숭실대학교 Soongsil University

프로그램 보고서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다.
3. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

교과목	시스템프로그래밍<나> 2025
프로젝트 명	Project #1
교과목 교수	최 재 영
제출인	컴퓨터학부(학과) 학번:20211397 성명:강동민 (출석번호:201)
제출일	2025년 4월 24일

차 례

1장 프로젝트 개요

1.1) 개발 배경 및 목적

본 프로젝트는 SIC/XE 머신을 위한 어셈블러(Assembler)를 구현하는 과제이다. 이전 과제(과제 6)에서는 어셈블리 코드의 파싱 기능만을 구현하였으며, 이번 프로젝트1에서는 이를 확장하여 PASS1과 PASS2를 포함한 어셈블리 전체 처리 과정을 구현하였다. 이를 통해 Symbol Table, Literal Table, Object Code를 생성하며, 실제 기계어 코드 파일을 생성할 수 있다.

2장 배경 지식

2.1) 주제에 관한 배경지식

어셈블러(Assembler)는 저수준 언어인 어셈블리 언어를 기계어(로 변환하는 프로그램이다. 어셈블리 언어는 사람이 이해하기 쉽도록 기계어 명령어를 의미 있는 기호로 표현한 것으로, 어셈블러는 이들을 해석하여 각 명령어에 대응하는 기계어 코드(opcode)를 생성한다.

SIC(Standard Instructional Computer) 및 그 확장인 SIC/XE는 교육용 가상 컴퓨터 아키텍처로, 구조가 단순하면서도 다양한 어셈블리어 개념(포맷, 주소 지정 방식, 연산자 등)을 포함하고 있다. SIC/XE는 기본 SIC보다 복잡한 주소 지정 방식과 확장된 명령어 셋을 지원한다.

2.2) 기술적 배경지식

SIC/XE 어셈블러는 전통적으로 2-pass 구조를 기반으로 한다.

▶ Pass 1:

- 각 명령어에 주소를 할당 (LOCCTR)
- 라벨이 나타나는 위치를 기록하여 심볼 테이블(Symbol Table) 및 리터럴 테이블(Literal table)구성
- 각 라인의 구조를 파싱하여 후처리용 토큰 테이블 생성

▶ Pass 2:

- 명령어와 피연산자를 참조하여 실제 기계어(Object code)로 변환
- 각 지시어(BYTE, WORD, RESW, LTORG 등)를 해석하여 데이터를 생성
- Object Code는 H/T/M/E 레코드로 구성하며, obj 파일 형태로 출력

▶ 주요 자료구조 및 알고리즘

- inst_table: 명령어 구조체를 저장하는 테이블. 명령어 구조체는 명령어 이름, 명령어 코드, 명령어 형식, 피연산자 개수 정보를 포함한다.
-token_table: 어셈블리 라인을 파싱하여 구조화된 토큰을 저장. 각 토큰은 label, operator, operand[], comment, nixbpe의 정보를 가지며 추가로 해당 명령어의 주소인 addr 정보를 저장하는 형태로 수정하였다.

-sym_table, literal_table: 각각 심볼과 리터럴을 주소와 함께 저장하는 테이블

-obj_codes[]: 실제 출력될 기계어 정보를 저장하는 _object_code 테이블. "_object_code"는 레코드 타입(H/T/M/E), 시작주소, 길이, 실제 object 코드 정보를 가진다.

3장 시스템 설계 내용

3.1) 전체 시스템 설계 내용

본 프로젝트에서 구현한 SIC/XE 어셈블러는 크게 초기화, 1차 분석(PASS1), 2차 처리(PASS2), 출력의 네 단계로 구성된다. 각 단계는 독립된 함수로 구성되어 있으며, 모듈 간의 책임 분담을 명확히 하여 코드의 유지보수성과 확장성을 높였다.

가장 먼저 실행되는 init_my_assembler() 함수는 어셈블러 동작에 필요한 기초 데이터 구조들을 초기화한다. init_inst_file()과 init_input_file()을 호출하여 각각 inst_table과 input_data를 업데이트한다.

이후 assem_pass1() 함수에서는 어셈블리 소스의 각 줄을 파싱하고, 각 명령어 혹은 지시어에 따라 LOCCTR을 계산하며, 동시에 토큰 테이블(token_table), 심볼 테이블(sym_table), 리터럴 테이블(literal_table)을 구성한다. 이 단계는 PASS2에서의 기계어 변환을 위한 정보를 선행적으로 구축하는 목적이다. label을 추가할 때 insert_symtab()을 호출하며, 중복검사를 통과한 label만 추가된다. 리터럴은 insert_literaltab()을 통해 등록되며, LTORG나 END에서 assign_littab_addresses()를 통해 실제 메모리 주소를 부여받는다.

assem_pass2()에서는 앞서 수집된 정보를 기반으로 object code를 생성한다. 각 명령어에 대한 포맷과 주소 지정 비트는 상황에 따라 분석되며, 생성된 기계

어 코드는 내부적으로 H/T/M/E/D/R 형태의 레코드로 구성되어 obj_codes[] 배열에 저장된다.

최종 출력은 make_objectcode_output() 함수를 통해 수행된다.

3.2) 모듈별 설계 내용

◆ 1) init_inst_file(char* inst_file)
- 명령어 테이블(inst_table.txt)을 읽어 각 명령어를 구조체에 저장
- 형식이 "3/4"인 경우에는 format을 3으로 저장

◆ 2) init_input_file(char* input_file)
- 어셈블리 소스 파일을 한 줄씩 읽어 input_data[]에 저장
- 최대 줄 수(MAX_LINES)를 넘지 않도록 체크

◆ 3) token_parsing(char* str)
- 각 줄을 분석하여 label, operator, operand[], comment를 추출
- 피연산자 분리 시 ','를 기준으로 최대 3개까지 처리하며, 주석 내 쉼표는 무시
- 명령어가 RSUB처럼 피연산자가 없을 경우 주석과 operand를 혼동하지 않도록 inst_table에서 피연산자 수를 참조

◆ 4) search_opcode(char* str)
- 명령어(operator)로부터 opcode를 찾아 inst_table에 존재시 인덱스 반환
- '+' 접두사가 붙은 format 4 명령어도 대응함

◆ 5) assem_pass1(void)
- 어셈블리 코드의 각 라인을 읽고, token_parsing()을 통해 label, operator, operand 정보 추출
- operator에 따라 피연산자 수를 결정하고, inst_table을 참고하여 유효성을 확인
- label이 있는 경우 insert_symtab()을 호출하여 중복 여부를 확인하고 symbol table에 등록
- 리터럴이 등장하면 insert_literaltab()을 통해 literal table에 추가하고, 주소는 -1로 초기화
- EQU 뒤에 label이 올 경우 get_symval()을 통해 값을 계산하여 심볼 주소로 저장
- LTORG / END가 등장하면 assign_littab_addresses()를 호출하여 리터럴의 실제 주소를 부여
- 다중 CSECT 구조를 지원하기 위해 섹션별 symtable, token 인덱스를 따로 관리

◆ 6) assem_pass2(void)

- START나 CSECT가 등장하면 H 레코드를 생성하여 프로그램 이름과 시작 주소를 저장
- EXTDEF, EXTREF가 등장하면 D/R 레코드를 구성하여 external 심볼 정보를 기록
- 명령어를 search_opcode()로 확인하고, set_nixbpe()를 통해 주소지정 비트 할당
- 피연산자의 주소를 구해 object code를 조립하고, text buffer에 누적하여 T 레코드를 구성
- SYM1 - SYM2 같은 표현이나 Format 4 명령어가 등장하면 M 레코드를 생성하여 수정 정보를 추가
- 리터럴은 LTORG나 END에서 처리되며, object code로 변환되어 textbuffer에 저장
- RESB, RESW가 나오면 object code를 끊고, text 레코드 강제 마무리
- END에서는 남은 text, modify, end 레코드를 모두 정리하여 하나의 완성된 obj 코드 완성

◆ 7) insert_symtab(int start_symtable, const char* label, int addr)

- label중복 검사 이후 sym_table[]에 삽입
- CSECT가 다를 시 같은 label이 허용되므로 start_symtable을 설정해 테이블 순회 위치 지정

◆ 8) insert_literaltab(const char* label)

- literal 중복 검사 이후 literal_table에 추가
- 리터럴의 주소는 아직 미정이므로 -1로 초기화
- CSECT별 리터럴 관리를 위해 literal_line_index[]로 현재 줄이 몇 번째 token인지 저장

◆ 9) get_symval(int start_symtable, const char* label)

- 심볼 주소값 검색
- CSECT별 검색을 위해 start_symtable을 설정해 검색 시작 인덱스 설정

◆ 10) set_nixbpe(token* t, int format)

- 주소비트(nixbpe) 설정
- pc-relative, base-relative 범위체크

◆ 11) get_last_allocated_addr(int idx)

- 마지막에 할당된 주소를 찾는 함수
- SECT종료 후 literal할당 등의 작업이 모두 끝난 뒤의 주소를 반환함

◆ 12) make_opcode_output(char* file_name)

- 토큰 테이블을 순회하며 각 줄 앞에 opcode를 출력 파일로 출력

◆ 13) make_symtab_output(char* file_name)

- 심볼테이블 출력 함수

◆ 14) make_literaltab_output(char* file_name)

- 리터럴 테이블출력 함수

◆ 15) make_objectcode_output(char* file_name)

- object_code[]를 순회하며 완성된 obj code를 출력하는 함수

◆ 16) assign_littab_addressed()

- LTORG를 만났을 때 locctr을 업데이트 하기 위한 함수

◆ 17) init_my_assembler(void)

- 프로그램 초기화를 위한 자료구조 생성 및 파일을 읽는 함수

4장 시스템 구현 내용 (구현 화면 포함)

4.1) 전체 시스템 구현 내용

본 어셈블러는 main() 함수에서 전체 동작을 순차적으로 호출하며, 초기화 → PASS1 → PASS2 → 출력의 흐름으로 구현되어 있다. 특히 PASS1과 PASS2는 각각의 독립된 로직을 갖고 있으며, PASS1은 테이블 구축, PASS2는 실제 object code 생성을 담당한다.

```
main() {
  init_my_assembler();
  assem_pass1();
  make_symtab_output();
  make_literaltab_output();
  assem_pass2();
  make_objectcode_output();
}
```

1. **init_my_assembler() : inst_table, input_data 초기화**
   - ├ init_inst_file() : inst_table 초기화
   - └ init_input_file() : input_data 초기화

2. **assem_pass1() : token_table생성, LOCCTR 계산, sym/lit_table 생성**
   - ├ insert_symtab() : label중복 확인 및 symbol_table 생성
   - ├ assign_littab_addresses() : LTRORG 와 END전에 호출되어 locctr 업데이트
   - ├ insert_literaltab() : literal중복 확인 및 literal_table 생성
   - ├ search_opcode() : inst_table을 확인해 입력 문자열이 기계어 코드인지 검사
   - └ token_parsing() : 소스코드 토큰단위 분석 및 token_table 생성

3. **assem_pass2() : obj 코드 생성 및 레코드 처리**
   - ├ get_symval() → 심볼 주소 반환
   - └ get_last_allocated_addr() → CSECT 종료 주소 계산

4. **make_symtab_output() : object file 출력**

4.2) 모듈별 구현 내용

◆ 1) init_inst_file()

기계어 명령어 테이블을 초기화하는 함수로, inst_table.txt 파일을 한 줄씩 읽어 각 명령어에 대한 정보를 구조체에 저장한다.

- 명령어 이름, 형식, opcode, 피연산자 수를 sscanf_s()를 사용해 파싱
- format 필드가 "3/4"일 경우, format은 3으로 저장(추후 + 접두사로 format 4 인식)
- 명령어는 inst_table[] 배열에 동적으로 할당하여 저장

inst_table.txt의 경우 "이름 형식 기계어 코드 오퍼랜드의개수 ₩n"의 형식을 가진다. 따라서 buffer에 파일을 한줄씩 읽고 아래와 같이 sscanf_s함수를 작성해 parsing한다.

```
if (sscanf_s(buffer, "%s\t%s\t%x\t%d",
    new_inst->str, (unsigned int)sizeof(new_inst->str),
    format, (unsigned int)sizeof(format),
    &op_tmp,
    &new_inst->ops) != 4) {
    printf("inst_table.txt 파일을 파싱하는데 실패했습니다.\n");
    return -1;
}
```

◆ 2) init_input_file()

어셈블리 소스 코드 파일(input.txt)을 한 줄씩 읽어 input_data[]에 저장하는 함수이다.

- 파일을 열고 fgets()로 한 줄씩 읽음
- 각 줄은 _strdup()을 통해 메모리에 복사한 후 저장
- 최대 줄 수는 MAX_LINES로 제한

```
// 파일을 라인 단위로 읽어 input_data에 저장
// MAXLINES을 넘을 경우 중단
while (fgets(buffer, sizeof(buffer), file) != NULL && line_num < MAX_LINES)
{
    char* temp = _strdup(buffer);
    if (temp == NULL) {
        printf("메모리 할당 실패 at line %d\n", line_num);
        break;
    }
    input_data[line_num++] = temp;
}
```

메모리 할당 실패 시 오류 메시지 출력, 파일 열기 실패 시 에러를 반환한다.

◆ 3) token_parsing()

어셈블리 코드 한 줄을 파싱하여 token 구조체로 분리하는 핵심 함수이다.

- '.'으로 시작할 경우 주석처리한다.

```
// 주석처리
if (str[0] == '.') {
    strcpy_s(tok->comment,sizeof(tok->comment), str);
    token_table[token_line] = tok;
    token_line++;
    return 0;
}
```

- 한 줄을 label, operator, operands, comment 4가지 요소로 분리

```
// label이 없는경우 : '\t'로 시작
if (str[0] == '\t') {
    tmp_str = tmp_str + 1;
    tmp_oper = strtok_s(tmp_str, " \t\n", &context);
}
//label이 있는 경우
else {
    tmp_label = strtok_s(tmp_str, " \t\n", &context);
    tmp_oper = strtok_s(NULL, " \t\n", &context);
}
```

입력받은 문자의 시작이 '₩t'인지 그렇지 않은지를 비교하여 label이 있는경우와 없는 경우로 나눈다.

