설계(프로젝트) 보고서

1. 나는 자력으로 보고서를 작성하였습니다.

2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다.

3. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

교과목	시스템프로그래밍
프로젝트 명	SIC/XE 머신 어셈블러
교과목 교수	조 용 윤
제출인	학부: 정보통신공학과 학번: 20194318 성명: 박상진
조원(해당시 작성)	-

차 례

1장 프로젝트 개요

1.1 개발 배경 및 목적과 구현 범위

SIC/XE Assembler를 구현함으로서 컴퓨터 아키텍쳐 및 시스템 프로그램의 개념을 이해하고 더 나아가 어셈블러를 직접 개발함으로서 하드웨어와 소프트웨어 간 상호작용에 대한 개념도 이해한다.
어셈블러를 구현함으로서 코드를 생성하고 최적화하는 과정을 연구하고 연습하여 프로그래밍 역량을 강화한다. 또한, Low Level 프로그래밍에 대한 기술을 향상시키고 복잡한 시스템을 다루는 능력을 키운다. SIC/XE Assembler를 통해 어셈블리 언어의 동작 원리와 문법에 대한 이해를 목적으로 더 나아가 복잡한 시스템에서 어셈블리 코드를 작성하고 이해하는 능력을 기른다. C언어로 본 프로젝트를 수행하며 C 라이브러리에 대한 활용과 C 프로그래밍 역량을 강화한다.
본 프로젝트에서 구현한 SIC/XE 어셈블러는 SIC/XE 기본 Instruction을 처리하고 기호 정의문, Literal 그리고 Control Section에 대한 기능도 처리할 수 있다.

본 SIC/XE Assembler는 다음과 같은 입력을 처리할 수 있다.

```
COPY    START   0
        EXTDEF BUFFER,BUFEND,LENGTH
        EXTREF RDREC,WRREC
FIRST   STL     RETADR
CLOOP           +JSUB   RDREC
        LDA     LENGTH
        COMP    #0
        JEQ     ENDFIL
        +JSUB   WRREC
        J       CLOOP
ENDFIL  LDA     =C'EOF'
        STA     BUFFER
        LDA     #3
        STA     LENGTH
        +JSUB   WRREC
        J       @RETADR
RETADR RESW     1
LENGTHRESW      1
        LTORG
BUFFER RESB     4096
BUFEND EQU      *
MAXLENEQU       BUFEND-BUFFER
RDREC   CSECT
.
.       SUBROUTINE TO READ RECORD INTO BUFFER
.
        EXTREF BUFFER,LENGTH,BUFEND
        CLEAR   X
        CLEAR   A
        CLEAR   S
        LDT     MAXLEN
RLOOP   TD      INPUT
```

```
        JEQ     RLOOP
        RD      INPUT
        COMPR   A,S
        JEQ     EXIT
        +STCH   BUFFER,X
        TIXR    T
        JLT     RLOOP
EXIT    +STX    LENGTH
        RSUB
INPUT   BYTE    X'F1'
MAXLENWORD      BUFEND-BUFFER
WRREC   CSECT
.
.       SUBROUTINE TO WRITE RECORD FROM BUFFER
.
        EXTREF LENGTH,BUFFER
        CLEAR   X
        +LDT    LENGTH
WLOOP   TD      =X'05'
        JEQ     WLOOP
        +LDCH   BUFFER,X
        WD      =X'05'
        TIXR    T
        JLT     WLOOP
        RSUB
        END     FIRST
```

어셈블러는 2 Pass로 동작한다.
초기화 과정에서 "input.txt"와 "inst.data" 읽어 각 테이블에 저장하고 "optab.txt"을 출력한다.
Pass 1에서는 locctr를 갱신하고 Symbol Table과 Literal Table, 외부참조 테이블에 각 심볼과 주소 등을 저장하여 "intermediate.txt", "symtab.txt"를 출력한다.
Pass 2에서는 오브젝트 코드를 생성하고 "list.txt"에 오브젝트 코드를 포함하여 출력한다. 또한 생성된 오브젝트 코드를 기반으로 "objectprogram.txt"을 출력한다.

"intermediate.txt"는 중간파일로 각 라인별 Locctr와 입력 원문을 포함한다.
"optab.txt"는 OP Table로 명령어 테이블에 저장된 Mnemonic, format, opcode를 포함한다.
"symtab.txt"는 Symbol Table로 입력 프로그램의 symbol과 literal을 정리하여 심볼, 주소, 섹션 또는 Pool Number(LT) 정보를 포함한다.
"list.txt"는 리스트 파일로 각 라인별 Locctr와 입력 원문 그리고 objectcode를 포함한다.
"objectprogram.txt"는 오브젝트 프로그램으로 Header, Text, End 그리고 Modification Record를 포함한다.

2장 배경 지식

2.1 주제에 관한 배경지식

SIC/XE는 IBM에서 개발한 컴퓨터 아키텍처로, 기본적으로 SIC(Simplified Instructional Computer) 아키텍처에서 확장된 형태이다.

SIC/XE 아키텍처는 기본 명령어에 새로운 명령어와 레지스터를 추가하여 성능을 향상시킨 것으로, 더 복잡하고 다양한 프로그램을 지원한다. 이 아키텍처의 특징 중 하나는 확장 기호(Extended Mnemonics)를 사용하는 것이다. 이는 더 직관적이고 표현력이 뛰어난 어셈블리 코드를 작성할 수 있게 한다.

기호 정의문은 어셈블리 언어에서 사용되는 기호(Symbol)에 대한 정보를 정의하는 문장이다. 각각의 기호는 메모리 주소와 연결되어 있으며, 어셈블러는 이러한 주소 정보를 생성된 기계어 코드에 포함한다. 기호는 주로 레이블(label)로 표현되며, 어셈블리 코드 내에서 가독성을 높이고 주소를 추적하는 데 사용된다. 어셈블러는 기호 정의문을 통해 프로그램에서 사용되는 모든 기호를 인식하고, 해당 기호에 대한 주소를 할당한다. 이는 프로그램이 실행될 때 정확한 메모리 위치에서 데이터나 명령어를 참조할 수 있도록 보장한다. 또한 기호 정의문은 사용자가 주소를 직접 할당하는 것이 아니라, 상대적인 주소나 상수를 사용하여 코드를 더 유연하게 작성할 수 있도록 한다. 기호 정의문은 어셈블러가 프로그램의 기호와 주소를 관리하고, 이를 통해 프로그램이 효과적으로 실행될 수 있도록 보장하는 역할을 한다.

