator, "EX
                        temp++;
                        if(!temp)
                            return err;
                        len--;
                        char **operands = operand_split(temp[0]);
                        if(operands == NULL) {
                            fprintf(stderr, "token_parsing: operand_split 함수에서 에러가 발생했습
                            return -1;
                        }
                        for(int i = 0; i < strings_len(operands); i++) {
                            token->operand[i] = operands[i];
                        }
                    }
                    else if (strcmp(token->operator, "LTORG")== 0 || strcmp(token->operator,
                        token->operand[0] = NULL;
                        temp++;
                        if(!temp)
                            return 1;
                        len--;
                        if(temp[0]){
                            token->comment = (char *)malloc(calc_totallen(temp));
                            for(int i = 0; i < len; i++) {
                                strcat(token->comment, temp[i]);
                                if(i != len - 1)
                                    strcat(token->comment, " ");
                            }
                        }
                        return 1;
                    }
                    else {
                        temp++;
                        if(!temp)
                            return 1;
                        len--;
                        fprintf(stderr, "warning: optable에 없는 명령어 %s가 들어왔습니다.\n", tok
                        token->operand[0] = temp[0];
                        return 1;
```

```
                }
                temp++;
                if(!temp)
                    return 1;
                len--;
                token->comment = (char *)malloc(calc_totallen(temp));
                for(int i = 0; i < len; i++) {
                    strcat(token->comment, temp[i]);
                    if(i != len - 1)
                        strcat(token->comment, " ");
                }
                return 1;
            }
            temp++;
            if(!temp)
                return 1;
            len--;

            if (get_ops(opcode_index, inst_table) == 0) {
            // operator가 필요없는 명령어
                token->operand[0] = NULL;
                token->comment = (char *)malloc(calc_totallen(temp));
                for(int i = 0; i < len; i++) {
                    strcat(token->comment, temp[i]);
                    if(i != len - 1)
                        strcat(token->comment, " ");
                }
            }
            else if (get_ops(opcode_index, inst_table) == 1) {
            // operands가 하나 있는 경우
                if(!temp[0]){
                    fprintf(stderr, "token_parsing: operand가 없는 input %s 있습니다.\n", to
                    return -1;
                }
                token->operand[0] = temp[0];
                temp++;
                if(!temp)
                    return err;
                len--;
                token->comment = (char *)malloc(calc_totallen(temp));
                for(int i = 0; i < len; i++) {
                    strcat(token->comment, temp[i]);
                    if(i != len - 1)
                        strcat(token->comment, " ");
                }
            }
            else if (get_ops(opcode_index, inst_table) == 2) {
            // operands가 두개 있는 경우
                char **operands = operand_split(temp[0]);
                if(operands == NULL || operands[2] != NULL) {
                    fprintf(stderr, "token_parsing: operand_split 함수에서 에러가 발생했습니다
                    return -1;
                }
                token->operand[0] = operands[0];
                token->operand[1] = operands[1];
                token->operand[2] = NULL;
                temp++;
                if(!temp)
                    return err;
                len--;
```

```
                token->comment = (char *)malloc(calc_totallen(temp));
                for(int i = 0; i < len; i++) {
                    strcat(token->comment, temp[i]);
                    if(i != len - 1)
                        strcat(token->comment, " ");
                }
            }
    }
    else {
    // 라인의 첫 글자가 존재 즉 레이블이 있는 경우
        if(len == 1){
            fprintf(stderr, "token_parsing: 레이블만 존재하는 input이 있습니다.\n");
            return err;
        }
        token->label = temp[0];

        if(strcmp(temp[1], "START") == 0 || strcmp(temp[1], "END") == 0) {
            token->operator = temp[1];
            token->operand[0] = NULL;
            temp += 2;
            if(!temp)
                return err;
            len -= 2;
        }
        else if(strcmp(temp[1], "BYTE") == 0 || strcmp(temp[1], "WORD") == 0 || strcmp(te
            token->operator = temp[1];
            if(!temp[2]){
                fprintf(stderr, "token_parsing: operand가 없는 BYTE, WORD, RESB, RESW가 있습
                return err;
            }
            token->operand[0] = temp[2];
            temp += 3;
            if(!temp)
                return err;
            len -= 3;
        }
        else if(strcmp(temp[1], "EXTDEF") == 0 || strcmp(temp[1], "EXTREF") == 0) {
            token->operator = temp[1];
            if(!temp[2]){
                fprintf(stderr, "token_parsing: operand가 없는 EXTDEF, EXTREF가 있습니다.\n"
                return err;
            }
            char **operands = operand_split(temp[2]);
            if(operands == NULL) {
                fprintf(stderr, "token_parsing: operand_split 함수에서 에러가 발생했습니다.\n"
                return -1;
            }
            int i = 0;
            while (operands[i]) {
                token->operand[i] = operands[i];
                i++;
            }
            token->operand[i] = NULL;
            temp += 3;
            if(!temp)
                return err;
            len -= 3;
        }
        else if (strcmp(temp[1], "LTORG") == 0 || strcmp(temp[1], "CSECT") == 0) {
            token->operator = temp[1];
```