- RSUB와 같이 operand가 없는 명령어의 경우, comment를 operand로 잘못 parsing 하는 경우가 생길 수 있다. 이를 방지하기 위해 inst_table->ops에 저장된 최대 operand수를 참고해 for문의 횟수를 제한한다.

```
if (tmp_oper != NULL) {
    tok->oper = _strdup(tmp_oper);

    // operator가 NULL이 아닌경우 최대 피연산자 개수를 가져온다
    for (int i = 0; i < inst_index; i++) {
        const char* opcheck = (tmp_oper[0] == '+') ? tmp_oper + 1 : tmp_oper;
        if (strcmp(inst_table[i]->str, opcheck) == 0) {
            num_operands = inst_table[i]->ops;
            break;
        }
    }
}
// operand 개수가 0 ex) RSUB인 경우에는 나머지 문자열을 모두 comment처리한다.
if (num_operands > 0)   tmp_operands = strtok_s(NULL, " \t\n", &context);
tmp_comment = strtok_s(NULL, "", &context); //남은 문자열을 모두 주석으로 처리
```

- 주석에 포함된 쉼표와 operand의 쉼표를 구분하기 위해 2단계 파싱을 수행
  1. tmp_opers에 쉼표(,)를 기준으로 피연산자를 나누지 않고, 피연산자가 2개 이상일 경우에도 operands 필드에 하나의 문자열로 저장한다.
  2. tmp_opers를 쉼표(,)을 기준으로 다시 parsing하여 저장한다.

```
/* operand parsing :
* operand는 ','로 구분되어 있다.
* tmp_operands로 따로 구분한 이유는 comment의 ','와 구분하기 위해서이다.
*/
if (tmp_operands != NULL) {
    char* operand_context = NULL;   // strtok_s()의 context 변수
    char* tmp_operand = strtok_s(tmp_operands, ",", &operand_context);
    int i = 0;
    while (tmp_operand != NULL && i < MAX_OPERAND) {
        tok->operand[i++] = _strdup(tmp_operand);
        tmp_operand = strtok_s(NULL, ",", &operand_context);
    }
}
```

◆ 4) search_opcode()

문자열로 주어진 operator를 inst_table[]에서 검색하여 해당 명령어의 inst_table 인덱스를 반환하는 함수이다.

- '+'로 시작하는 명령어 처리 (예: +JSUB → JSUB)

```
//4형식 처리 : +로 시작
if (str[0] == '+')
{
    str++;  // +를 제거


}
```

- inst_table에서 문자열 비교를 통해 opcode 검색하고, 일치하는 opcode를 찾으면 해당 opcode값을, 그렇지 못한 경우 -1을 반환한다.

```
for (int i = 0; i < inst_index; i++)
{
    if (strcmp(inst_table[i]->str, str) == 0)
    {
        return i;
    }
}
return -1;  // 찾지 못한 경우
```

◆ 5) assem_pass1(void)

PASS1에서는 라인 단위로 소스를 파싱한 뒤, LOCCTR를 계산하고 심볼 테이블과 리터럴 테이블을 생성한다.

- assem_pass1() 호출 시 초기화 변수 :

1. csect_start_token_line[] : CSECT별 관리를 위해 csect_start_token_line[]배열에 각 섹터별 시작 토큰 인덱스를 저장한다. assem_pass1()이 처음 호출된 경우 0번째 토큰이 섹터의 시작이므로 0을 저장한다.

2. start_encounter : START가 2회 이상 등장할 경우 에러 처리를 하기위한 flag

3. start_symtab : 섹터별 symbol_table을 따로 분리하여 서칭하기 위해 저장하는 검색 시작 index

```
csect_start_token_line[0] = 0;  // CSECT 시작 토큰 라인 저장
bool start_encounter = false;   // START 여부
int start_symtab = 0;   // sym table 서칭 시작 인덱스
```

- Comment 처리

'.'으로 시작할 시 comment 처리

```
//comment line
if (t->oper == NULL && t->comment[0] == '.') {
    // 주석 라인
    continue;
}
```

-START 처리

1. start_encounter를 참고하여 START가 2회이상 나왔을 시 에러처리

```
//START
if (strcmp(t->oper, "START") == 0) {
    // START가 2회 이상 나오면 에러
    if (start_encounter) {
        printf("START 명령어가 중복되었습니다.\n");
        return -1;
    }
```

2. START 1000과 같이 시작주소를 입력해 주었을 때 locctr에 해당 주소 입력
   만약 아무런 값이 없을 경우 자동으로 0 할당

```
    // 프로그램 시작 주소를 설정
    if (t->operand[0] != NULL) {
        locctr = (int)strtol(t->operand[0], NULL, 16);
    }
    else {
        locctr = 0; // 시작주소가 없으면 locctr를 0으로 초기화
    }
    start_addr = locctr;    // 프로그램 시작 주소 저장
    start_encounter = true; // START가 나왔음을 표시
```

3. sym_tab에 START 주소 저장

```
    // START label저장
    if (t->label != NULL) {
        if (insert_symtab(start_symtab, t->label, locctr) < 0) return -1;
    }
    t->addr = locctr;   // locctr를 addr에 저장
    continue;   // START는 locctr를 증가시키지 않음
}
```

- CSECT 처리

1.CSECT를 만날시 새로운 섹터의 symtab에서의 위치를 csect_start_symbol_num[]에 저장해 차후 symtab 서칭 시작위치를 할당한다.

2. csect_start_token_line[]에는 새로운 섹터의 시작 토큰 위치를 저장한다.

3. symtab에 공백 및 NULL값을 넣어 섹터를 구분한다.

```
// CSECT
if (t->oper && strcmp(t->oper, "CSECT") == 0) {
    assign_littab_addresses();
    csect_start_symbol_num[csect_num++] = label_num; // CSECT 시작 심볼 번호 저장
    csect_start_token_line[csect_num] = i; // CSECT 시작 토큰 라인 저장
    start_symtab = label_num;   // 다음 label부터 시작
    insert_symtab(start_symtab, "", NULL); // 빈 label 섹터 구분용
    locctr = 0;
    if (t->label) insert_symtab(start_symtab, t->label, locctr); // CSECT 종료 위치 저장
    t->addr = locctr;   // locctr를 addr에 저장
    continue;   // LOCCTR 업데이트 없음
}
```

- label 처리

label을 만날 시 symtable에 locctr과 함께 저장한다.

```
// label 저장
if (t->label != NULL && strcmp(t->oper, "EQU") != 0 ) {
    if (insert_symtab(start_symtab, t->label, locctr) < 0) return -1;
}
```

-EQU 처리

케이스별로 나누어 value에 값을 할당하고 symtab에 value와 함께 저장한다.

1. EQU * 일 경우 현재 주소 저장

```
// 피연산자가 '*'인 경우
if (strcmp(t->operand[0], "*") == 0) {
    value = locctr;
}
```

2. EQU (숫자)인 경우

```
// 피연산자가 숫자인 경우
else if (isdigit(t->operand[0][0])) {
    value = atoi(t->operand[0]);
}
```

3. 피연산자에 연산이 들어간 경우

get_symval로 각각의 주소값을 가져와 계산한 값을 value에 저장한다.

만약 피연산자의 값이 테이블에 존재하지 않은 경우 0을 저장한다.

```
// '-'연산 처리
else if(strchr(t->operand[0], '-') != NULL) {
    char operand_copy[100];
    strcpy_s(operand_copy, sizeof(operand_copy), t->operand[0]);

    char* left = strtok(operand_copy, "-");
    char* right = strtok(NULL, "-");

    int left_val = get_symval(start_symtab , left);
    int right_val = get_symval(start_symtab ,right);

    value = (left_val == -1 || right_val == -1) ? 0 : left_val - right_val;
}
```

4. 심볼값이 주어진 경우

해당 심볼값을 get_symval()로 찾아 저장한다. 만약 찾지 못한 경우 0을 저장한다.

```
// 피연산자가 심볼인 경우
else {
    int symvalue = get_symval(start_symtab, t->operand[0]);
    value = (symvalue == -1) ? 0 : symvalue;    //심볼을 찾지 못한경우 0으로 채움
}
```

5. value에 저장한 값을 테이블에 업데이트

```
// 심볼 테이블에 저장
if (t->label != NULL) {
    if (insert_symtab(start_symtab, t->label, value) < 0) return -1;
}
t->addr = locctr;   // locctr를 addr에 저장
continue; // LOCCTR 업데이트 없음
```

-LTORG

pass1에서는 LTORG를 만나면 locctr업데이트 실시

```
//LTORG
if (strcmp(t->oper, "LTORG") == 0) {
    assign_littab_addresses();
    continue;
}
```

- Literal처리

'='를 만나면 리터럴 테이블에 리터럴 추가

```
// 리터럴 처리
if (t->operand[0] != NULL && t->operand[0][0] == '=') {
    // 리터럴 테이블에 추가
    insert_literaltab(t->operand[0]);
}
```

- 지시어 처리

1. Byte는 C일 때 문자의 개수, X일 때 1바이트의 크기를 할당한다.

```
// 지시어 처리
//BYTE
if (strcmp(t->oper, "BYTE") == 0) {
    int added_byte = 0;
    if (t->operand[0][0] == 'C') {
        added_byte = strlen(t->operand[0]) - 3; // C'abc' -> 3
    }
    else if (t->operand[0][0] == 'X') {
        added_byte = (strlen(t->operand[0]) - 3) / 2; // X'12' -> 2
    }
    else {
        return -1; // 잘못된 BYTE 형식
    }
    locctr += added_byte;
}
```

2. WORD처리

word는 무조건 3바이트공간 할당

```
//WORD
else if (strcmp(t->oper, "WORD") == 0) {
    locctr += 3;
}
```

3. RESERVED 처리

RESB는 할당한 1B * 할당한 공간 수, RESW는 3B * (할당한 공간 수) 크기의 메모리 할당

```
//RESB
else if (strcmp(t->oper, "RESB") == 0) {
    if (t->operand[0] != NULL) {
        locctr += atoi(t->operand[0]);
    }
}
//RESW
else if (strcmp(t->oper, "RESW") == 0) {
    if (t->operand[0] != NULL) {
        locctr += 3 * atoi(t->operand[0]);
    }
}
```

-명령어 처리

3형식의 경우 3B 4형식의 경우 4B의 locctr 증가

```
else{
// opcode 검색
    int idx = search_opcode(t->oper);
    if (idx >= 0) {
        if (t->oper[0] == '+') {
            // 4형식인 경우
            locctr += inst_table[idx]->format + 1;
        }
        else {
            locctr += inst_table[idx]->format;
        }
    }
}
```

-마지막 토큰 호출

마지막 토큰 END 만난 뒤 리터럴 처리 및 0 리턴

```
// locctr 업데이트
assign_littab_addresses();
return 0;
```

◆ 6) assem_pass2(void)

어셈블러의 두 번째 단계(PASS2)를 수행하는 핵심 함수로, PASS1에서 생성된 token_table[], sym_table[], literal_table[]을 바탕으로 SIC/XE 포맷에 맞는 object code를 생성하고 이를 obj_codes[] 배열에 저장한다.

- 함수 호출시 초기화되는 변수들:

```
csect_num = 0;
int start_symtab = 0;       // sym table 서칭 시작 인덱스
bool mainroutine = false;   // 메인루틴 여부
int last_head_idx = -1;     // 마지막 헤더 레코드 인덱스
int text_length = 0;        // 텍스트 레코드 길이
int text_start = -1;        // 텍스트 레코드 시작 주소
char text_buffer[70] = "";  // 텍스트 레코드 버퍼
```

- START 지시어 → H레코드 생성:

프로그램 시작을 나타내는 START를 만나면 Header 레코드를 생성한다.
프로그램 이름과 시작 주소를 기록하고, 나중에 길이(length)는
get_last_allocated_addr()로 보완한다.

```
// START 처리
if (t->oper != NULL && strcmp(t->oper, "START") == 0) {
    last_head_idx = obj_count;  // 헤더 레코드 인덱스 저장
    mainroutine = true;

    // start 심볼은 H로 시작
    oc.record_type = 'H';
    oc.locctr = t->addr;    // locctr 저장

    // object_code를 공백으로 초기화하고, 프로그램 이름 복사
    memset(oc.object_code, ' ', 6);  // 6칸 전부 공백
    if (t->label != NULL) {
        size_t len = strlen(t->label);
        if (len > 6) len = 6;
        memcpy(oc.object_code, t->label, len);  // 앞에서부터 복사
    }
    oc.object_code[6] = '\0';
    obj_codes[obj_count++] = oc;    // 객체 코드 저장
    continue;
}
```

- EXTDEF → D 레코드: EXTDEF는 외부에서 참조할수 있도록 정의한 심볼

```
// EXTDEF 처리 → D 레코드
if (strcmp(t->oper, "EXTDEF") == 0) {
    for (int j = 0; j < MAX_OPERAND && t->operand[j]; j++) {
        strcpy_s(extdef_list[extdef_count++], MAX_LINE_LENGTH, t->operand[j]);
    }
    // EXTDEF가 끝나면 D 레코드 생성
    if (extdef_count > 0) {
        object_code drec = { 'D', -1, -1, "" };
        for (int k = 0; k < extdef_count; k++) {
            int addr = get_symval(start_symtab, extdef_list[k]);
            char name[7], entry[20];
            snprintf(name, sizeof(name), "%-6s", extdef_list[k]);       // 이름: 6자 공백 패딩
            snprintf(entry, sizeof(entry), "%s%06X", name, addr);       // 전체: 이름+주소
            strcat_s(drec.object_code, sizeof(drec.object_code), entry);
        }
        obj_codes[obj_count++] = drec;
    }

    continue;
}
```