Literal은 프로그램에서 사용되는 상수 값이나 문자열을 나타내는데, 이는 어셈블리 코드에서 고정된 값을 나타내는 역할을 한다. 리터럴은 주로 명령어나 데이터의 피연산자로 사용되며, 프로그램 실행 중에 메모리에 할당된다. 어셈블러는 프로그램에서 사용된 모든 리터럴을 식별하고, 이들에 대한 주소를 할당하는 역할을 수행한다. 이는 프로그램이 실행될 때 정확한 값이 사용되도록 보장하며, 어셈블리 코드의 가독성을 높인다. 또한 어셈블러는 리터럴을 효율적으로 관리하기 위해 중복된 리터럴을 공유하는 기능을 제공한다. 이는 프로그램의 메모리 사용을 최적화하고 실행 속도를 향상시킨다.

Control Section은 프로그램의 실행을 제어하는 섹션으로, 프로그램의 흐름을 나타내고 제어한다. 각 Control Section은 프로그램의 한 부분을 나타내며, 프로그램이 여러 모듈로 구성되어 있을 때 각 모듈은 각각의 Control Section으로 나뉜다. 어셈블러는 Control Section을 인식하고, 프로그램의 시작 및 종료 지점을 식별하여 프로그램이 어떻게 실행될지를 결정한다. 또한 다른 Control Section 간의 연결 및 데이터 전달을 관리하여 프로그램이 목적에 맞게 동작하도록 한다. Control Section은 프로그램의 구조를 정의하고, 각 섹션 간의 통합을 가능하게 하며, 프로그램의 유지보수성을 향상시킨다.

2.2 기술적 배경지식

어셈블러는 정해진 자연어 형태인 어셈블리어로 작성된 프로그램을 기계어 형태의 오브젝트 프로그램으로 번역하는 번역기이다. SIC/XE는 할당, 정의, 선언, 연산, 호출 등 응용 프로그램의 기본적인 기능과 동작들을 가능하게 한다.

본인은 SIC/XE 아키텍쳐를 가장 친숙한 프로그래밍 언어인 C의 개념에 비교하여 이해하였다.

WORD, BYTE와 같은 Directive는 변수 선언, RESW, RESB는 malloc()

START와 END는 main() 함수의 ‘{’, ‘}’ 함수의 시작과 끝

Literal은 문자, 문자열을 처리를 가능하게 하는 char와 string.h 라이브러리

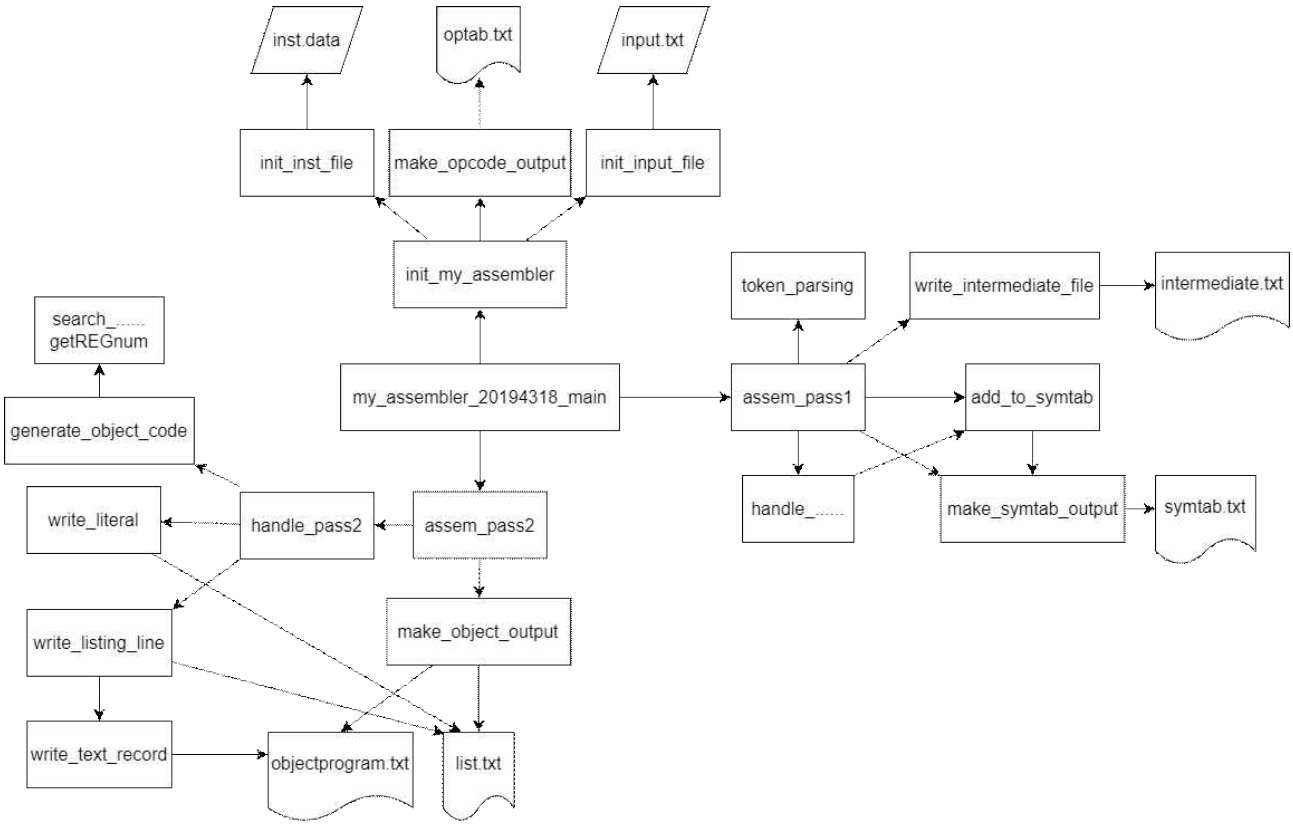
기호정의문은 #define 또는 typedef 등과 같은 매크로

Control Section은 프로그램을 함수로 모듈화하는 refactor 과정

이와 같이 C라이브러리에 SIC/XE를 빗대어 이해하고 프로젝트를 진행하였다. 물론 SIC/XE와 C의 두 개념이 완전히 일치하지는 않지만, 하드웨어와 소프트웨어 간의 상호작용을 가능하게 한다는 공통점을 중심으로 SIC/XE의 기능과 원리를 이해하였고 이는 어셈블러를 직접 구현하는 데 도움이 되었다.