```
            token->operand[0] = NULL;
            temp += 2;
            if(!temp)
                return err;
            len -= 2;
        }
        else {
            token->operator = temp[1];
            int opcode_index = search_opcode(temp[1], inst_table, inst_table_length);
            temp += 2;
            if(!temp)
                return err;
            len -= 2;

            if (opcode_index == -1) {
            // opcode 테이블에 없는 경우
                fprintf(stderr, "warning: optable에 없는 명령어 %s가 들어왔습니다.\n", token->
                char **operands = operand_split(temp[0]);
                if(operands == NULL) {
                    return 1;
                }
                int i = 0;
                while (operands[i]) {
                    token->operand[i] = operands[i];
                    i++;
                }
                token->operand[i] = NULL;
                temp++;
                if(!temp)
                    return err;
                len--;
                token->comment = (char *)malloc(calc_totallen(temp));
                for(int i = 0; i < len; i++) {
                    strcat(token->comment, temp[i]);
                    if(i != len - 1)
                        strcat(token->comment, " ");
                }
            }
            else if (get_ops(opcode_index, inst_table) == 0) {
            // opcode 테이블에 있고 operands가 없는 경우
                token->operand[0] = NULL;
                token->comment = (char *)malloc(calc_totallen(temp));
                for(int i = 0; i < len; i++) {
                    strcat(token->comment, temp[i]);
                    if(i != len - 1)
                        strcat(token->comment, " ");
                }
            }
            else if (get_ops(opcode_index, inst_table) == 1) {
            // opcode 테이블에 있고 operands가 하나 있는 경우
                if(!temp[0]){
                    fprintf(stderr, "token_parsing: operand가 없는 input이 있습니다.\n");
                    return err;
                }
                token->operand[0] = temp[0];
                temp++;
                if(!temp)
                    return err;
                len--;
                token->comment = (char *)malloc(calc_totallen(temp));
```

```
                for(int i = 0; i < len; i++) {
                    strcat(token->comment, temp[i]);
                    if(i != len - 1)
                        strcat(token->comment, " ");
                }
            }
            else if (get_ops(opcode_index, inst_table) == 2) {
            // opcode 테이블에 있고 operands가 두개 있는 경우
                char **operands = operand_split(temp[0]);
                if(operands == NULL) {
                    fprintf(stderr, "token_parsing: operand_split 함수에서 에러가 발생했습니다
                    return -1;
                }
                if(operands[2] != NULL) {
                    fprintf(stderr, "token_parsing: operand가 2개 이상인 operator에 operand
                    return err;
                }
                int i = 0;
                while (operands[i]) {
                    token->operand[i] = operands[i];
                    i++;
                }
                token->operand[i] = NULL;
                temp++;
                if(!temp)
                    return err;
                len--;
                token->comment = (char *)malloc(calc_totallen(temp));
                for(int i = 0; i < len; i++) {
                    strcat(token->comment, temp[i]);
                    if(i != len - 1)
                        strcat(token->comment, " ");
                }
            }
        }
    }

    return 0;
}
```

assem_pass1

```
int assem_pass1(const inst **inst_table, int inst_table_length,
                const char **input, int input_length, token **tokens,
                int *tokens_length, symbol **symbol_table,
                int *symbol_table_length, literal **literal_table,
                int *literal_table_length) {

    *tokens_length = 0;
    int err = 0;
    // 한줄씩 토큰을 만들어 tokens에 저장
    for(int i = 0; i < input_length; i++) {
        tokens[i] = (token *)malloc(sizeof(token));
        err = token_parsing(input[i], tokens[i], inst_table, inst_table_length);
        if(err < 0) {
            fprintf(stderr, "assem_pass1: 토큰 파싱에 실패했습니다.\n");
            return err;
        }
        (*tokens_length)++;
```

```
    }

    return 0;
}
```

make_opcode_output.c

```
int make_opcode_output(const char *output_dir, const token **tokens,
                       int tokens_length, const inst **inst_table,
                       int inst_table_length) {
    FILE *fp;
    int err;

    err = 0;

    fp = fopen(output_dir, "w");
    if (fp == NULL) {
        fprintf(stderr, "make_opcode_output: 출력 파일을 열 수 없습니다.\n");
        err = -1;
        return err;
    }

    for (int i = 0; i < tokens_length; i++) {
    if(tokens[i]->label == NULL && tokens[i]->operator == NULL && tokens[i]->operand[0] =
        fprintf(fp, "%-10s\t", tokens[i]->comment);
        fprintf(fp, "\n");
        continue;
    }