- EXTREF → R 레코드 : EXTREF는 외부에서 가져올 심볼 목록을 나타냄

```
// EXTREF 처리 → R 레코드
if (strcmp(t->oper, "EXTREF") == 0) {
    for (int j = 0; j < MAX_OPERAND && t->operand[j]; j++) {
        strcpy_s(extref_list[extref_count++], MAX_LINE_LENGTH, t->operand[j]);
    }
    // R 레코드 생성
    if (extref_count > 0) {
        object_code rrec = { 'R', -1, -1, "" };
        for (int k = 0; k < extref_count; k++) {
            char buffer[7];
            snprintf(buffer, sizeof(buffer), "%-6s", extref_list[k]);   // 이름: 6자 공백 패딩
            strcat_s(rrec.object_code, sizeof(rrec.object_code), buffer);
        }
        obj_codes[obj_count++] = rrec;
    }
    continue;
}
```

- CSECT 처리 → 섹터 마무리 & 다음레코드 시작 :

CSECT를 만나면 현재의 obj코드를 마무리 해야함

1.이전 T레코드 종료

```
// 이전 T 레코드 종료
if (text_length > 0) {
    object_code rec = { 'T', text_start, text_length, "" };
    strcpy_s(rec.object_code, sizeof(rec.object_code), text_buffer);
    obj_codes[obj_count++] = rec;   // 객체 코드 저장


}

text_length = 0;    // 텍스트 레코드 길이 초기화
text_start = -1;    // 텍스트 레코드 시작 주소 초기화
memset(text_buffer, 0, sizeof(text_buffer));    // 버퍼 초기화
// 이전 헤더 레코드 길이 업데이트
if (last_head_idx != -1 ) {
    int end_addr = get_last_allocated_addr(i);
    if (end_addr >= 0) {
        obj_codes[last_head_idx].length = end_addr - obj_codes[last_head_idx].locctr;
    }
}
```

2. 이전 obj 코드에 M레코드 추가

2-1) 4형식의 경우 해당 주소+1위치의 하위 5개 obj 코드 수정

```
// 4형식인 경우
if (tk->oper != NULL && tk->oper[0] == '+') {
    // EXTREF 안에 있는 operand만 수정 대상
    for (int r = 0; r < extref_count; r++) {
        if (tk->operand[0] && strcmp(tk->operand[0], extref_list[r]) == 0) {
            object_code mrec = { 'M', tk->addr + 1, 5, "" }; // 주소 +1, 길이 5
            snprintf(mrec.object_code, sizeof(mrec.object_code), "05+%s", extref_list[r]);
            obj_codes[obj_count++] = mrec;
        }
    }
}
```

2-2) BUFEND - BUFFER 등의 연산 포함 : 각 operand를 파싱해 M레코드 작성

```
// WORD 명령어에서 SYM1 - SYM2 형식 확인
else if (tk->oper != NULL && strcmp(tk->oper, "WORD") == 0) {
    //-여부 확인
    char* minus = strchr(tk->operand[0], '-');
    if (minus) {
        char left[MAX_LINE_LENGTH], right[MAX_LINE_LENGTH];
        strncpy_s(left, sizeof(left), tk->operand[0], minus - tk->operand[0]);
        strcpy_s(right, sizeof(right), minus + 1);

        for (int r = 0; r < extref_count; r++) {
            if (strcmp(left, extref_list[r]) == 0) {
                object_code mrec = { 'M', tk->addr, 6, "" };
                sprintf_s(mrec.object_code, sizeof(mrec.object_code), "06+%s", extref_list[r]);
                obj_codes[obj_count++] = mrec;
            }
        }
        for (int r = 0; r < extref_count; r++) {
            if (strcmp(right, extref_list[r]) == 0) {
                object_code mrec = { 'M', tk->addr, 6, "" };
                sprintf_s(mrec.object_code, sizeof(mrec.object_code), "06-%s", extref_list[r]);
                obj_codes[obj_count++] = mrec;
            }
        }
    }
}
```

3. END 레코드 추가 및 새로운 T레코드 시작

```
// END 레코드 추가
if (mainroutine) {
    object_code end_rec = { 'E', -1, -1, "" };
    end_rec.locctr = start_addr;    // 시작 주소
    obj_codes[obj_count++] = end_rec;   // 객체 코드 저장
}
else {
    object_code end_rec = { 'E', -1, -1, "" };
    obj_codes[obj_count++] = end_rec;   // 객체 코드 저장
}
mainroutine = false;    // 메인루틴 종료

// 새로운 H 레코드 추가
object_code new_rec = { 'H', t->addr, 0, "" };
memset(new_rec.object_code, ' ', 6);  // 6칸 전부 공백
if (t->label != NULL) {
    size_t len = strlen(t->label);
    if (len > 6) len = 6;
    memcpy(new_rec.object_code, t->label, len);  // 앞에서부터 복사
}
```

- END 처리 :

남은 리터럴 데이터를 모두 텍스트 레코드로 저장하고, 해당 CSECT의 마지막 주소로 H레코드 길이를 계산. E레코드를 통해 프로그램 종료 표시

- 리터럴 정보 표시

```
// =C'...' 처리
if (literal_table[j].symbol[1] == 'C') {
    char* lit = literal_table[j].symbol + 3;  // =C'EOF' → 'EOF'
    for (int k = 0; lit[k] != '\'' && lit[k] != '\0'; k++) {
        char hex[3];
        sprintf_s(hex, sizeof(hex), "%02X", lit[k]);
        strcat_s(obj, sizeof(obj), hex);
    }
    len = strlen(obj) / 2;
}
// =X'...' 처리
else if (literal_table[j].symbol[1] == 'X') {
    strncpy_s(obj, sizeof(obj), literal_table[j].symbol + 3, strlen(literal_table[j].symbol) - 4);
    obj[strlen(literal_table[j].symbol) - 4] = '\0';
    len = strlen(obj) / 2;
}
```

-헤더 레코드 길이 설정

```
//헤더 레코드 길이 설정
if (last_head_idx != -1) {
    int end_addr = get_last_allocated_addr(i);
    if (end_addr >= 0) {
        obj_codes[last_head_idx].length = end_addr - obj_codes[last_head_idx].locctr;
    }
}
```

-남은 레코드 처리

```
//남은 text_buffer 한 번에 저장
if (text_length > 0) {
    object_code rec = { 'T', text_start, text_length, "" };
    strcpy_s(rec.object_code, sizeof(rec.object_code), text_buffer);
    obj_codes[obj_count++] = rec;

    text_length = 0;
    text_start = -1;
    memset(text_buffer, 0, sizeof(text_buffer));
}
```

- CSECT를 만났을때와 마찬가지로 M레코드 처리

- E레코드 추가

```
//E 레코드 추가
object_code end_rec = { 'E', -1, -1, "" };
end_rec.locctr = mainroutine ? start_addr : -1;
obj_codes[obj_count++] = end_rec;

mainroutine = false;
continue;
```

- RESW, RESB 처리 : 해당 명령어를 만날 시 obj코드 개행

```
// RESB, RESW 처리 :얘들을 만나면 obj코드 개행을 해야함
else if (strcmp(t->oper, "RESB") == 0 || strcmp(t->oper, "RESW") == 0) {
    if (text_length > 0) {
        object_code rec = { 'T', text_start, text_length, "" };
        strcpy_s(rec.object_code, sizeof(rec.object_code), text_buffer);
        obj_codes[obj_count++] = rec;   // 객체 코드 저장

        text_length = 0;    // 텍스트 레코드 길이 초기화
        text_start = -1;    // 텍스트 레코드 시작 주소 초기화
        memset(text_buffer, 0, sizeof(text_buffer));    // 버퍼 초기화
    }
    continue;
}
```

- LTORG 처리 :

리터럴 테이블 순회하며 아직 처리되지 않은 리터럴을 obj code로 변환하여 text_buffer에 추가

```
// LTORG 이전에 등장한 리터럴 중 아직 처리 안 한 것만
if (literal_line_index[j] <= i && literal_table[j].addr != -1) {
    int len = 0;
    char obj[70] = "";

    if (literal_table[j].symbol[1] == 'C') {
        char* lit = literal_table[j].symbol + 3;  // =C'EOF' → 'EOF'
        for (int k = 0; lit[k] != '\'' && lit[k] != '\0'; k++) {
            char hex[3];
            sprintf_s(hex, sizeof(hex), "%02X", lit[k]);
            strcat_s(obj, sizeof(obj), hex);
        }
        len = strlen(obj) / 2;
    }
    else if (literal_table[j].symbol[1] == 'X') {
        strncpy_s(obj, sizeof(obj), literal_table[j].symbol + 3, strlen(literal_table[j].symbol) - 4);
        obj[strlen(literal_table[j].symbol) - 4] = '\0';
        len = strlen(obj) / 2;
    }
```

- **일반 명령어 처리:**

  형식 분석(2,3,4), 심볼주소 참조 및 최종적으로 기계어 코드를 생성하고 text_buffer에누적 저장한다.

  1. immediate

```
// immediate
if (t->operand[0] != NULL && t->operand[0][0] == '#') {
    target = (int)strtol(t->operand[0] + 1, NULL, 16);  // immediate 값 가져오기
    unsigned int tmp = (opcode << 16);
    tmp |= (t->nixbpe) << 12;
    tmp |= (target & 0x0FFF);   // 12비트 상대 주소
    sprintf_s(obj, sizeof(obj), "%06X", tmp);
}
```

  2. indirect

```
// immediate
if (t->operand[0] != NULL && t->operand[0][0] == '#') {
    target = (int)strtol(t->operand[0] + 1, NULL, 16);  // immediate 값 가져오기
    unsigned int tmp = (opcode << 16);
    tmp |= (t->nixbpe) << 12;
    tmp |= (target & 0x0FFF);   // 12비트 상대 주소
    sprintf_s(obj, sizeof(obj), "%06X", tmp);
}
```

  3. 일반 명령어

```
// 일반 명령어
else if (t->operand[0] != NULL) {
    target = get_symval(start_symtab, t->operand[0]);                    // 심볼 주소 가져오기
}
```

  위의 과정을 통해 target에 TA를 저장하고 아래의 계산에서 pc-relative를 계산한다.

  3/4형식:

```
// format 3
else if (format == 3) {
    disp = target - (t->addr + 3);  // 상대 주소 계산
    unsigned int tmp = (opcode << 16);
    tmp |= (t->nixbpe) << 12;
    tmp |= (disp & 0x0FFF); // 12비트 상대 주소
    sprintf_s(obj, sizeof(obj), "%06X", tmp);
}
// format 4
else if (format == 4) {
    disp = target - (t->addr + 4);  // 상대 주소 계산
    unsigned int tmp = (opcode << 24);
    tmp |= (t->nixbpe) << 20;
    tmp |= 0x00000000;
    sprintf_s(obj, sizeof(obj), "%08X", tmp);
}
```

2형식:

```
else if (format == 2) {
    unsigned int tmp = opcode << 8;
    if (t->operand[0] != NULL) {
        if (strcmp(t->operand[0], "A") == 0) {
            tmp |= 0x00;
        }
        else if (strcmp(t->operand[0], "X") == 0) {
            tmp |= 0x10;
        }
        else if (strcmp(t->operand[0], "L") == 0) {
            tmp |= 0x20;
        }
        else if (strcmp(t->operand[0], "B") == 0) {
            tmp |= 0x30;
        }
        else if (strcmp(t->operand[0], "S") == 0) {
            tmp |= 0x40;
        }
        else if (strcmp(t->operand[0], "T") == 0) {
            tmp |= 0x50;
        }
    }
    if (t->operand[1] != NULL) {
        if (strcmp(t->operand[1], "A") == 0) {
            tmp |= 0x00;
        }
        else if (strcmp(t->operand[1], "X") == 0) {
            tmp |= 0x01;
        }
        else if (strcmp(t->operand[1], "L") == 0) {
            tmp |= 0x02;
        }
        else if (strcmp(t->operand[1], "B") == 0) {
            tmp |= 0x03;
        }
        else if (strcmp(t->operand[1], "S") == 0) {
            tmp |= 0x04;
        }
        else if (strcmp(t->operand[1], "T") == 0) {
            tmp |= 0x05;
        }
    }
```

마지막으로 text_buffer에 복사하여 누적한다.

```
// 텍스트 레코드에 추가
strcat_s(text_buffer, sizeof(text_buffer), obj);
text_length += strlen(obj) / 2; // 텍스트 레코드 길이 업데이트
```

◆ 7) insert_symtab(int start_symtable, const char* label, int addr)

- 중복 체크 후 sym_table에 label과 addr 정보 저장

```
int insert_symtab(int start_symtable,  const char* label, int addr) {
    for (int i = start_symtable; i < label_num ; i++) {
        if (strcmp(sym_table[i].symbol, label) == 0) {
            printf("중복된 label이 존재합니다.\n");
            return -1;
        }
    }
    strcpy_s(sym_table[label_num].symbol, sizeof(sym_table[label_num].symbol), label);
    sym_table[label_num].addr = addr;
    label_num++;
    return 0;
}
```

◆ 8) insert_literaltab(const char* label)

- 중복 체크 후 literal_table에 literal과 addr 저장. 이때 리터럴은 LTORG/END를 만나기 전까지 주소를 모르기에 우선 -1을 저장해 둔다.

```
int insert_literaltab(const char* label) {
    for (int i = 0; i < literal_num; i++) {
        if (strcmp(literal_table[i].symbol, label) == 0) {
            return -1;
        }
    }
    strcpy_s(literal_table[literal_num].symbol,sizeof(literal_table[literal_num].symbol), label);
    literal_table[literal_num].addr = -1;   // 주소는 아직 미정(-1)

    // literal 등장 시점 저장
    literal_line_index[literal_num] = token_line - 1;  // 현재 줄 인덱스 (token_table 기준)

    literal_num++;
    return 0;
}
```

◆ 9) get_symval(int start_symtable, const char* label)

- sym_table을 순회하며 파라미터로 들어온 label이 존재하는지 확인하는 함수.
- label이 존재할 시 해당 label의 addr를 반환, 그렇지 않으면 -1을 반환한다.

```
int get_symval(int start_symtable, const char* label) {
    if (start_symtable > label_num) {
        return -1;
    }

    for (int i = start_symtable; i < label_num; i++) {
        if (strcmp(sym_table[i].symbol, label) == 0) {
            return sym_table[i].addr;
        }
    }
    printf("%s symbol이 존재하지 않습니다.\n", label);
    return -1;
}
```