3장 시스템 설계 내용

3.1 전체 시스템 설계 내용



3.2 모듈별 설계 내용

int init_my_assembler(void);

: 어셈블러 초기화, init_inst_file(), init_input_file() 포함

int init_inst_file(uchar *inst_file);

: inst.data를 읽고 inst_table[]에 저장

int init_input_file(uchar *input_file);

: input.txt를 읽고 input_data[]에 저장

int token_parsing(uchar *str);

: input_data[]의 문장을 토큰화하여 token_table[]에 저장

int search_opcode(uchar *str);

: inst_table[]에서 opcode를 검색하고 opcode의 인덱스를 반환

static int assem_pass1(void);

: Pass 1 과정 처리

void make_opcode_output(uchar *file_name);

: OP 테이블 파일 출력

void make_symtab_output(uchar *file_name);

: Symbol 테이블 파일 출력

static int assem_pass2(void);

: Pass 2 과정 처리

void make_objectcode_output(uchar *file_name, uchar *list_name);

: 오브젝트 프로그램과 리스트 파일 출력

void write_intermediate_file(uchar *str, int locctr);

: 중간 파일을 생성하고 작성

void add_to_symtab(const uchar *label, int loc, int is_equ, int sec);

: sym_table[]에 symbol을 추가

int search_symtab(uchar *symbol, int section);

: sym_table[]에서 symbol을 검색하고 그 주소를 반환

int init_token_table(void);

: token_table[] 초기화

int evaluate_expression(uchar *expr);

: 수식을 처리 ex) BUFEND-BUFFER 수식의 결과를 반환

int search_literal(uchar *operand);

: LTtab[]에서 LT를 검색하고 그 주소를 반환

int calculate_byte_length(uchar *operand);

: 피연산자의 바이트 길이를 계산 그 길이를 반환

int search_extRtab(uchar *symbol, int section);

: extRef[]에서 현재 섹션의 symbol을 검색 그 주소를 반환

int search_extDtab(uchar *symbol);

: extDef[]에서 심볼을 검색하고 그 주소를 반환

void handle_extdef(uchar *symbol);

: EXTDEF를 처리: extDef[]에 심볼과 주소를 추가

void handle_extref(uchar *symbol, int section);

: EXTREF를 처리: extRef[]에 심볼과 섹션 그리고 주소를 저장

void handle_equ_directive(uchar *label, uchar *operand);

: EQU를 처리: locctr를 갱신하고 evaluate_expression()를 포함

void handle_ltorg_directive(void);
: LTORG를 처리: 지난 세션의 LT를 LTtab[]에 저장하고, locctr 갱신
int hexstr2dec(char H);
: 16진수 문자열을 10진수로 변환
int getREGnum(uchar *register_name);
: 레지스터 번호를 반환
int generate_object_code(int format);
: 명령어 포맷과 피연산자에 따라서 오브젝트 코드를 생성
int write_listing_line(int format);
: 행번호, locctr, 입력문, 오브젝트 코드를 명령어 포맷과 리스트 파일에 작성
void write_text_record(void);
: 오브젝트 프로그램에 T레코드 작성
int write_literal(void);
: 리스트 파일에 리터럴 작성
int handle_pass2(void);
: 디렉티브에 따라 Pass 2를 처리: 레코드와 리스트를 출력

4장 시스템 구현 내용

4.1 전체 시스템 구현 내용

optab.txt

Mnemonic	MachineCode	Format
ADD	18	3
ADDF	58	3
ADDR	90	2
AND	40	3
BYTE	0B	0
CLEAR	B4	2
COMP	28	3
COMPF	88	3
COMPR	A0	2
CSECT	0A	0
DIV	24	3
DIVF	64	3
DIVR	9C	2
END	0E	0
EQU	09	0
EXTDEF	02	0
EXTREF	03	0
FIX	C4	1
FLOAT	C0	1
HIO	F4	1
J	3C	3
JEQ	30	3
JGT	34	3
JLT	38	3
JSUB	48	3
LDA	00	3
LDB	68	3
LDCH	50	3
LDF	70	3
LDL	08	3
LDS	6C	3
LDT	74	3
LDX	04	3
LPS	D0	3
LTORG	07	0
MUL	20	3
MULF	60	3
MULR	98	2
NORM	C8	1
OR	44	3
RD	D8	3
RESB	06	0
RESW	05	0
RMO	AC	2
RSUB	4C	3
SHIFTL	A4	2
SHIFTR	A8	2
SIO	F0	1
SSK	EC	3
STA	0C	3
START	01	0
STB	78	3
STCH	54	3
STF	80	3
STI	D4	3
STL	14	3
STS	7C	3
STSW	E8	3
STT	84	3
STX	10	3
SUB	1C	3
SUBF	5C	3
SUBR	94	2
SVC	B0	2
TD	E0	3
TIO	F8	1
TIX	2C	3
TIXR	B8	2
WD	DC	3
WORD	0D	0

symtab.txt

Symbol	Address	Section
FIRST	0000	0
CLOOP	0003	0
ENDFIL	0017	0
RETADR	002A	0
LENGTH	002D	0
BUFFER	0033	0
BUFEND	1033	0
MAXLEN	1000	0
RDREC	0000	1
RLOOP	0009	1
EXIT	0020	1
INPUT	0027	1
MAXLEN	0028	1
WRREC	0000	2
WLOOP	0006	2