    // 레이블 출력: 최대 10자리, 왼쪽 정렬
    fprintf(fp, "%-10s\t", tokens[i]->label ? tokens[i]->label : "");

    // 연산자 출력: 최대 8자리, 왼쪽 정렬
    fprintf(fp, "%-8s\t", tokens[i]->operator ? tokens[i]->operator : "");

    // 오퍼랜드 출력
    if (tokens[i]->operand[0]) {
        fprintf(fp, "%s", tokens[i]->operand[0]);
        for (int j = 1; j < 3 && tokens[i]->operand[j]; j++) {
            fprintf(fp, ",%s", tokens[i]->operand[j]);
        }
    }

    int space_size = 0;
    if(tokens[i]->operand[0])
        space_size = strlen(tokens[i]->operand[0]);
    if(tokens[i]->operand[1])
        space_size += strlen(tokens[i]->operand[1]) + 1;

    for(int j = 0; j < 15 - space_size; j++) {
        fprintf(fp, " ");
    }

    // opcode 출력
    if (tokens[i]->operator) {
        int s_op = search_opcode(tokens[i]->operator, inst_table, inst_table_length);
        if (s_op != -1) {
            fprintf(fp, "%02X", inst_table[s_op]->op);
```

```
        }
    }

    fprintf(fp, "\n");
}
    return err;
}
```

main.c

```
int main(int argc, char **argv) {
    // SIC/XE 머신의 instruction 정보를 저장하는 테이블이다.
    inst *inst_table[MAX_INST_TABLE_LENGTH];
    int inst_table_length;

    // 소스코드를 저장하는 테이블이다. 라인 단위 저장한다.
    char *input[MAX_INPUT_LINES];
    int input_length;

// 소스코드의 각 라인을 토큰으로 전환하여 저장한다.
token *tokens[MAX_INPUT_LINES];
int tokens_length;

// 소스코드 내의 심볼을 저장하는 테이블이다. 추후 과제에 사용 예정.
symbol *symbol_table[MAX_TABLE_LENGTH];
int symbol_table_length;

// 소스코드 내의 리터럴을 저장하는 테이블이다. 추후 과제에 사용 예정.
literal *literal_table[MAX_TABLE_LENGTH];
int literal_table_length;

// 오브젝트 코드를 저장하는 테이블이다. 추후 과제에 사용 예정.
char object_code[MAX_OBJECT_CODE_LENGTH][MAX_OBJECT_CODE_STRING];
int object_code_length;

int err = 0;

if ((err = init_inst_table(inst_table, &inst_table_length, "inst_table.txt")) < 0) {
    fprintf(stderr, "init_inst_table: 기계어 목록 초기화에 실패했습니다. (error_code: %d)\n
    return -1;
}


if ((err = init_input(input, &input_length, "input.txt")) < 0) {
    fprintf(stderr, "init_input: 소스코드 입력에 실패했습니다. (error_code: %d)\n", err);
    return -1;
}

if ((err = assem_pass1((const inst **)inst_table, inst_table_length,
                       (const char **)input, input_length, tokens,
                       &tokens_length, symbol_table, &symbol_table_length,
                       literal_table, &literal_table_length)) < 0) {
    fprintf(stderr, "assem_pass1: 패스1 과정에서 실패했습니다. (error_code: %d)\n", err);
    return -1;
}

if ((err = make_opcode_output("./output_20181259.txt",
                              (const token **)tokens,
                              tokens_length,
                              (const inst **)inst_table,
```

```
                                    inst_table_length)) < 0) {
        fprintf(stderr, "make_opcode_output: opcode 파일 출력 과정에서 실패했습니다. (error_coc
        return -1;
    }

    // 추후 프로젝트에서 사용되는 부분
    /*
    if ((err = make_symbol_table_output("symtab_00000000", (const symbol **)symbol_table,
        fprintf(stderr, "make_symbol_table_output: 심볼테이블 파일 출력 과정에서 실패했습니다. (e
        return -1;
    }

    if ((err = make_literal_table_output("littab_00000000", (const literal **)literal_tab
        fprintf(stderr, "make_literal_table_output: 리터럴테이블 파일 출력 과정에서 실패했습니다.
        return -1;
    }

    if ((err = assem_pass2((const token **)tokens, tokens_length,
                           (const symbol **)symbol_table, symbol_table_length,
                           (const literal **)literal_table, literal_table_length,
                           object_code, &object_code_length)) < 0) {
        fprintf(stderr, "assem_pass2: 패스2 과정에서 실패했습니다. (error_code: %d)\n", err);
        return -1;
    }

    if ((err = make_objectcode_output("output_00000000",
                                      (const char(*)[74])object_code,
                                      object_code_length)) < 0) {
        fprintf(stderr, "make_objectcode_output: 오브젝트코드 파일 출력 과정에서 실패했습니다. (e
        return -1;
    }
    */

    return 0;
}
```