◆ 10) set_nixbpe(token* t, int format)

- 2형식은 nixbpe가 존재하지 않음

```
if (format == 2) return;
```

- 4형식은 e bit = 1

```
if (format == 4) {
    t->nixbpe |= 0x01; // e = 1
}
```

- '#'로 시작하는 operand는 immediate
- '@'로 시작하는 operand는 indirect
- 그렇지 않은 경우 i = n = 1 (SIC/XE머신)

```
// operand[0] 기준
if (t->operand[0] != NULL) {
    if (t->operand[0][0] == '#') {
        t->nixbpe |= (1 << 4);  // i = 1
    }
    else if (t->operand[0][0] == '@') {
        t->nixbpe |= (1 << 5);  // n = 1
    }
    else {
        t->nixbpe |= (1 << 5);  // n = 1
        t->nixbpe |= (1 << 4);  // i = 1
    }
}
else {
    // operand 없으면 기본으로 ni = 11
    t->nixbpe |= (1 << 5) | (1 << 4);
}
```

◆ 11) get_last_allocated_addr(int idx)

각 CSECT의 끝나는 주소를 계산하기 위한 함수

- CSECT는 token_table을 기준으로 시작 인덱스를 저장중이다.
- 현재 주어진 idx가 속한 섹터의 시작 token index를 찾아 sect_start에 저장.

```
// idx가 몇번째 sect에 있는지 확인
for (int i = 1; i < csect_num; i++) {
    if (csect_start_token_line[i] >= idx) break;
    sect_start = csect_start_token_line[i];
}
```

- 뒤에서부터 훑으며 마지막으로 메모리를 할당하는 명령어 찾음

- EQU는 주소할당이 없으므로 제외

```
// 일반 명령어 기준 가장 마지막 주소
for (int j = idx - 1; j >= sect_start; j--) {
    token* t = token_table[j];
    if (!t || t->addr < 0 || !t->oper) continue;

    if (strcmp(t->oper, "EQU") == 0) continue;
```

- RESB, RESW는 해당 크기만큼 addr 증가

- WORD, BYTE는 실제 데이터 크기에 따라 주소 계산

```
    if (strcmp(t->oper, "RESB") == 0 && t->operand[0])
        max_addr = t->addr + atoi(t->operand[0]);
    else if (strcmp(t->oper, "RESW") == 0 && t->operand[0])
        max_addr = t->addr + 3 * atoi(t->operand[0]);
    else if (strcmp(t->oper, "WORD") == 0)
        max_addr = t->addr + 3;
    else if (strcmp(t->oper, "BYTE") == 0) {
        if (t->operand[0][0] == 'C')
            max_addr = t->addr + strlen(t->operand[0]) - 3;
        else if (t->operand[0][0] == 'X')
            max_addr = t->addr + (strlen(t->operand[0]) - 3) / 2;
    }
```

- 일반 명령어 처리 4형식일 경우 format++을 해주어야함

```
    else {
        int inst_idx = search_opcode(t->oper);
        if (inst_idx >= 0) {
            int format = inst_table[inst_idx]->format;
            if (t->oper[0] == '+') format++;
            max_addr = t->addr + format;
        }
    }
```

- 리터럴 처리

```
// 리터럴 처리 (해당 섹터 안에 있는 것만)
for (int i = 0; i < literal_num; i++) {
    if (literal_line_index[i] < sect_start || literal_line_index[i] >= idx) continue;
    if (literal_table[i].addr == -1) continue;

    int len = 0;
    if (literal_table[i].symbol[1] == 'C')
        len = strlen(literal_table[i].symbol) - 4;
    else if (literal_table[i].symbol[1] == 'X')
        len = (strlen(literal_table[i].symbol) - 4) / 2;

    int lit_end = literal_table[i].addr + len;
    if (lit_end > max_addr) max_addr = lit_end;
}

return max_addr;
```

◆ 12)make_opcode_output(char* file_name)

- 토큰 테이블을 순회하며 각 줄 앞에 opcode를 출력 파일로 출력

◆ 13) make_symtab_output(char* file_name)

- 심볼테이블 출력 함수
- addr를 16진수로 출력

```
for (int i = 0; i < label_num; i++) {
    if(strcmp(sym_table[i].symbol, "") == 0){
        fprintf(file, "\n");
    }
    else if (sym_table[i].symbol != NULL) {
        fprintf(file, "%s\t\t%X\n", sym_table[i].symbol, sym_table[i].addr);
    }
}
```

◆ 14) make_literaltab_output(char* file_name)

- 리터럴 테이블출력 함수
- =‘X / =C‘를 없애기 위해 literal_table[i].symbol +3부터 복사를 시작

```
for (int i = 0; i < literal_num ; i++) {
    if (strcmp(literal_table[i].symbol, "") == 0) {
        fprintf(file, "\n");
    }
    else if (literal_table[i].symbol != NULL) {
        char tmp[100];
        strncpy_s(tmp, sizeof(tmp),literal_table[i].symbol+3, strlen(literal_table[i].symbol) - 4);
        fprintf(file, "%s\t\t%X\n", tmp, literal_table[i].addr);
    }
}
```

◆ 15) make_objectcode_output(char* file_name)

object_code[]를 순회하며 완성된 obj code를 출력하는 함수

- H 레코드 : H[프로그램이름][시작위치][길이]

```
if (rec->record_type == 'H') {
    fprintf(file, "\nH%06s%06X%06X\n",rec->object_code, rec->locctr, rec->length);
}
```

- T 레코드 : T[시작주소][길이][코드내용]

```
else if (rec->record_type == 'T') {
    fprintf(file, "T%06X%02X%s\n", rec->locctr, rec->length, rec->object_code);
}
```

- M 레코드 : M[주소][길이][+/-심볼명]

```
else if (rec->record_type == 'M') {
    fprintf(file, "M%06X%s\n", rec->locctr, rec->object_code);
}
```

- D 레코드 : D[이름1][주소1]...[이름n][주소n]

```
else if (rec->record_type == 'D') {
    fprintf(file, "D%s\n", rec->object_code);
}
```

- R 레코드 : R[이름1]...[이름n]

```
else if (rec->record_type == 'R') {
    fprintf(file, "R%s\n", rec->object_code);
}
```

- E 레코드 : E[시작주소] or E

```
else if (rec->record_type == 'E') {
    if (rec->locctr == -1) {
        fprintf(file, "E\n");
    }
    else {
        fprintf(file, "E%06X\n", rec->locctr);
    }
}
```

◆ 16) assign_littab_addressed()

LTORG를 만났을 때 locctr을 업데이트 하기 위한 함수

이 함수를 통해 리터럴이 실제 메모리에서 차지할 공간을 LOCCTR 기준으로 지정할 수 있으며, 이후 PASS2에서 object code 생성 시 정확한 주소값으로 참조된다.

```
void assign_littab_addresses() {
    for (int i = 0; i < literal_num; i++) {
        if (literal_table[i].addr == -1) {
            literal_table[i].addr = locctr;
            if (literal_table[i].symbol[1] == 'C') {
                locctr += strlen(literal_table[i].symbol) - 4;
            }
            else if (literal_table[i].symbol[1] == 'X') {
                locctr += (strlen(literal_table[i].symbol) - 4) / 2;
            }
        }
    }
}
```

◆ 17) init_my_assembler(void)
이 함수는 어셈블러의 전체 실행을 시작하기 위한 초기화 루틴으로,
프로그램 실행 전 필요한 모든 자료구조를 준비하는 역할을 한다.

5장 기대효과 및 결론

본 프로젝트에서는 SIC/XE 머신을 기반으로 한 어셈블러를 설계하고 구현함으로써,
저수준 시스템 프로그래밍의 핵심 개념들을 직접 체험하고, 기계어 코드가 생성되는 전체 흐름을 이해하는 데 목적을 두었다.

2단계 어셈블리 구조(PASS1과 PASS2)를 명확히 구분하여 설계함으로써, 어셈블리 과정의 각 단계를 단계적으로 디버깅하고 분석할 수 있었다.
특히, PASS1에서는 토큰 파싱, LOCCTR 계산, 심볼 테이블 및 리터럴 테이블 구축 등 주소 기반 처리의 기초 로직을 직접 설계하였으며, PASS2에서는 이 구조적 정보를 기반으로 실제 object code를 생성하고, SIC/XE의 포맷 규칙에 맞춰 Header, Text, Modify, End 레코드를 구성하는 과정을 구현하였다.

이를 통해 어셈블리 언어의 구조와 명령어 포맷, 다양한 주소계산 방식에 대해 심화적으로 학습할 수 있었으며, 단순히 이론적으로 이해하는 과정 이외에도 로직을 직접 구현하며 마주치는 오류와 예외상황을 해결하고 코드를 구조화 하는 과정에서 실질적인 소프트웨어의 능력을 증진할 수 있었다.

또한, 본 프로젝트는 단순히 번역하는 것이 아닌 구조분석 및 메모리 처리 로직이 결합되었다는 점에서 큰 의미를 가지며, 차후 컴파일러, 링커, 로더 등의 고급 시스템에 대한 공부를 하기 전 기초 역량을 키울 수 있는 중요한 기반이 되었으리라 생각한다.

한편, 구현과정에서 아쉬움이 남는 부분은 다음과 같다.
CSECT를 구분하고 처리하는 로직은 현재 csect_start_token_line[],
csect_start_symbol_num[] 배열에 의존한 하드코딩에 가까운 방식으로 구성되어 있어, 가독성과 유지보수 측면에서 아쉬움이 크다.
섹션별 경계 구분, 헤더 레코드 길이 계산, 리터럴 처리 시점 등을 배열 인덱스로 추적 하다 보니, 전체 로직이 복잡해지고 구조적으로도 분리되지 않아 로직 흐름이 지저분하게 꼬일 여지가 많았다.
향후에는 이를 구조체 기반으로 재설계하거나, 섹션 단위의 처리 블록을 별도의 함수로 모듈화함으로써 보다 명확하고 일관된 설계 구조를 구축하는 것이 필요하다고 판단된다.