Literal	Address	PoolNum
=C'EOF'	0030	0
=X'05'	001B	1

intermediate.txt

```
0000 COPY        START   0
0000             EXTDEF  BUFFER,BUFEND,LENGTH
0000             EXTREF  RDREC,WRREC
0000 FIRST       STL     RETADR
0003 CLOOP       +JSUB   RDREC
0007             LDA     LENGTH
000A             COMP    #0
000D             JEQ     ENDFIL
0010             +JSUB   WRREC
0014             J       CLOOP
0017 ENDFIL      LDA     =C'EOF'
001A             STA     BUFFER
001D             LDA     #3
0020             STA     LENGTH
0023             +JSUB   WRREC
0027             J       @RETADR
002A RETADR      RESW    1
002D LENGTH      RESW    1
0030             LTORG
0030             =C'EOF'
0033 BUFFER      RESB    4096
1033 BUFEND      EQU     *
1000 MAXLEN      EQU     BUFEND-BUFFER
0000 RDREC       CSECT
0000             EXTREF  BUFFER,LENGTH,BUFEND
0000             CLEAR   X
0002             CLEAR   A
0004             CLEAR   S
0006             LDT     MAXLEN
0009 RLOOP       TD      INPUT
000C             JEQ     RLOOP
000F             RD      INPUT
0012             COMPR   A,S
0014             JEQ     EXIT
0017             +STCH   BUFFER,X
001B             TIXR    T
001D             JLT     RLOOP
0020 EXIT        +STX    LENGTH
0024             RSUB
0027 INPUT       BYTE    X'F1'
0028 MAXLEN      WORD    BUFEND-BUFFER
0000 WRREC       CSECT
0000             EXTREF  LENGTH,BUFFER
0000             CLEAR   X
0002             +LDT    LENGTH
0006 WLOOP       TD      =X'05'
0009             JEQ     WLOOP
000C             +LDCH   BUFFER,X
0010             WD      =X'05'
0013             TIXR    T
0015             JLT     WLOOP
0018             RSUB
001B             END     FIRST
001B             =X'05'
```

list.txt

```
  5    0000  COPY   START   0
 10                EXTDEF  BUFFER,BUFEND,LENGTH
 15                EXTREF  RDREC,WRREC
 20    0000  FIRST  STL     RETADR      172027
 25    0003  CLOOP         +JSUB       RDREC       4B100000
 30    0007         LDA     LENGTH      032023
 35    000A         COMP    #0                      290000
 40    000D         JEQ     ENDFIL      332007
 45    0010         +JSUB   WRREC       4B100000
 50    0014         J       CLOOP       3F2FEC
 55    0017  ENDFIL LDA     =C'EOF'     032016
 60    001A         STA     BUFFER   0F2016
 65    001D         LDA     #3          010003
 70    0020         STA     LENGTH   0F200A
 75    0023         +JSUB   WRREC       4B100000
 80    0027         J       @RETADR     3E2000
 85    002A  RETADR         RESW    1
 90    002D  LENGTH         RESW    1
 95                LTORG
100    0030  *    =C'EOF'   454F46
105    0033  BUFFER         RESB    4096
110    1033  BUFEND         EQU     *
115    1000  MAXLEN         EQU     BUFEND-BUFFER

120    0000  RDREC  CSECT
125                EXTREF  BUFFER,LENGTH,BUFEND
130    0000         CLEAR   X           B410
135    0002         CLEAR   A           B400
140    0004         CLEAR   S           B440
145    0006         LDT     MAXLEN   77201F
150    0009  RLOOP  TD      INPUT       E3201B
155    000C         JEQ     RLOOP       332FFA
160    000F         RD      INPUT       DB2015
165    0012         COMPR   A,S         A004
170    0014         JEQ     EXIT        332009
175    0017         +STCH   BUFFER,X    57900000
180    001B         TIXR    T           B850
185    001D         JLT     RLOOP       3B2FE9
190    0020  EXIT   +STX     LENGTH      13100000
195    0024         RSUB                4F0000
200    0027  INPUT  BYTE    X'F1'       F1
205    0028  MAXLEN         WORD        BUFEND-BUFFER   000000

210    0000  WRREC  CSECT
215                EXTREF  LENGTH,BUFFER
220    0000         CLEAR   X           B410
225    0002         +LDT    LENGTH  77100000
230    0006  WLOOP  TD      =X'05'      E32012
235    0009         JEQ     WLOOP       332FFA
240    000C         +LDCH   BUFFER,X    53900000
245    0010         WD      =X'05'      DF2008
250    0013         TIXR    T           B850
255    0015         JLT     WLOOP       3B2FEE
260    0018         RSUB                4F0000
265                END     FIRST
270    001B  *    =X'05'  05
```

objectprogram.txt

```
HCOPY   000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDREC  WRREC
T0000001D1720274B10000003202329000033200074B1000003F2FEC0320160F2016
T00001D0D0100030F200A4B1000003E2000
T00003003454F46
M00000405+ RDREC
M00001105+ WRREC
M00002405+ WRREC
E000000

HRDREC  00000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A00433200957900000B850
T0000000E3B2FE9131000004F0000F1000000
M00001805+ BUFFER
M00002105+ LENGTH
M00002806+ BUFEND
M00002806-BUFFER
E

HWRREC  00000000001C
RLENGTHBUFFER
T0000001BB41077100000E32012332FFA53900000DF2008B8503B2FEE4F0000
T00001B0105
M00000305+ LENGTH
M00000D05+ BUFFER
E
```

4.2 모듈별 구현 내용

```
// 심볼 테이블에 심볼을 추가하는 함수
void add_to_symtab(const uchar *label, int loc, int is_equ, int sec) {
    // 중복된 심볼인지 확인
    for (int i = 0; i < sym_index; i++) {
        if (strcmp(sym_table[i].symbol, label) == 0 && sym_table[i].sec == sec) {
            if (is_equ) {
                fprintf(stderr, "오류: 중복된 심볼 발견 - %s₩n", label);
                // 필요한 대로 오류 처리를 수행
                return;
            }

            return;
        }
    }
    // 심볼 테이블에서 빈 슬롯을 찾음
    for (int index = 0; index < MAX_LINES; index++) {
        if (sym_table[index].symbol[0] == '₩0') {
            break;  // 빈 슬롯을 찾음
        }
    }
    // 심볼 테이블이 가득 찼는지 확인
    if (sym_index == MAX_LINES) {
        fprintf(stderr, "오류: 심볼 테이블이 가득 찼습니다.₩n");
        // 필요한 대로 오류 처리를 수행
        return;
    }
    // 심볼을 심볼 테이블에 추가
    strcpy(sym_table[sym_index].symbol, label);
    sym_table[sym_index].addr = loc;
    sym_table[sym_index].sec = sec;
    printf("심볼을 심볼 테이블에 추가 중: %s, 섹션 %d, 주소 %04X₩n", label, sec, loc);
    sym_index++;
}

// 리터럴 테이블에서 리터럴을 검색하는 함수
int search_literal(uchar *operand) {
    for (int i = 0; i < LT_num; ++i) {
        if (strcmp(LTtab[i].name, operand) == 0) {
            // 리터럴 찾음, 리터럴 인덱스 반환
            return i;
        }
    }

    // 리터럴을 찾지 못함
    return -1;
}
```

```
// BYTE 또는 WORD 피연산자의 길이를 계산하는 함수
int calculate_byte_length(uchar *operand) {
    // '='로 시작하는 경우 리터럴 처리
    if (operand[0] == '=') {
        // 'C' 또는 'X'로 시작하는 경우
        if (operand[1] == 'C' || operand[1] == 'c' || operand[1] == 'X' || operand[1] == 'x') {
            // 작은 따옴표 사이의 문자 수를 센다
            int length = 0;
            for (int i = 3; operand[i] != '₩''; ++i) {
                length++;
            }
            if (operand[1] == 'X' || operand[1] == 'x') {
                // 16진수 형식이므로 길이를 2로 나눈다
                return (length + 1) / 2;
            } else {
                return length;
            }
        } else {
            // 리터럴 테이블에서 길이를 찾아 반환
            int literal_index = search_literal(operand);
            if (literal_index != -1) {
                return LTtab[literal_index].leng;
            } else {
                // 리터럴이 정의되지 않았으므로 에러 처리
                fprintf(stderr, "Error: Undefined literal - %s₩n", operand);
                return -1;
            }
        }
    }
    // 'C' 또는 'X'로 시작하는 경우
    if (operand[0] == 'C' || operand[0] == 'c' || operand[0] == 'X' || operand[0] == 'x') {
        // 작은 따옴표 사이의 문자 수를 센다
        int length = 0;
        for (int i = 2; operand[i] != '₩''; ++i) {
            length++;
        }
        if (operand[0] == 'X' || operand[0] == 'x') {
            // 16진수 형식이므로 길이를 2로 나눈다
            return (length + 1) / 2;
        } else {
            return length;
        }
    }
}

// 심볼 테이블에서 심볼 검색하는 함수
int search_symtab(uchar *symbol, int section) {
    for (int i = 0; i < MAX_LINES; ++i) {
        // 심볼과 섹션이 테이블 항목과 일치하는지 확인
        if ((strcmp(sym_table[i].symbol, symbol) == 0) && (sym_table[i].sec == section)) {
```

```
            // 심볼을 찾으면 주소를 반환
            return sym_table[i].addr;
        }
    }
    // 해당 섹션의 심볼 테이블에서 심볼을 찾지 못한 경우
    return -1;
}

// 외부 참조 테이블에서 외부 참조 검색하는 함수
int search_extRtab(uchar *symbol, int section) {
    for (int i = 0; i < MAX_EXTREF; ++i) {
        // 심볼과 섹션이 외부 참조 항목과 일치하는지 확인
        if ((strcmp(extRef[i].symbol, symbol) == 0) && (extRef[i].sec == section)) {
            // 외부 참조를 찾으면 주소를 반환
            return extRef[i].addr;
        }
    }
    // 외부 참조를 찾지 못한 경우 에러를 보고
    printf("Error: Undefined external reference - %s₩n", symbol);
    return -1;
}

// 어셈블러를 초기화하는 함수로, 명령어 데이터와 소스 코드를 로드한다
int init_my_assembler(void) {
    // 명령어 데이터 초기화
    if (init_inst_file("inst.data") == -1) {
        fprintf(stderr, "Error: Instruction Initiation Failed₩n");
        return -1;
    }

    // 소스 코드 파일("input.txt") 초기화
    if (init_input_file("input.txt") == -1) {
        fprintf(stderr, "Error: Source Code Initiation Failed₩n");
        return -1;
    }

    // op코드 출력 파일("optab.txt") 생성
    make_opcode_output("optab.txt");

    // 토큰 테이블 초기화
    init_token_table();

    return 0;
}

// 파일에서 명령어 데이터를 초기화하는 함수
int init_inst_file(uchar *inst_file) {
    FILE *file = fopen(inst_file, "r");

    if (file == NULL) {
```

```
        perror("Error opening inst.data file");
        return -1;
    }

    char line[100];  // inst.data 파일의 각 행이 100자를 초과하지 않을 것으로 가정

    while (fgets(line, sizeof(line), file) != NULL) {
        if (inst_index >= MAX_INST) {
            fprintf(stderr, "Error: Maximum number of instructions reached.₩n");
            fclose(file);
            return -1;
        }

        inst_table[inst_index] = (inst *)malloc(sizeof(inst));

        // inst.data 파일의 각 행이 "str ops format op" 형식으로 가정
        sscanf(line, "%s %d %02X", inst_table[inst_index]->str, &(inst_table[inst_index]->format),
&(inst_table[inst_index]->op));

        // 테스트 출력문
        printf("Instruction %d: Mnemonic %s Format %d OP %02X₩n", inst_index + 1,
inst_table[inst_index]->str, inst_table[inst_index]->format, inst_table[inst_index]->op);

        inst_index++;
    }
    inst_count = inst_index;
    fclose(file);
    return 0;
}

// 파일에서 소스 코드를 초기화하는 함수
int init_input_file(uchar *input_file) {
    FILE *input_fp = fopen(input_file, "r");
    if (input_fp == NULL) {
        fprintf(stderr, "Error opening input file: %s₩n", input_file);