첨부) 프로그램 소스파일

<my_assembler_20211397.c>

```
/*
 * 파일명 : my_assembler.c
 * 설  명 : 이 프로그램은 SIC/XE 머신을 위한 간단한 Assembler 프로그램의 메인루틴으로,
 * 입력된 파일의 코드 중, 명령어에 해당하는 OPCODE를 찾아 출력한다.
 *
 */
/*
  *
  * 프로그램의 헤더를 정의한다.
  *
  */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <stdbool.h>
  // 파일명의 "00000000"은 자신의 학번으로 변경할 것.
#include "my_assembler_20211397.h"
/*
------------------------------------------------------------------------------
------
 * 설명 : 사용자로 부터 어셈블리 파일을 받아서 명령어의 OPCODE를 찾아 출력한다.
 * 매계 : 실행 파일, 어셈블리 파일
 * 반환 : 성공 = 0, 실패 = < 0
 * 주의 : 현재 어셈블리 프로그램의 리스트 파일을 생성하는 루틴은 만들지 않았다.
 *                또한 중간파일을 생성하지 않는다.
 *
------------------------------------------------------------------------------
------
 */
int main(int args, char* arg[])
{
        if (init_my_assembler() <0)
        {
                printf("init_my_assembler: 프로그램 초기화에 실패 했습니다.₩n");
                return -1;
        }
        if (assem_pass1() <0)
        {
                printf("assem_pass1: 패스1 과정에서 실패하였습니다.  ₩n");
                return -1;
        }
        //make_opcode_output(NULL);    // 기계어 코드 출력
        make_symtab_output("output_symtab.txt");          //"output_symtab.txt"
        make_literaltab_output("output_littab.txt");      //  "output_littab.txt"
        if (assem_pass2() <0)
        {
                printf(" assem_pass2: 패스2 과정에서 실패하였습니다.  ₩n");
                return -1;
        }
        make_objectcode_output(NULL);
        return 0;
}
/*
```

```
------------------------------------------------------------------------------
------
 * 설명 : 프로그램 초기화를 위한 자료구조 생성 및 파일을 읽는 함수이다.
 * 매계 : 없음
 * 반환 : 정상종료 = 0 , 에러 발생 = -1
 * 주의 : 각각의 명령어 테이블을 내부에 선언하지 않고 관리를 용이하게 하기
 *                위해서 파일 단위로 관리하여 프로그램 초기화를 통해 정보를 읽어 올 수 있도록
 *                구현하였다.
 *
------------------------------------------------------------------------------
------
 */
int init_my_assembler(void)
{
        int result;
        if ((result = init_inst_file("inst_table.txt")) <0)
                return -1;
        if ((result = init_input_file("input.txt")) <0)
                return -1;
        return result;
}
/                                                                             *
------------------------------------------------------------------------------
-----
 * 설명 : 머신을 위한 기계 코드목록 파일(inst_table.txt)을 읽어
 *       기계어 목록 테이블(inst_table)을 생성하는 함수이다.
 *
 *
 * 매계 : 기계어 목록 파일
 * 반환 : 정상종료 = 0 , 에러 < 0
 * 주의 : 기계어 목록파일 형식은 자유롭게 구현한다. 예시는 다음과 같다.
 *
 *
==============================================================================
==
 *                | 이름 | 형식 | 기계어 코드 | 오퍼랜드의 갯수 | ₩n |
 *
==============================================================================
==
 *
 *
------------------------------------------------------------------------------
-----
 */
int init_inst_file(char* inst_file)
{
        FILE* file;
        int errno;
        char buffer[255];
        errno = fopen_s(&file, inst_file, "r");
        if (errno !=0)
        {
                printf("파일 열기 실패₩n");
                return -1;
        }
        fseek(file, 0, SEEK_SET);
        while (fgets(buffer, sizeof(buffer), file) !=NULL)
        {
```

```
                inst* new_inst = (inst*)malloc(sizeof(inst));
                if (new_inst ==NULL)
                {
                        printf("메모리 할당 실패₩n");
                        return -1;
                }
                unsigned int op_tmp;
                char format[10];

                if (sscanf_s(buffer, "%s₩t%s₩t%x₩t%d",
                        new_inst->str, (unsigned int)sizeof(new_inst->str),
                        format, (unsigned int)sizeof(format),
                        &op_tmp,
                        &new_inst->ops) !=4) {
                        printf("inst_table.txt 파일을 파싱하는데 실패했습니다.₩n");
                        return -1;
                }
                //opcode 저장
                new_inst->op = (unsigned char)op_tmp;
            /*
             * format 저장 :
                * format == "3/4" 이면 일단 ops == 3을 저장하고, Input_data에서 + 를 만나면
ops == 4로 업데이트 한다.
                */
                if(format[1] =='/'){
                        new_inst->format =3;
                }
                else {
                        new_inst->format = format[0] -'0';
                }
                // inst_table에 저장
                inst_table[inst_index++] = new_inst;
        }
        fclose(file);
        return errno;
}
/                                                                                *
-------------------------------------------------------------------------------
-----
 * 설명 : 어셈블리 할 소스코드 파일(input.txt)을 읽어 소스코드 테이블(input_data)를 생성하는 함수이
다.
 * 매계 : 어셈블리할 소스파일명
 * 반환 : 정상종료 = 0 , 에러 < 0
 * 주의 : 라인단위로 저장한다.
 *
 *
-------------------------------------------------------------------------------
-----
 */
int init_input_file(char* input_file)
{
        FILE* file;
        int errno;
        char buffer[MAX_LINE_LENGTH];
        errno = fopen_s(&file, input_file, "r");
        if (errno !=0)
        {
                printf("파일 열기 실패₩n");
```

```
                return -1;
        }
        fseek(file, 0, SEEK_SET);
        // 파일을 라인 단위로 읽어 input_data에 저장
        // MAXLINES을 넘을 경우 중단
        while (fgets(buffer, sizeof(buffer), file) !=NULL && line_num < MAX_LINES)
        {
                char* temp = _strdup(buffer);
                if (temp ==NULL) {
                        printf("메모리 할당 실패 at line %d₩n", line_num);
                        break;
                }
                input_data[line_num++] = temp;
        }
        // MAXLINES을 넘을 경우 에러
        if (line_num >= MAX_LINES) {
                printf("최대 라인수를 초과하였습니다.₩n");
                return -1;
        }
        fclose(file);
        return errno;
}
/                                                                                *
-------------------------------------------------------------------------------
-----
 * 설명 : 소스 코드를 읽어와 토큰단위로 분석하고 토큰 테이블을 작성하는 함수이다.
 *        패스 1로 부터 호출된다.
 * 매계 : 파싱을 원하는 문자열
 * 반환 : 정상종료 = 0 , 에러 < 0
 * 주의 : my_assembler 프로그램에서는 라인단위로 토큰 및 오브젝트 관리를 하고 있다.
 *
-------------------------------------------------------------------------------
-----
 */
int token_parsing(char* str)
{
        // 임시토큰 초기화
        token* tok = (token*)malloc(sizeof(token));
        if (tok ==NULL) {
                printf("메모리 할당 실패₩n");
                return -1;
        }
        tok->label =NULL;
        tok->oper =NULL;
        for (int i =0; i < MAX_OPERAND; i++)    tok->operand[i] =NULL;
        tok->comment[0] ='₩0';
        // 주석처리
        if (str[0] =='.') {
                strcpy_s(tok->comment,sizeof(tok->comment), str);
                token_table[token_line] = tok;
                token_line++;
                return 0;
        }
        char* tmp_label =NULL;
        char* tmp_oper =NULL;
        char* tmp_str = _strdup(str);
        char* tmp_operands =NULL;
        char* tmp_comment =NULL;
```

```
        char* context =NULL;    // strtok_s()의 context 변수
        int num_operands = MAX_OPERAND;     // 피연산자 개수
        // label이 없는경우 : '₩t'로 시작
        if (str[0] =='₩t') {
                tmp_str = tmp_str +1;
                tmp_oper = strtok_s(tmp_str, " ₩t₩n", &context);
        }
        //label이 있는 경우
        else {
                tmp_label = strtok_s(tmp_str, " ₩t₩n", &context);
                tmp_oper = strtok_s(NULL, " ₩t₩n", &context);
        }
        if (tmp_oper !=NULL) {
                tok->oper = _strdup(tmp_oper);
                // operator가 NULL이 아닌경우 최대 피연산자 개수를 가져온다
                for (int i =0; i < inst_index; i++) {
                        const char* opcheck = (tmp_oper[0] =='+') ? tmp_oper +1 : tmp_oper;
                        if (strcmp(inst_table[i]->str, opcheck) ==0) {
                                num_operands = inst_table[i]->ops;
                                break;
                        }
                }
        }

        // operand 개수가 0 ex) RSUB인 경우에는 나머지 문자열을 모두 comment처리한다.
        if (num_operands >0)    tmp_operands = strtok_s(NULL, " ₩t₩n", &context);
        tmp_comment = strtok_s(NULL, "", &context);     //남은 문자열을 모두 주석으로 처리
        // label과 존재시 할당
        if (tmp_label !=NULL) {
                tok->label = _strdup(tmp_label);
        }
    /* operand parsing :
        * operand는 ','로 구분되어 있다.
        * tmp_operands로 따로 구분한 이유는 comment의 ','와 구분하기 위해서이다.
        */
        if (tmp_operands !=NULL) {
                char* operand_context =NULL;   // strtok_s()의 context 변수
                char* tmp_operand = strtok_s(tmp_operands, ",", &operand_context);
                int i =0;
                while (tmp_operand !=NULL && i < MAX_OPERAND) {
                        tok->operand[i++] = _strdup(tmp_operand);
                        tmp_operand = strtok_s(NULL, ",", &operand_context);
                }
        }
        // comment는 ','와 상관없이 통째로 저장
        if (tmp_comment !=NULL) {
                //comment 공백제거
                while (*tmp_comment ==' '||*tmp_comment =='₩t') tmp_comment++;
                strcpy_s(tok->comment,sizeof(tok->comment), tmp_comment);
        }
        token_table[token_line++] = tok;
        return 0;
}
/                                                                               *
-------------------------------------------------------------------------------
-----
* 설명 : 어셈블리 코드를 위한 패스1과정을 수행하는 함수이다.
*                  패스1에서는..
```

```
*                  1. 프로그램 소스를 스캔하여 해당하는 토큰단위로 분리하여 프로그램 라인별 토큰
*                  테이블을 생성한다.
*          2. 토큰 테이블은 token_parsing()을 호출하여 설정한다.
*          3. assem_pass2 과정에서 사용하기 위한 심볼테이블 및 리터럴 테이블을 생성한다.
*
*
*
* 매계 : 없음
* 반환 : 정상 종료 = 0 , 에러 = < 0
* 주의 : 현재 초기 버전에서는 에러에 대한 검사를 하지 않고 넘어간 상태이다.
*            따라서 에러에 대한 검사 루틴을 추가해야 한다.
*
*        OPCODE 출력 프로그램에서는 심볼테이블, 리터럴테이블을 생성하지 않아도 된다.
*        그러나, 추후 프로젝트 1을 수행하기 위해서는 심볼테이블, 리터럴테이블이 필요하다.
*
*
------------------------------------------------------------------------------
------
*/
static int assem_pass1(void)
{
        csect_start_token_line[0] =0;     // CSECT 시작 토큰 라인 저장
        bool start_encounter =false;      // START 여부
        int start_symtab =0;      // sym table 서칭 시작 인덱스
        /* input_data의 문자열을 한줄씩 입력 받아서
         * token_parsing()을 호출하여 _token에 저장
         */
        for (int i =0; i < line_num; i++) {
                if (token_parsing(input_data[i]) <0) {
                        printf("token_parsing: 토큰 파싱에 실패하였습니다.₩n");
                        return -1;
                }
                // 파싱된 토큰
                token* t = token_table[i];
                t->addr =-1; // 주소 초기화
                if (!t ||!t->oper) continue; // NULL 체크
                //comment line
                if (t->oper ==NULL && t->comment[0] =='.') {
                        // 주석 라인
                        continue;
                }
                //START
                if (strcmp(t->oper, "START") ==0) {
                        // START가 2회 이상 나오면 에러
                        if (start_encounter) {
                                printf("START 명령어가 중복되었습니다.₩n");
                                return -1;
                        }
                        // 프로그램 시작 주소를 설정
                        if (t->operand[0] !=NULL) {
                                locctr = (int)strtol(t->operand[0], NULL, 16);
                        }
                        else {
                                locctr =0; // 시작주소가 없으면 locctr를 0으로 초기화
                        }
                        start_addr = locctr;       // 프로그램 시작 주소 저장
                        start_encounter =true;    // START가 나왔음을 표시
                        // START label저장
```

```
                if (t->label !=NULL) {
                        if (insert_symtab(start_symtab, t->label, locctr) <0) return -1;
                }
                t->addr = locctr;       // locctr를 addr에 저장
                continue;       // START는 locctr를 증가시키지 않음
        }
        // CSECT
        if (t->oper && strcmp(t->oper, "CSECT") ==0) {
                assign_littab_addresses();
                csect_start_symbol_num[csect_num++] = label_num; // CSECT 시작 심볼 번호 저장
                csect_start_token_line[csect_num] = i; // CSECT 시작 토큰 라인 저장
                start_symtab = label_num;       // 다음 label부터 시작
                insert_symtab(start_symtab, "", NULL); // 빈 label 섹터 구분용
                locctr =0;
                if (t->label) insert_symtab(start_symtab, t->label, locctr); // CSECT 종료 위치 저장
                t->addr = locctr;       // locctr를 addr에 저장
                continue;       // LOCCTR 업데이트 없음
        }
        // label 저장
        if (t->label !=NULL && strcmp(t->oper, "EQU") !=0 ) {
                if (insert_symtab(start_symtab, t->label, locctr) <0) return -1;
        }
        // EQU
        if (strcmp(t->oper, "EQU") ==0) {
                int value =0;
                // 피연산자가 '*'인 경우
                if (strcmp(t->operand[0], "*") ==0) {
                        value = locctr;
                }
                // 피연산자가 숫자인 경우
                else if (isdigit(t->operand[0][0])) {
                        value = atoi(t->operand[0]);
                }
                // '-'연산 처리
                else if(strchr(t->operand[0], '-') !=NULL) {
                        char operand_copy[100];
                        strcpy_s(operand_copy, sizeof(operand_copy), t->operand[0]);
                        char* left = strtok(operand_copy, "-");
                        char* right = strtok(NULL, "-");
                        int left_val = get_symval(start_symtab , left);
                        int right_val = get_symval(start_symtab ,right);
                        value = (left_val ==-1 || right_val ==-1) ? 0 : left_val - right_val;
                }
                // 피연산자가 심볼인 경우
                else {
                        int symvalue = get_symval(start_symtab, t->operand[0]);
                        value = (symvalue ==-1) ? 0 : symvalue;         //심볼을 찾지 못한경우 0으로 채움
                }
                // 심볼 테이블에 저장
                if (t->label !=NULL) {
                        if (insert_symtab(start_symtab, t->label, value) <0) return -1;
                }
                t->addr = locctr;       // locctr를 addr에 저장
                continue; // LOCCTR 업데이트 없음
```

```
        }
        //LTORG
        if (strcmp(t->oper, "LTORG") ==0) {
                assign_littab_addresses();
                continue;
        }
        // 리터럴 처리
        if (t->operand[0] !=NULL && t->operand[0][0] =='=') {
                // 리터럴 테이블에 추가
                insert_literaltab(t->operand[0]);
        }
        t->addr = locctr;          // locctr를 addr에 저장

        // 지시어 처리
        //BYTE
        if (strcmp(t->oper, "BYTE") ==0) {
                int added_byte =0;
                if (t->operand[0][0] =='C') {
                        added_byte = strlen(t->operand[0]) -3; // C'abc' -> 3
                }
                else if (t->operand[0][0] =='X') {
                        added_byte = (strlen(t->operand[0]) -3) /2; // X'12' -> 2
                }
                else {
                        return -1; // 잘못된 BYTE 형식
                }
                locctr += added_byte;
        }
        //WORD
        else if (strcmp(t->oper, "WORD") ==0) {
                locctr +=3;
        }
        //RESB
        else if (strcmp(t->oper, "RESB") ==0) {
                if (t->operand[0] !=NULL) {
                        locctr += atoi(t->operand[0]);
                }
        }
        //RESW
        else if (strcmp(t->oper, "RESW") ==0) {
                if (t->operand[0] !=NULL) {
                        locctr +=3 * atoi(t->operand[0]);
                }
        }
        else{
        // opcode 검색
                int idx = search_opcode(t->oper);
                if (idx >=0) {
                        if (t->oper[0] =='+') {
                                // 4형식인 경우
                                locctr += inst_table[idx]->format +1;
                        }
                        else {
                                locctr += inst_table[idx]->format;
                        }
                }
        }
}
```

```
        // locctr 업데이트
        assign_littab_addresses();
        return 0;
}
/                                                                                *
-------------------------------------------------------------------------------
-----
 * 설명 : 입력 문자열이 기계어 코드인지를 검사하는 함수이다.
 * 매개 : 토큰 단위로 구분된 문자열
 * 반환 : 정상종료 = 기계어 테이블 인덱스, 에러 < 0
 * 주의 : 기계어 목록 테이블에서 특정 기계어를 검색하여, 해당 기계어가 위치한 인덱스를 반환한다.
 *        '+JSUB'과 같은 문자열에 대한 처리는 자유롭게 처리한다.
 *
 *
-------------------------------------------------------------------------------
-----
 */
int search_opcode(char* str)
{
        // operator가 NULL인 경우
        if (str ==NULL)
        {
                return -1;
        }
        //4형식 처리 : +로 시작
        if (str[0] =='+')
        {
                str++;   // +를 제거

        }
        for (int i =0; i < inst_index; i++)
        {
                if (strcmp(inst_table[i]->str, str) ==0)
                {
                        return i;
                }
        }
        return -1;       // 찾지 못한 경우
}
/                                                                                *
-------------------------------------------------------------------------------
-----
* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
*
* 매개 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 소스코드 명령어 앞에 OPCODE가 기록된 코드를 파일에 출력한다.
*        파일이 NULL값이 들어온다면 프로그램의 결과를 stdout으로 보내어
*        화면에 출력해준다.
*
*        OPCODE 출력 프로그램의 최종 output 파일을 생성하는 함수이다.
*        (추후 프로젝트 1에서는 불필요)
*
*
-------------------------------------------------------------------------------
------
*/
void make_opcode_output(char* file_name)
```

```
{
	FILE* file;
	int errno;
	if (file_name ==NULL)
	{
		file = stdout;
	}
	else {
		errno = fopen_s(&file, file_name, "w");
		if (errno !=0)
		{
			printf("파일 열기 실패₩n");
			return;
		}
	}
	fseek(file, 0, SEEK_SET);
	for (int i =0; i < token_line; i++)
	{
		token* t = token_table[i];
		// 주석 라인
		if (t->oper ==NULL)
		{
			fprintf(file, "%s₩n", t->comment);
			continue;
		}
		// operand 문자열 만들기
		char operand_str[100] ="";
		for (int j =0; j < MAX_OPERAND; j++) {
			if (t->operand[j] !=NULL) {
				if (operand_str[0] !='₩0') {
					strcat_s(operand_str, sizeof(operand_str), ",");
				}
				strcat_s(operand_str, sizeof(operand_str), t->operand[j]);
			}
		}
		// opcode 문자열 만들기
		char opcode_str[10] ="";
		int idx = search_opcode(t->oper);
		if (idx !=-1) {
			sprintf_s(opcode_str, sizeof(opcode_str), "%02X", inst_table[idx]->op);
		}
		// 정렬된 출력
		fprintf(file, "%-8s%-8s%-16s%-8s₩n",
			t->label ? t->label : "",
			t->oper ? t->oper : "",
			operand_str,
			opcode_str);
	}
	if (file != stdout)
	{
		fclose(file);
	}
	return;
}
/                                                                                *
--------------------------------------------------------------------------------
-----
* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
```

```
*        여기서 출력되는 내용은 SYMBOL별 주소값이 저장된 TABLE이다.
* 매개 : 생성할 오브젝트 파일명 혹은 경로
* 반환 : 없음
* 주의 : 파일이 NULL값이 들어온다면 프로그램의 결과를 stdout으로 보내어
*        화면에 출력해준다.
*
*
------------------------------------------------------------------------------
------
*/
void make_symtab_output(char* file_name)
{
        FILE* file;
        if (file_name ==NULL)
                file = stdout;
        else
                // 파일 열기
                fopen_s(&file, file_name, "w");
        for (int i =0; i < label_num; i++) {
                if(strcmp(sym_table[i].symbol, "") ==0){
                        fprintf(file, "₩n");
                }
                else if (sym_table[i].symbol !=NULL) {
                        fprintf(file, "%s₩t₩t%X₩n", sym_table[i].symbol, sym_table[i].addr);
                }
        }
        fprintf(file, "₩n");
        if (file_name !=NULL)
                fclose(file);       // 파일 닫기
}
/                                                                              *
------------------------------------------------------------------------------
-----
* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
*        여기서 출력되는 내용은 LITERAL별 주소값이 저장된 TABLE이다.
* 매개 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 파일이 NULL값이 들어온다면 프로그램의 결과를 stdout으로 보내어
*        화면에 출력해준다.
*
*
------------------------------------------------------------------------------
------
*/
void make_literaltab_output(char* filename)
{
        FILE* file;
        if (filename ==NULL)
                file = stdout;
        else
                // 파일 열기
                fopen_s(&file, filename, "w");
        for (int i =0; i < literal_num ; i++) {
                if (strcmp(literal_table[i].symbol, "") ==0) {
                        fprintf(file, "₩n");
                }
                else if (literal_table[i].symbol !=NULL) {
                        char tmp[100];
```

```