        return -1;
    }

    char line[100];
    int line_index = 0;

    while (fgets(line, sizeof(line), input_fp) != NULL) {
        // 줄 끝에 있는 개행 문자를 제거
        line[strcspn(line, "₩n")] = '₩0';

        // 줄이 점으로 시작하지 않으면(주석이 아니면)
        if (line[0] != '.') {
            // input_data 배열에 입력 저장
            input_data[line_index] = strdup(line);
```

```
            // 다음 줄로 이동
            line_index++;
        }
    }

    // 전체 줄 수 저장
    line_num = line_index;

    printf("Input Data:₩n");
    for (int i = 0; i < line_num; i++) {
        printf("%d: %s₩n", i + 1, input_data[i]);
    }
    printf("Total Lines: %d₩n", line_num);

    fclose(input_fp);
    return 0;
}

// 토큰 테이블을 초기화하는 함수
int init_token_table(void) {
    // 토큰 테이블의 각 요소를 초기화
    for (int i = 0; i < MAX_LINES; i++) {
        token_table[i] = malloc(sizeof(token));
        if (token_table[i] == NULL) {
            fprintf(stderr, "Error: Memory allocation failed.₩n");
            return -1;
        }
        // 필드를 기본 값으로 초기화
        token_table[i]->label = NULL;
        token_table[i]->operator = NULL;
        for (int j = 0; j < MAX_OPERAND; j++) {
            token_table[i]->operand[j][0] = '₩0';
        }
        token_table[i]->comment[0] = '₩0';
    }
    return 0;
}

// 코드 한 줄을 토큰으로 파싱하고 해당 토큰을 토큰 테이블에 저장하는 함수
int token_parsing(uchar *str) {
    // 토큰 엔트리에 대한 메모리 초기화
    token_table[token_line] = malloc(sizeof(token));
    if (token_table[token_line] == NULL) {
        fprintf(stderr, "Error: Memory allocation failed.₩n");
        return -1;
    }

    // 토큰화를 위해 입력 문자열의 복제 생성
    uchar *scopy = strdup((uchar *)str);
```

```
// 공백이나 탭 문자를 사용하여 토큰화
uchar *token_str = strtok(scopy, " ₩t");

// 첫 번째 토큰이 연산자인지 확인
int is_operator = search_opcode(token_str);

if (is_operator >= 0) {
    // 첫 번째 토큰이 연산자인 경우
    token_table[token_line]->label = NULL;
    token_table[token_line]->operator = strdup(token_str);
    token_str = strtok(NULL, " ₩t"); // 연산자 파싱 생략
} else {
    // 첫 번째 토큰이 레이블인 경우
    token_table[token_line]->label = strdup(token_str);
    token_str = strtok(NULL, " ₩t₩n"); // 다음 토큰으로 이동

    // 연산자 파싱
    if (token_str != NULL) {
        token_table[token_line]->operator = strdup(token_str);
        token_str = strtok(NULL, " ₩t₩n"); // 다음 토큰으로 이동
    } else {
        fprintf(stderr, "Error: Operator missing.₩n");
        return -1;
    }
}

// 피연산자 파싱
int operand_index = 0;
while (token_str != NULL) {
    if (operand_index < MAX_OPERAND) {
        // 쉼표로 구분된 여러 피연산자가 있는지 확인
        if (strchr(token_str, ',')) {
            char *operand_token = strtok(token_str, ",");
            while (operand_token != NULL) {
                strcpy((char *)token_table[token_line]->operand[operand_index], operand_token);
                operand_token = strtok(NULL, ",₩t");
                operand_index++;
            }
        } else {
            // 피연산자가 없는 경우, 피연산자 배열의 첫 번째 요소에 '₩0' 설정
            if (token_str != NULL) {
                strcpy((char *)token_table[token_line]->operand[operand_index], token_str);
                operand_index++;
            } else {
                token_table[token_line]->operand[operand_index][0] = '₩0';
            }
            token_str = strtok(NULL, "₩t");
            break;
        }
    } else {
```

```
            fprintf(stderr, "Error: Too many operands.₩n");
            free(scopy);
            return -1;
        }
        token_str = strtok(NULL, " ₩t₩n"); // 다음 토큰으로 이동
    }

    // 토큰화에 사용된 문자열 복제 해제
    free(scopy);

    // 테스트 출력
    printf("Line %d₩n", token_line + 1);
    printf("Label: %s₩n", token_table[token_line]->label);
    printf("Operator: %s₩n", token_table[token_line]->operator);
    printf("Operands: ₩n");
    for (int i = 0; i < operand_index; i++) {
        printf("  %s₩n", token_table[token_line]->operand[i]);
    }

    return 0;
}

// 주어진 문자열이 명령어 테이블에서 어떤 연산자와 일치하는지 검색하는 함수
int search_opcode(uchar *str) {
    for (int i = 0; i < inst_count; i++) {
        // 주어진 문자열이 연산자 니모닉과 일치하는지 확인 (혹은 '+' 접두사와 함께)
        if (strcmp(str, inst_table[i]->str) == 0 || (str[0] == '+' && strcmp(str + 1, inst_table[i]->str) ==
0)) {
            // 터미널에 검색된 연산자 정보를 출력하여 테스트
            printf("Operator: %s / ", inst_table[i]->str);
            printf("Format: %d / ", inst_table[i]->format);
            printf("Opcode Value: 0x%02X₩n", inst_table[i]->op);
            return i; // 명령어 테이블에서 찾은 연산자의 인덱스를 반환
        }
    }

    // 명령어를 찾을 수 없을 때 오류 보고
    fprintf(stderr, "Error: Opcode not found for %s₩n", str);
    return -1;
}

// EXTDEF 지시문을 처리하는 함수
void handle_extdef(uchar *symbol) {
    // 추가 전에 중복 확인
    for (int i = 0; i < MAX_EXTDEF; i++) {
        if (strcmp(extDef[i].symbol, symbol) == 0) {
            fprintf(stderr, "Error: Duplicate EXTDEF found - %s₩n", symbol);
            // 필요에 따라 오류 처리
            return;
        }
```

```
    }
    // 심볼을 EXTDEF 테이블에 추가
    strcpy(extDef[extDefCount].symbol, symbol);
    extDef[extDefCount].addr = search_symtab(symbol, sec);
    extDefCount++;
}

// EXTREF 지시문을 처리하는 함수
void handle_extref(uchar *symbol, int section) {
    // 추가 전에 중복 확인
    for (int i = 0; i < MAX_EXTREF; i++) {
        if (strcmp(extRef[i].symbol, symbol) == 0 && extRef[i].sec == section) {
            fprintf(stderr, "Error: Duplicate EXTREF found - %s₩n", symbol);
            // 필요에 따라 오류 처리
            return;
        }
    }
    // 심볼을 EXTREF 테이블에 추가
    extRef[extRefCount].sec = section;
    extRef[extRefCount].addr = locctr;
    strcpy(extRef[extRefCount].symbol, symbol);
    extRefCount++;
}

// EQU 지시문을 처리하는 함수
void handle_equ_directive(uchar *label, uchar *operand) {
    int equ_value;

    // 오퍼랜드가 "*"인지 확인
    if (strcmp(operand, "*") == 0) {
        // 오퍼랜드가 "*", 위치 카운터(locctr)를 참조
        equ_value = locctr;
    } else {
        // 오퍼랜드가 식인 경우, 그 값을 계산
        equ_value = evaluate_expression(operand);
        locctr = equ_value; // 평가된 식으로 위치 카운터를 업데이트
    }

    // 심볼을 심볼 테이블에 추가 (is_equ 매개변수는 1로 설정)
    add_to_symtab(label, locctr, 1, sec);
}

// LTORG 지시문을 처리하는 함수
void handle_ltorg_directive(void) {
    int literal_length = 0;
    current_pool = -1;  // 현재 LTORG 풀을 추적

    // 리터럴 테이블을 반복
    for (int i = 0; i < LT_num; i++) {
        // 리터럴이 주소를 할당 받았는지 확인
```

```
            if (LTtab[i].addr == -1) {
                // 할당 받지 않았다면, 현재 LTORG 풀에서 다음 사용 가능한 주소를 할당
                if (current_pool == -1) {
                    // 새로운 LTORG 풀을 시작
                    current_pool = i;
                }

                LTtab[i].addr = locctr +  literal_length;
                literal_length += LTtab[i].leng;
                LTtab[i].value = current_pool;
                // 중간 파일에서 리터럴 주소를 업데이트
                write_intermediate_file(LTtab[i].name, LTtab[i].addr);
            } else {
                // 리터럴이 이미 주소를 할당받았다면, LTORG 풀을 재설정
                current_pool = -1;
            }
        }

        // 위치 카운터를 리터럴의 총 길이로 업데이트
        locctr = locctr +  literal_length;
    }

    // 어셈블러의 첫 번째 패스를 수행하는 함수
    static int assem_pass1(void) {
        // 다양한 변수 및 데이터 구조를 초기화
        locctr = 0;
        starting_address = 0;
        csect_start_address = 0;
        token_line = 0;
        sym_index = 0;
        sec = 0;
        extRefCount = 0;
        LT_num = 0;

        // 심볼 테이블 초기화
        for (int i = 0; i < MAX_LINES; ++i) {
            sym_table[i].sec = -1;
            sym_table[i].addr = -1;
            sym_table[i].symbol[0] = '₩0';
        }

        // 리터럴 테이블 초기화
        for (int i = 0; i < MAX_LITERALS; i++) {
            LTtab[i].name[0] = '₩0';
            LTtab[i].addr = -1;
            LTtab[i].leng = -1;
        }

        // 컨트롤 섹션 테이블 초기화
        for (int i = 0; i < MAX_CSECT; ++i) {
```

```
    csect_table[i].sec = -1;
    csect_table[i].program_length = 0;
}

// 외부 정의 및 참조 테이블 초기화
for (int i = 0; i < MAX_EXTDEF; i++) {
    extDef[i].addr = -1;
    extDef[i].sec = -1;
    extDef[i].symbol[0] = '₩0';
    extRef[i].addr = -1;
    extRef[i].sec = -1;
    extRef[i].symbol[0] = '₩0';
}

// 첫 번째 입력 라인을 읽고 토큰 분석 수행
uchar *current_line = input_data[0];
token_parsing(current_line);

// START 지시문 확인
if (strcmp(token_table[0]->operator, "START") == 0) {
    // #[OPERAND]를 시작 주소로 저장
    starting_address = strtol(token_table[0]->operand[0], NULL, 16);

    // LOCCTR를 시작 주소로 초기화
    locctr = starting_address;

    // 라인을 중간 파일에 작성
    token_table[token_line]->addr = locctr;
    write_intermediate_file(current_line, locctr);

    token_line++;

} else {
    // START 지시문이 없으면 LOCCTR를 0으로 초기화
    locctr = 0;
    token_table[token_line]->addr = locctr;
    write_intermediate_file(current_line, locctr);
}

// OPCODE가 'END'가 될 때까지 라인을 처리
while (token_line < line_num) {
    current_line = input_data[token_line];
    token_parsing(current_line);

    if (token_table[token_line] != NULL) {

        // CSECT 지시문 확인
        if (strcmp(token_table[token_line]->operator, "CSECT") == 0) {
            for (int i = 2; i < token_line; i++) {
                if (locctr < token_table[token_line - i]->addr) {
```

```
                        csect_table[sec].program_length = token_table[token_line - i]->addr - csect_start_address;
                        break;
                    } else {
                        csect_table[sec].program_length = locctr - csect_start_address;
                        break;
                    }
                }
                csect_table[sec].sec = sec;
                // 새로운 섹션 시작, 프로그램 카운터 재설정
                locctr = starting_address;
                csect_start_address = locctr;
                sec++;

                // 섹션 이름을 심볼 테이블에 추가
                if (token_table[token_line]->label != NULL) {
                    add_to_symtab(token_table[token_line]->label, locctr, 0, sec);
                }
                // 라인을 중간 파일에 작성
                write_intermediate_file(current_line, locctr);
                token_table[token_line]->addr = locctr;
                token_line++;
                continue;  // CSECT에 대한 나머지 루프를 건너뜀
            }
            // EQU 지시문 확인
            if (strcmp(token_table[token_line]->operator, "EQU") == 0) {
                handle_equ_directive(token_table[token_line]->label, token_table[token_line]->operand[0]);
            }
            // EXTREF 지시문 확인
            if (strcmp(token_table[token_line]->operator, "EXTREF") == 0) {
                // EXTREF 지시문 처리
                for (int i = 0; i < MAX_OPERAND && token_table[token_line]->operand[i][0] != '₩0'; i++) {
                    handle_extref(token_table[token_line]->operand[i], sec);
                }
            }
            // 라벨을 심볼 테이블에 추가
            if (token_table[token_line]->label != NULL) {
                add_to_symtab(token_table[token_line]->label, locctr, 0, sec);
            }

            // 다른 지시문 및 명령어에 대한 중간 파일에 라인 작성