                        strncpy_s(tmp,                          sizeof(tmp),literal_table[i].symbol+3,
strlen(literal_table[i].symbol) -4);
                        fprintf(file, "%s₩t₩t%X₩n", tmp, literal_table[i].addr);
                }
        }
        fprintf(file, "₩n");
        if (filename !=NULL)
                fclose(file);       // 파일 닫기
}
/                                                                                                   *
------------------------------------------------------------------------------------------
------
 * 설명 : 어셈블리 코드를 기계어 코드로 바꾸기 위한 패스2 과정을 수행하는 함수이다.
 *                 패스 2에서는 프로그램을 기계어로 바꾸는 작업은 라인 단위로 수행된다.
 *                 다음과 같은 작업이 수행되어 진다.
 *                 1. 실제로 해당 어셈블리 명령어를 기계어로 바꾸는 작업을 수행한다.
 * 매개 : 없음
 * 반환 : 정상종료 = 0, 에러발생 = < 0
 * 주의 :
 *
------------------------------------------------------------------------------------------
------
 */
static int assem_pass2(void)
{
        csect_num =0;
        int start_symtab =0;            // sym table 서칭 시작 인덱스
        bool mainroutine =false; // 메인루틴 여부
        int last_head_idx =-1;          // 마지막 헤더 레코드 인덱스
        int text_length =0;             // 텍스트 레코드 길이
        int text_start =-1;             // 텍스트 레코드 시작 주소
        char text_buffer[70] ="";// 텍스트 레코드 버퍼
        // EXTREF, EXTDEF 목록 저장용
        char extdef_list[10][MAX_LINE_LENGTH]; int extdef_count =0;
        char extref_list[10][MAX_LINE_LENGTH]; int extref_count =0;
        for (int i =0; i < token_line; i++) {
                token* t = token_table[i];
                object_code oc;
                char obj[10] ="";               // 객체 코드 버퍼
                // 주석 라인
                if (!t->oper || t->oper[0] =='.') continue;
                // START 처리
                if (t->oper !=NULL && strcmp(t->oper, "START") ==0) {
                        last_head_idx = obj_count;      // 헤더 레코드 인덱스 저장
                        mainroutine =true;
                        // start 심볼은 H로 시작
                        oc.record_type ='H';
                        oc.locctr = t->addr;    // locctr 저장
                        // object_code를 공백으로 초기화하고, 프로그램 이름 복사
                        memset(oc.object_code, ' ', 6);  // 6칸 전부 공백
                        if (t->label !=NULL) {
                                size_t len = strlen(t->label);
                                if (len >6) len =6;
                                memcpy(oc.object_code, t->label, len);  // 앞에서부터 복사
                        }
                        oc.object_code[6] ='₩0';
                        obj_codes[obj_count++] = oc;  // 객체 코드 저장
                        continue;
```

```
                }
                // EXTDEF 처리 → D 레코드
                if (strcmp(t->oper, "EXTDEF") ==0) {
                        for (int j =0; j < MAX_OPERAND && t->operand[j]; j++) {
                                strcpy_s(extdef_list[extdef_count++],           MAX_LINE_LENGTH,
t->operand[j]);
                        }
                        // EXTDEF가 끝나면 D 레코드 생성
                        if (extdef_count >0) {
                                object_code drec = { 'D', -1, -1, "" };
                                for (int k =0; k < extdef_count; k++) {
                                        int addr = get_symval(start_symtab, extdef_list[k]);
                                        char name[7], entry[20];
                                        snprintf(name,   sizeof(name),   "%-6s",   extdef_list[k]);
 // 이름: 6자 공백 패딩
                                        snprintf(entry,   sizeof(entry),   "%s%06X",   name,   addr);
 // 전체: 이름+주소
                                        strcat_s(drec.object_code,         sizeof(drec.object_code),
entry);
                                }
                                obj_codes[obj_count++] = drec;
                        }
                        continue;
                }
                // EXTREF 처리 → R 레코드
                if (strcmp(t->oper, "EXTREF") ==0) {
                        for (int j =0; j < MAX_OPERAND && t->operand[j]; j++) {
                                strcpy_s(extref_list[extref_count++],           MAX_LINE_LENGTH,
t->operand[j]);
                        }
                        // R 레코드 생성
                        if (extref_count >0) {
                                object_code rrec = { 'R', -1, -1, "" };
                                for (int k =0; k < extref_count; k++) {
                                        char buffer[7];
                                        snprintf(buffer, sizeof(buffer), "%-6s", extref_list[k]);    //
이름: 6자 공백 패딩
                                        strcat_s(rrec.object_code,         sizeof(rrec.object_code),
buffer);
                                }
                                obj_codes[obj_count++] = rrec;
                        }
                        continue;
                }
                // CSECT 처리
                else if (strcmp(t->oper, "CSECT") ==0) {
                        start_symtab = csect_start_symbol_num[csect_num++];   // 다음 label부터
시작
                        // 이전 T 레코드 종료
                        if (text_length >0) {
                                object_code rec = { 'T', text_start, text_length, "" };
                                strcpy_s(rec.object_code, sizeof(rec.object_code), text_buffer);
                                obj_codes[obj_count++] = rec;  // 객체 코드 저장
                        }
                        text_length =0;  // 텍스트 레코드 길이 초기화
                        text_start =-1;  // 텍스트 레코드 시작 주소 초기화
                        memset(text_buffer, 0, sizeof(text_buffer));        // 버퍼 초기화
                        // 이전 헤더 레코드 길이 업데이트
```

```
                        if (last_head_idx !=-1 ) {
                                int end_addr = get_last_allocated_addr(i);
                                if (end_addr >=0) {
                                        obj_codes[last_head_idx].length      =      end_addr      -
obj_codes[last_head_idx].locctr;
                                }
                        }
                        // Modify 레코드 추가
                        int sect_start = csect_start_token_line[csect_num -1];
                        int      sect_end      =      csect_start_token_line[csect_num]      !=0      ?
csect_start_token_line[csect_num] : token_line;
                        for (int j = sect_start; j < sect_end; j++) {
                                token* tk = token_table[j];
                                // 4형식인 경우
                                if (tk->oper !=NULL && tk->oper[0] =='+') {
                                        // EXTREF 안에 있는 operand만 수정 대상
                                        for (int r =0; r < extref_count; r++) {
                                                if  (tk->operand[0]  &&  strcmp(tk->operand[0],
extref_list[r]) ==0) {
                                                        object_code mrec = { 'M', tk->addr +1,
5, "" }; // 주소 +1, 길이 5
                                                        snprintf(mrec.object_code,
sizeof(mrec.object_code), "05+%s", extref_list[r]);
                                                        obj_codes[obj_count++] = mrec;
                                                }
                                        }
                                }
                                // WORD 명령어에서 SYM1 - SYM2 형식 확인
                                else if (tk->oper !=NULL && strcmp(tk->oper, "WORD") ==0) {
                                        //-여부 확인
                                        char* minus = strchr(tk->operand[0], '-');
                                        if (minus) {
                                                char                    left[MAX_LINE_LENGTH],
right[MAX_LINE_LENGTH];
                                                strncpy_s(left, sizeof(left), tk->operand[0], minus
- tk->operand[0]);
                                                strcpy_s(right, sizeof(right), minus +1);
                                                for (int r =0; r < extref_count; r++) {
                                                        if (strcmp(left, extref_list[r]) ==0) {
                                                                object_code   mrec   =   {   'M',
tk->addr, 6, "" };
                                                                sprintf_s(mrec.object_code,
sizeof(mrec.object_code), "06+%s", extref_list[r]);
                                                                obj_codes[obj_count++] = mrec;
                                                        }
                                                }
                                                for (int r =0; r < extref_count; r++) {
                                                        if (strcmp(right, extref_list[r]) ==0) {
                                                                object_code   mrec   =   {   'M',
tk->addr, 6, "" };
                                                                sprintf_s(mrec.object_code,
sizeof(mrec.object_code), "06-%s", extref_list[r]);
                                                                obj_codes[obj_count++] = mrec;
                                                        }
                                                }
                                        }
                                }
                        }
```

```
                    // END 레코드 추가
                    if (mainroutine) {
                            object_code end_rec = { 'E', -1, -1, "" };
                            end_rec.locctr = start_addr;      // 시작 주소
                            obj_codes[obj_count++] = end_rec;      // 객체 코드 저장
                    }
                    else {
                            object_code end_rec = { 'E', -1, -1, "" };
                            obj_codes[obj_count++] = end_rec;      // 객체 코드 저장
                    }
                    mainroutine =false;       // 메인루틴 종료
                    // 새로운 H 레코드 추가
                    object_code new_rec = { 'H', t->addr, 0, "" };
                    memset(new_rec.object_code, ' ', 6);  // 6칸 전부 공백
                    if (t->label !=NULL) {
                            size_t len = strlen(t->label);
                            if (len >6) len =6;
                            memcpy(new_rec.object_code, t->label, len);  // 앞에서부터 복사
                    }
                    new_rec.object_code[6] ='₩0';  // 문자열 종료
                    new_rec.length =0;                          // 나중에 다시 채움
                    last_head_idx = obj_count;        // 헤더 레코드 인덱스 저장
                    obj_codes[obj_count++] = new_rec;      // 객체 코드 저장
                    //  ext 리스트 초기화
                    for (int k =0; k <10; k++) {
                            memset(extdef_list[k],'₩0', MAX_LINE_LENGTH);
                            memset(extref_list[k], '₩0', MAX_LINE_LENGTH);
                    }
                    extdef_count =0;
                    extref_count =0;
                    continue;
            }
            // END 처리
            else if (strcmp(t->oper, "END") ==0) {
                    csect_num++;
                    //리터럴 처리 (끊지 않고 text_buffer에 그대로 붙이기)
                    for (int j =0; j < literal_num; j++) {
                            if (literal_line_index[j] <= i && literal_table[j].addr !=-1) {
                                    int len =0;
                                    char obj[70] ="";
                                    // =C'...' 처리
                                    if (literal_table[j].symbol[1] =='C') {
                                            char* lit = literal_table[j].symbol +3;  // =C'EOF'
→ 'EOF'
                                            for (int k =0; lit[k] !='₩''&& lit[k] !='₩0'; k++) {
                                                    char hex[3];
                                                    sprintf_s(hex, sizeof(hex), "%02X", lit[k]);
                                                    strcat_s(obj, sizeof(obj), hex);
                                            }
                                            len = strlen(obj) /2;
                                    }
                                    // =X'...' 처리
                                    else if (literal_table[j].symbol[1] =='X') {
                                            strncpy_s(obj,  sizeof(obj),  literal_table[j].symbol
+3, strlen(literal_table[j].symbol) -4);
                                            obj[strlen(literal_table[j].symbol) -4] ='₩0';
                                            len = strlen(obj) /2;
                                    }
```

```
                                        // 텍스트 레코드 초과시 끊기
                                        if (text_length + len >30) {
                                                object_code rec = { 'T', text_start, text_length, ""
};
                                                strcpy_s(rec.object_code, sizeof(rec.object_code),
text_buffer);
                                                obj_codes[obj_count++] = rec;
                                                text_start =-1;
                                                text_length =0;
                                                j--;
                                                memset(text_buffer, 0, sizeof(text_buffer));
                                                continue;
                                        }
                                        // 텍스트 시작 주소 설정
                                        if (text_start ==-1)
                                                text_start = literal_table[j].addr;
                                        strcat_s(text_buffer, sizeof(text_buffer), obj);
                                        text_length += len;
                                }
                        }
                        //남은 text_buffer 한 번에 저장
                        if (text_length >0) {
                                object_code rec = { 'T', text_start, text_length, "" };
                                strcpy_s(rec.object_code, sizeof(rec.object_code), text_buffer);
                                obj_codes[obj_count++] = rec;
                                text_length =0;
                                text_start =-1;
                                memset(text_buffer, 0, sizeof(text_buffer));
                        }
                        //헤더 레코드 길이 설정
                        if (last_head_idx !=-1) {
                                int end_addr = get_last_allocated_addr(i);
                                if (end_addr >=0) {
                                        obj_codes[last_head_idx].length      =      end_addr      -
obj_codes[last_head_idx].locctr;
                                }
                        }
                        int sect_start = csect_start_token_line[csect_num -1];
                        int      sect_end      =      csect_start_token_line[csect_num]      !=0      ?
csect_start_token_line[csect_num] : token_line;
                        // Modify 레코드 추가
                        for (int j = sect_start; j < sect_end; j++) {
                                token* tk = token_table[j];
                                // 4형식인 경우 5비트 수정
                                if (tk->oper !=NULL && tk->oper[0] =='+') {
                                        // EXTREF 안에 있는 operand만 수정 대상
                                        for (int r =0; r < extref_count; r++) {
                                                if  (tk->operand[0]  &&  strcmp(tk->operand[0],
extref_list[r]) ==0) {
                                                        object_code mrec = { 'M', tk->addr +1,
5, "" }; // 주소 +1, 길이 5
                                                        snprintf(mrec.object_code,
sizeof(mrec.object_code), "05+%s", extref_list[r]);
                                                        obj_codes[obj_count++] = mrec;
                                                }
                                        }
                                }
                                // WORD 명령어에서 SYM1 - SYM2 형식 확인
```

```
                                else if (tk->oper !=NULL && strcmp(tk->oper, "WORD") ==0) {
                                        //-여부 확인
                                        char* minus = strchr(tk->operand[0], '-');
                                        if (minus) {
                                                char                    left[MAX_LINE_LENGTH],
right[MAX_LINE_LENGTH];
                                                strncpy_s(left, sizeof(left), tk->operand[0], minus
- tk->operand[0]);
                                                strcpy_s(right, sizeof(right), minus +1);
                                                for (int r =0; r < extref_count; r++) {
                                                        if (strcmp(right, extref_list[r]) ==0) {
                                                                object_code   mrec   =   {   'M',
tk->addr, 6, "" };
                                                                sprintf_s(mrec.object_code,
sizeof(mrec.object_code), "06-%s", extref_list[r]);
                                                                obj_codes[obj_count++] = mrec;
                                                        }
                                                }
                                                for (int r =0; r < extref_count; r++) {
                                                        if (strcmp(left, extref_list[r]) ==0) {
                                                                object_code   mrec   =   {   'M',
tk->addr, 6, "" };
                                                                sprintf_s(mrec.object_code,
sizeof(mrec.object_code), "06+%s", extref_list[r]);
                                                                obj_codes[obj_count++] = mrec;
                                                        }
                                                }
                                        }
                                }
                        }
                        //E 레코드 추가
                        object_code end_rec = { 'E', -1, -1, "" };
                        end_rec.locctr = mainroutine ? start_addr : -1;
                        obj_codes[obj_count++] = end_rec;
                        mainroutine =false;
                        continue;
                }
                // RESB, RESW 처리 :얘들을 만나면 obj코드 개행을 해야함
                else if (strcmp(t->oper, "RESB") ==0 || strcmp(t->oper, "RESW") ==0) {
                        if (text_length >0) {
                                object_code rec = { 'T', text_start, text_length, "" };
                                strcpy_s(rec.object_code, sizeof(rec.object_code), text_buffer);
                                obj_codes[obj_count++] = rec;  // 객체 코드 저장
                                text_length =0;  // 텍스트 레코드 길이 초기화
                                text_start =-1;  // 텍스트 레코드 시작 주소 초기화
                                memset(text_buffer, 0, sizeof(text_buffer));        // 버퍼 초기화
                        }
                        continue;
                }
                else if (strcmp(t->oper, "LTORG") ==0) {
                        for (int j =0; j < literal_num; j++) {
                                // LTORG 이전에 등장한 리터럴 중 아직 처리 안 한 것만
                                if (literal_line_index[j] <= i && literal_table[j].addr !=-1) {
                                        int len =0;
                                        char obj[70] ="";
                                        if (literal_table[j].symbol[1] =='C') {
                                                char* lit = literal_table[j].symbol +3;   // =C'EOF'
→ 'EOF'
```

```
                                        for (int k =0; lit[k] !='₩''&& lit[k] !='₩0'; k++) {
                                                char hex[3];
                                                sprintf_s(hex, sizeof(hex), "%02X", lit[k]);
                                                strcat_s(obj, sizeof(obj), hex);
                                        }
                                        len = strlen(obj) /2;
                                }
                                else if (literal_table[j].symbol[1] =='X') {
                                        strncpy_s(obj,  sizeof(obj),  literal_table[j].symbol
+3, strlen(literal_table[j].symbol) -4);
                                        obj[strlen(literal_table[j].symbol) -4] ='₩0';
                                        len = strlen(obj) /2;
                                }
                                // 텍스트 시작점 초기화
                                if (text_start ==-1) text_start = literal_table[j].addr;
                                // text 길이 초과 시 끊기
                                if (text_length + len >30) {
                                        object_code rec = { 'T', text_start, text_length, ""
};
                                        strcpy_s(rec.object_code, sizeof(rec.object_code),
text_buffer);
                                        obj_codes[obj_count++] = rec;
                                        text_start =-1;
                                        text_length =0;
                                        memset(text_buffer, 0, sizeof(text_buffer));
                                        // 다시 처리
                                        j--;
                                        continue;
                                }
                                // 리터럴 주소를 -1로 설정하여 처리 완료 표시
                                literal_table[j].addr =-1;
                                strcat_s(text_buffer, sizeof(text_buffer), obj);
                                text_length += len;
                        }
                }
                continue;
        }
        // BYTE처리
        else if (strcmp(t->oper, "BYTE") ==0) {
                if (text_start ==-1) text_start = t->addr; // 텍스트 레코드 시작 주소 저장
                if (t->operand[0][0] =='X') {
                        strncpy_s(obj,          sizeof(obj),          t->operand[0]          +2,
strlen(t->operand[0]) -3);          // X'12' -> 12
                }
                else if (t->operand[0][0] =='C') {
                        int len = strlen(t->operand[0]);
                        for (int j =2; j < len -1; j++) {
                                char hex[3] ="";
                                sprintf_s(hex, sizeof(hex), "%02X", t->operand[0][j]);
                                strcat_s(obj, sizeof(obj), hex);    // C'abc' -> 616263
                        }
                }
                if (text_length + (strlen(obj) /2) >30) {
                        object_code rec = { 'T', text_start, text_length, "" };
                        strcpy_s(rec.object_code, sizeof(rec.object_code), text_buffer);
                        obj_codes[obj_count++] = rec;  // 객체 코드 저장
                        text_length =0;  // 텍스트 레코드 길이 초기화
                        text_start =-1;  // 텍스트 레코드 시작 주소 초기화
```

```
                        i--;                                    // 다시 읽기
                        memset(text_buffer, 0, sizeof(text_buffer));        // 버퍼 초기화
                }
        }
        //WORD 처리
        else if (strcmp(t->oper, "WORD") ==0) {
                int value =0;
                bool is_external =false;  // 외부 심볼 여부
                if (t->operand[0] !=NULL) {
                        if (get_symval(start_symtab, t->operand[0]) ==-1) {
                                // 심볼이 없는 경우
                                is_external =true;
                                value =0;                               // 심볼을 찾지 못한 경우
0으로 초기화
                        }
                        else {
                                // 심볼이 있는 경우
                                value = get_symval(start_symtab, t->operand[0]);
// 심볼 주소 가져오기
                        }
                }
                sprintf_s(obj, sizeof(obj), "%06X", value);
                // 텍스트 레코드 길이 넘는지 체크
                if (text_length +3 >30) {
                        object_code rec = { 'T', text_start, text_length, "" };
                        strcpy_s(rec.object_code, sizeof(rec.object_code), text_buffer);
                        obj_codes[obj_count++] = rec;
                        text_start =-1;
                        i--;    // 다시 읽기
                        text_length =0;
                        memset(text_buffer, 0, sizeof(text_buffer));
                }
        }
        // 일반 명령어
        else{
                if (text_start ==-1) {
                        text_start = t->addr;     // 텍스트 레코드 시작 주소 저장
                }
                // opcode 검색
                int idx = search_opcode(t->oper);
                if (idx <0) continue;
// opcode가 유효하지 않음
                unsigned char opcode = inst_table[idx]->op;             // opcode 가져
오기
                int format = inst_table[idx]->format;                   // 포맷 가져오기
                if (t->oper[0] =='+') format++;                         // 4형식인 경우
                // 넘치는지 체크
                if (text_length + format >30) {
                        object_code rec = { 'T', text_start, text_length, "" };
                        strcpy_s(rec.object_code, sizeof(rec.object_code), text_buffer);
                        obj_codes[obj_count++] = rec;
                        text_length =0;
                        text_start = t->addr;
                        memset(text_buffer, 0, sizeof(text_buffer));
                        i--;
                        continue;
                }
                //nixbpe 비트 설정
```

```
                        set_nixbpe(t, format);
                        // object code 생성
                        int disp =0; int target =0;
                        // immediate
                        if (t->operand[0] !=NULL && t->operand[0][0] =='#') {
                                target = (int)strtol(t->operand[0] +1, NULL, 16); // immediate 값
가져오기
                                unsigned int tmp = (opcode <<16);
                                tmp |= (t->nixbpe) <<12;
                                tmp |= (target &0x0FFF);        // 12비트 상대 주소
                                sprintf_s(obj, sizeof(obj), "%06X", tmp);
                        }
                        else {
                                // 타겟주소 불러오기
                                // literal
                                if (t->operand[0] !=NULL && t->operand[0][0] =='=') {
                                        for (int i =0; i < literal_num; i++) {
                                                if (strcmp(literal_table[i].symbol, t->operand[0])
==0) {
                                                        target = literal_table[i].addr;      //   리터
럴 주소 가져오기
                                                        break;
                                                }
                                        }
                                }
                                // indirect addressing
                                else if (t->operand[0] !=NULL && t->operand[0][0] =='@') {
                                        target = get_symval(start_symtab, t->operand[0] +1);
// 심볼 주소 가져오기
                                }
                                // 일반 명령어
                                else if (t->operand[0] !=NULL) {
                                        target = get_symval(start_symtab, t->operand[0]);
                                // 심볼 주소 가져오기
                                }
                                int idx = search_opcode(t->oper);
                                if (idx <0) continue;
        // opcode가 유효하지 않음
                                // 상대 주소 계산
                                //피연산자 0개 (ex) RSUB))
                                if (inst_table[idx]->ops ==0) {
                                        unsigned int tmp = (opcode <<16);
                                        tmp |= (t->nixbpe) <<12;
                                        sprintf_s(obj, sizeof(obj), "%06X", tmp);
                                }
                                // format 3
                                else if (format ==3) {
                                        disp = target - (t->addr +3);    // 상대 주소 계산
                                        unsigned int tmp = (opcode <<16);
                                        tmp |= (t->nixbpe) <<12;
                                        tmp |= (disp &0x0FFF); // 12비트 상대 주소
                                        sprintf_s(obj, sizeof(obj), "%06X", tmp);
                                }
                                // format 4
                                else if (format ==4) {
                                        disp = target - (t->addr +4);    // 상대 주소 계산
                                        unsigned int tmp = (opcode <<24);
                                        tmp |= (t->nixbpe) <<20;
```

```
                        tmp |=0x00000000;
                        sprintf_s(obj, sizeof(obj), "%08X", tmp);
                    }
                    // format 2
                    else if (format ==2) {
                        unsigned int tmp = opcode <<8;
                        if (t->operand[0] !=NULL) {
                            if (strcmp(t->operand[0], "A") ==0) {
                                tmp |=0x00;
                            }
                            else if (strcmp(t->operand[0], "X") ==0) {
                                tmp |=0x10;
                            }
                            else if (strcmp(t->operand[0], "L") ==0) {
                                tmp |=0x20;
                            }
                            else if (strcmp(t->operand[0], "B") ==0) {
                                tmp |=0x30;
                            }
                            else if (strcmp(t->operand[0], "S") ==0) {
                                tmp |=0x40;
                            }
                            else if (strcmp(t->operand[0], "T") ==0) {
                                tmp |=0x50;
                            }
                        }
                        if (t->operand[1] !=NULL) {
                            if (strcmp(t->operand[1], "A") ==0) {
                                tmp |=0x00;
                            }
                            else if (strcmp(t->operand[1], "X") ==0) {
                                tmp |=0x01;
                            }
                            else if (strcmp(t->operand[1], "L") ==0) {
                                tmp |=0x02;
                            }
                            else if (strcmp(t->operand[1], "B") ==0) {
                                tmp |=0x03;
                            }
                            else if (strcmp(t->operand[1], "S") ==0) {
                                tmp |=0x04;
                            }
                            else if (strcmp(t->operand[1], "T") ==0) {
                                tmp |=0x05;
                            }
                        }
                        sprintf_s(obj, sizeof(obj), "%04X", tmp);
                    }
                }
            }
            // 텍스트 레코드에 추가
            strcat_s(text_buffer, sizeof(text_buffer), obj);
            text_length += strlen(obj) /2;      // 텍스트 레코드 길이 업데이트
            //printf("%s₩t%s₩t%s₩t%s₩n", t->label ? t->label : "", t->oper ? t->oper : "",
t->operand[0] ? t->operand[0] : "", obj);

        }
        return 0;
```

```
}
/                                                                                *
-------------------------------------------------------------------------------
-----
* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
*        여기서 출력되는 내용은 object code이다.
* 매개 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 파일이 NULL값이 들어온다면 프로그램의 결과를 stdout으로 보내어
*        화면에 출력해준다.
*        명세서의 주어진 출력 결과와 완전히 동일해야 한다.
*        예외적으로 각 라인 뒤쪽의 공백 문자 혹은 개행 문자의 차이는 허용한다.
*
*
-------------------------------------------------------------------------------
------
*/
void make_objectcode_output(char* file_name)
{
        FILE* file;
        if (file_name ==NULL) {
                file = stdout;  // 표준 출력
        }
        else {
                fopen_s(&file, file_name, "w");
        }
        for (int i =0; i < obj_count; i++) {
                object_code* rec =&obj_codes[i];
                // 레코드 타입에 따라 구분 출력
                // H : 헤더 레코드
                // 프로그램 명은 object_code에 저장되어 있음
                if (rec->record_type =='H') {
                        fprintf(file,     "₩nH%06s%06X%06X₩n",rec->object_code,     rec->locctr,
rec->length);
                }
                else if (rec->record_type =='T') {
                        fprintf(file,       "T%06X%02X%s₩n",       rec->locctr,       rec->length,
rec->object_code);
                }
                else if (rec->record_type =='M') {
                        fprintf(file, "M%06X%s₩n", rec->locctr, rec->object_code);
                }
                else if (rec->record_type =='E') {
                        if (rec->locctr ==-1) {
                                fprintf(file, "E₩n");
                        }
                        else {
                                fprintf(file, "E%06X₩n", rec->locctr);
                        }
                }
                else if (rec->record_type =='D') {
                        fprintf(file, "D%s₩n", rec->object_code);
                }
                else if (rec->record_type =='R') {
                        fprintf(file, "R%s₩n", rec->object_code);
                }
                else {
                        fprintf(file, "ERROR: Unknown record type₩n");
```

```
                }
        }
        if (file != stdout)
                fclose(file);
}
/                                                                                *
--------------------------------------------------------------------------------
-----
* 설명 : label의 중복을 확인하고 label을 symtable에 추가하는 함수이다.
* 매개 : 테이블 순회 시작 idx, 추가할 label, 주소
* 반환 : 성공 : 0, 실패 : -1
*
--------------------------------------------------------------------------------
------
*/
int insert_symtab(int start_symtable,  const char* label, int addr) {
        for (int i = start_symtable; i < label_num ; i++) {
                if (strcmp(sym_table[i].symbol, label) ==0) {
                        printf("중복된 label이 존재합니다.₩n");
                        return -1;
                }
        }
        strcpy_s(sym_table[label_num].symbol, sizeof(sym_table[label_num].symbol), label);
        sym_table[label_num].addr = addr;
        label_num++;
        return 0;
}
/                                                                                *
--------------------------------------------------------------------------------
-----
* 설명 : literal의 중복을 확인하고 literal을 literal_table에 추가하는 함수이다.
* 매개 : 추가할 literal, 주소
* 반환 : 성공 : 0, 실패 : -1
*
--------------------------------------------------------------------------------
------
*/
int insert_literaltab(const char* label) {
        for (int i =0; i < literal_num; i++) {
                if (strcmp(literal_table[i].symbol, label) ==0) {
                        return -1;
                }
        }
        strcpy_s(literal_table[literal_num].symbol,sizeof(literal_table[literal_num].symbol), label);
        literal_table[literal_num].addr =-1;          // 주소는 아직 미정(-1)
        // literal 등장 시점 저장
        literal_line_index[literal_num] = token_line -1;  // 현재 줄 인덱스 (token_table 기준)
        literal_num++;
        return 0;
}
/                                                                                *
--------------------------------------------------------------------------------
-----
* 설명 : label에 대한 주소를 반환하는 함수이다.
* 매개 : 검색 시작 idx, 가져올 label
* 반환 : 성공 : 주소, 실패 :-1)
*
--------------------------------------------------------------------------------
```

```
------
*/
int get_symval(int start_symtable, const char* label) {
        if (start_symtable > label_num) {
                return -1;
        }
        for (int i = start_symtable; i < label_num; i++) {
                if (strcmp(sym_table[i].symbol, label) ==0) {
                        return sym_table[i].addr;
                }
        }
        return -1;
}
/                                                                                *
-------------------------------------------------------------------------------
-----
* 설명 : LTORG 명령어를 처리하기 위한 함수이다.
*
-------------------------------------------------------------------------------
------
*/
void assign_littab_addresses() {
        for (int i =0; i < literal_num; i++) {
                if (literal_table[i].addr ==-1) {
                        literal_table[i].addr = locctr;
                        if (literal_table[i].symbol[1] =='C') {
                                locctr += strlen(literal_table[i].symbol) -4;
                        }
                        else if (literal_table[i].symbol[1] =='X') {
                                locctr += (strlen(literal_table[i].symbol) -4) /2;
                        }
                }
        }
}
/                                                                                *
-------------------------------------------------------------------------------
-----
* 설명 : nixbpe를 설정
* 매개 : 토큰, 포맷
*
-------------------------------------------------------------------------------
------
*/
void set_nixbpe( token* t, int format) {
        t->nixbpe =0;
        if (format ==2) return;
        if (format ==4) {
                t->nixbpe |=0x01; // e = 1
        }
        // operand[0] 기준
        if (t->operand[0] !=NULL) {
                if (t->operand[0][0] =='#') {
                        t->nixbpe |= (1 <<4);  // i = 1
                }
                else if (t->operand[0][0] =='@') {
                        t->nixbpe |= (1 <<5);  // n = 1
                }
                else {
```

```
                        t->nixbpe |= (1 <<5);  // n = 1
                        t->nixbpe |= (1 <<4);  // i = 1
                }
        }
        else {
                // operand 없으면 기본으로 ni = 11
                t->nixbpe |= (1 <<5) | (1 <<4);
        }
        // x 사용
        if (t->operand[1] !=NULL && strcmp(t->operand[1], "X") ==0) {
                t->nixbpe |= (1 <<3);  // x = 1
        }

        // p, b 설정은 format 3/4일 때만 해당
                int target =0;
                int disp =0;
        if (t->operand[0] !=NULL && t->operand[0][0] !='#') {
                if (format ==3) {
                        disp = target - (t->addr +3);
                        if (disp >=-2048 && disp <2048) {
                                t->nixbpe |= (1 <<1);  // p = 1
                        }
                        else {
                                t->nixbpe |= (1 <<2);  // b = 1 (추후 base 처리 필요)
                        }
                }
        }

}
/                                                                                *
-------------------------------------------------------------------------------
-----
* 설명 : 마지막 할당된 주소를 찾는 함수이다. SECT가 종료후 주소를 찾기 위해 사용된다.
* 매개 : csect 시작 인덱스
* 반환 : 주소
*
-------------------------------------------------------------------------------
------
*/
int get_last_allocated_addr(int idx) {
        int sect_start =0;
        // idx가 몇번째 sect에 있는지 확인
        for (int i =1; i < csect_num; i++) {
                if (csect_start_token_line[i] >= idx) break;
                sect_start = csect_start_token_line[i];
        }
        int max_addr =-1;
        // 일반 명령어 기준 가장 마지막 주소
        for (int j = idx -1; j >= sect_start; j--) {
                token* t = token_table[j];
                if (!t || t->addr <0 ||!t->oper) continue;
                if (strcmp(t->oper, "EQU") ==0) continue;
                if (strcmp(t->oper, "RESB") ==0 && t->operand[0])
                        max_addr = t->addr + atoi(t->operand[0]);
                else if (strcmp(t->oper, "RESW") ==0 && t->operand[0])
                        max_addr = t->addr +3 * atoi(t->operand[0]);
                else if (strcmp(t->oper, "WORD") ==0)
                        max_addr = t->addr +3;
```

```
                else if (strcmp(t->oper, "BYTE") ==0) {
                        if (t->operand[0][0] =='C')
                                max_addr = t->addr + strlen(t->operand[0]) -3;
                        else if (t->operand[0][0] =='X')
                                max_addr = t->addr + (strlen(t->operand[0]) -3) /2;
                }
                else {
                        int inst_idx = search_opcode(t->oper);
                        if (inst_idx >=0) {
                                int format = inst_table[inst_idx]->format;
                                if (t->oper[0] =='+') format++;
                                max_addr = t->addr + format;
                        }
                }
                if (max_addr !=-1) break;  // 가장 가까운 거 하나만 찾으면 되니까
        }
        // 리터럴 처리 (해당 섹터 안에 있는 것만)
        for (int i =0; i < literal_num; i++) {
                if (literal_line_index[i] < sect_start || literal_line_index[i] >= idx) continue;
                if (literal_table[i].addr ==-1) continue;
                int len =0;
                if (literal_table[i].symbol[1] =='C')
                        len = strlen(literal_table[i].symbol) -4;
                else if (literal_table[i].symbol[1] =='X')
                        len = (strlen(literal_table[i].symbol) -4) /2;
                int lit_end = literal_table[i].addr + len;
                if (lit_end > max_addr) max_addr = lit_end;
        }
        return max_addr;
}
```

<my_assembler_20211397.h>

```
/*
* my_assembler 함수를 위한 변수 선언 및 매크로를 담고 있는 헤더 파일이다.
*
*/
#define MAX_INST 256
#define MAX_LINES 5000
#define MAX_COLUMNS 4
#define MAX_OPERAND 3
#define MAX_LINE_LENGTH 100
/*
* instruction 목록을 저장하는 구조체이다.
* instruction 목록 파일로부터 정보를 받아와서 생성한다.
* instruction 목록 파일에는 라인별로 하나의 instruction을 저장한다.
*
*/
typedef struct _inst
{
        char str[10];                   // 명령어 이름
        unsigned char op;               // 명령어 코드
        int format;                             // 명령어 형식
        int ops;                        // 피연산자 갯수
} inst;
// 기계어를 관리하는 테이블
inst* inst_table[MAX_INST];
int inst_index;
// 어셈블리 할 소스코드를 파일로부터 불러와 라인별로 관리하는 테이블
char* input_data[MAX_LINES];
static int line_num;
int label_num;
int literal_num;
/*
* 어셈블리 할 소스코드를 토큰으로 변환하여 저장하는 구조체 변수이다.
* operator 변수명은 renaming을 허용한다.
*/
typedef struct _token {
        char* label;
        char* oper;                     //operator
        char* operand[MAX_OPERAND];
        char comment[100];
        char nixbpe;    // 하위 6비트 사용 _ _ n i x b p e
        int addr;       // 주소
} token;
// 어셈블리 할 소스코드를 5000라인까지 관리하는 테이블
token* token_table[MAX_LINES];
static int token_line;
/*
* 심볼을 관리하는 구조체이다.
* 심볼 테이블은 심볼 이름, 심볼의 위치로 구성된다.
*/
typedef struct _symbol {
        char symbol[10];
        int addr;
} symbol;
/*
* 리터럴을 관리하는 구조체이다.
* 리터럴 테이블은 리터럴의 이름, 리터럴의 위치로 구성된다.
*/
typedef struct _literal {
```

```
        char* literal;
        int addr;
} literal;
symbol sym_table[MAX_LINES];
symbol literal_table[MAX_LINES];
/**
* 오브젝트 코드 전체에 대한 정보를 담는 구조체이다.
* Header Record, Define Recode,
* Modification Record 등에 대한 정보를 모두 포함하고 있어야 한다. 이
* 구조체 변수 하나만으로 object code를 충분히 작성할 수 있도록 구조체를 직접
* 정의해야 한다.
*/
typedef struct _object_code {
        char record_type;       //'H', 'T', 'M', 'E'
        int locctr;             // 주소
        int length;             // 길이
        char object_code[100]; // 오브젝트 코드
} object_code;
object_code obj_codes[MAX_LINES];
static int obj_count =0; // 오브젝트 코드 개수
static int start_addr =0; // 프로그램 시작 주소
static int locctr;              // 현재 주소
static int prog_length =0;      // 프로그램 길이
static int csect_start_symbol_num[MAX_LINES];   // CSECT 시작 심볼 번호
static int csect_num =0;// CSECT 개수
static int literal_line_index[100]; //csect별 리터럴 나누기용
static int csect_start_token_line[10]; // 섹터별 시작 토큰 인덱스
//---------------
static char* input_file;
static char* output_file;
int init_my_assembler(void);
int init_inst_file(char* inst_file);
int init_input_file(char* input_file);
int token_parsing(char*str);
int search_opcode(char*str);
static int assem_pass1(void);
void make_opcode_output(char* file_name);
void make_symtab_output(char* file_name);
void make_literaltab_output(char* filename);
static int assem_pass2(void);
void make_objectcode_output(char* file_name);
// 추가 함수
int insert_symtab(int start_symtable, const char* label, int addr);
int insert_literaltab(const char* label);
int get_symval(int start_symtable, const char* label);
void assign_littab_addresses();
void set_nixbpe(token* t, int format);
int get_last_allocated_addr(int idx);
```

첨부2) 디버거 사용 확인

11번 라인에 “ DIVR k,l”입력시 oper=DIVR, operand[0]=k, operand[1]=l이 저장됨을 확인할 수 있다. label = NULL이다.