            write_intermediate_file(current_line, locctr);
            token_table[token_line]->addr = locctr;

            // 명령어 테이블에서 OPCODE의 인덱스를 검색
            int opcode_index = search_opcode(token_table[token_line]->operator);

            if (opcode_index != -1) {
                // 4형식 명령어 확인
```

```
    if (token_table[token_line]->operator[0] == '+') {
        locctr += 4;
    } else {
        locctr += inst_table[opcode_index]->format;
    }

    // 위치 카운터에 영향을 주는 특정 지시문 처리
    if (strcmp(token_table[token_line]->operator, "WORD") == 0) {
        locctr += 3;
    } if (strcmp(token_table[token_line]->operator, "RESW") == 0) {
        locctr += 3 * atoi(token_table[token_line]->operand[0]);
    } if (strcmp(token_table[token_line]->operator, "RESB") == 0) {
        locctr += atoi(token_table[token_line]->operand[0]);
    } if (strcmp(token_table[token_line]->operator, "BYTE") == 0) {
        locctr += calculate_byte_length(token_table[token_line]->operand[0]);
    } if (strcmp(token_table[token_line]->operator, "LTORG") == 0) {
        // LTORG 지시문 처리
        handle_ltorg_directive();
    }

    // "END" 지시문 확인하여 루프 종료
    if (strcmp(token_table[token_line]->operator, "END") == 0) {
        csect_table[sec].program_length = locctr;
        handle_ltorg_directive();
        break;
    }
}

// 리터럴 확인하고 리터럴 테이블에 추가
for (int i = 0; i < MAX_OPERAND; ++i) {
    int length;
    uchar *operand = token_table[token_line]->operand[i];
    if (operand[0] == '=' && (operand[1] == 'C' || operand[1] == 'X')) {
        // 리터럴 발견
        length = calculate_byte_length(operand);
        int duplicate_found = 0;  // 중복된 리터럴을 추적하는 플래그
        for (int j = 0; j < LT_num; j++) {
            if (strcmp(LTtab[j].name, operand) == 0) {
                fprintf(stderr, "Error: 중복된 리터럴 발견 - %s₩n", operand);
                duplicate_found = 1;
                // 필요한 대로 오류 처리
                break;  // 더 이상 확인할 필요 없음
            }
        }
        if (!duplicate_found) {
            // 리터럴을 리터럴 테이블에 추가
            LTtab[LT_num].leng = length;
            strcpy(LTtab[LT_num].name, operand);
            LTtab[LT_num].addr = -1;  // 아직 주소가 할당되지 않음
            LT_num++;
```

```
                    }
                }
            }

        }
        // 다음 라인 읽기
        token_line++;
    }

    // 마지막 컨트롤 섹션의 프로그램 길이 설정
    csect_table[sec].program_length = locctr;

    // 심볼 테이블 출력 생성
    make_symtab_output("symtab.txt");

    // 디버깅을 위해 각 라인의 주소를 출력
    for (int i = 0; i < line_num; i++) {
        printf("라인: %d₩n Locctr: %04X₩n", i + 1, token_table[i]->addr);
    }

    return 0;
}

// 옵코드 출력 파일을 생성하는 함수
void make_opcode_output(uchar *file_name) {
    FILE *output_file = fopen(file_name, "w");

    // 파일이 성공적으로 열렸는지 확인
    if (output_file == NULL) {
        fprintf(stderr, "출력 파일을 열 수 없습니다.₩n");
        return;
    }

    // 파일에 헤더 출력
    fprintf(output_file, "연상기호  기계 코드  형식₩n");

    // 명령어 테이블을 순회하며 정보 출력
    for (int i = 0; i < inst_index; i++) {
        fprintf(output_file, "%-10s %02X %10d₩n",
                inst_table[i]->str, inst_table[i]->op, inst_table[i]->format);
    }

    // 출력 파일 닫기
    fclose(output_file);
}

// 심볼 테이블 출력 파일을 생성하는 함수
void make_symtab_output(uchar *file_name) {
    FILE *symtab_output_file = fopen(file_name, "w");
```

```
    // 파일이 성공적으로 열렸는지 확인
    if (symtab_output_file == NULL) {
        fprintf(stderr, "%s 파일을 쓰기 위해 열 수 없습니다.₩n", file_name);
        exit(EXIT_FAILURE);
    }

    // 심볼 테이블 헤더를 파일에 출력
    fprintf(symtab_output_file, "심볼₩t주소₩t섹션₩n");
    fprintf(symtab_output_file, "-------------------------₩n");

    // 심볼 테이블을 순회하며 정보 출력
    for (int i = 0; i < sym_index; i++) {
        if (sym_table[i].addr != -1) {
            fprintf(symtab_output_file, "%s₩t%04X₩t%d₩n", sym_table[i].symbol, sym_table[i].addr,
sym_table[i].sec);
        }
    }

    // 리터럴 테이블 헤더를 파일에 출력
    fprintf(symtab_output_file, "₩n리터럴₩t주소₩t풀 번호₩n");
    fprintf(symtab_output_file, "---------------------------₩n");

    // 리터럴 테이블을 순회하며 정보 출력
    for (int i = 0; i < LT_num; i++) {
        if (LTtab[i].addr != -1) {
            fprintf(symtab_output_file, "%s₩t%04X₩t%d₩n", LTtab[i].name, LTtab[i].addr, LTtab[i].value);
        }
    }

    // 심볼 테이블 출력 파일 닫기
    fclose(symtab_output_file);
}

// 지정된 문자열과 위치 카운터(locctr)를 사용하여 중간 파일에 항목을 작성하는 함수
void write_intermediate_file(uchar *str, int locctr) {
    FILE *intermediate_file;

    // 파일에 처음 작성하는지 확인
    if (is_first_write) {
        intermediate_file = fopen("intermediate.txt", "w");
        if (intermediate_file == NULL) {
            fprintf(stderr, "오류: 중간 파일을 쓰기 위해 열 수 없습니다.₩n");
            return;
        }
        is_first_write = 0; // 첫 번째 쓰기를 나타내는 플래그 업데이트
    } else {
        // 첫 번째 쓰기가 아니면 파일을 추가 모드로 열기
        intermediate_file = fopen("intermediate.txt", "a");
        if (intermediate_file == NULL) {
            fprintf(stderr, "오류: 중간 파일을 쓰기 위해 열 수 없습니다.₩n");
```

```
            return;
        }
    }

    // 중간 파일에 형식화된 항목 작성
    fprintf(intermediate_file, "%04X %s₩n", locctr, str);

    // 중간 파일 닫기
    fclose(intermediate_file);
}

// 문자열 형식(expr)으로 제공된 식을 평가하는 함수
int evaluate_expression(uchar *expr) {
    sign = '₩0'; // 부호 변수 초기화

    // 식에서 '+' 또는 '-'가 있는지 확인
    if (strchr(expr, '+')) {
        sign = '+'; // 부호를 '+'로 설정
    } else if (strchr(expr, '-')) {
        sign = '-'; // 부호를 '-'로 설정
    }

    // '-' 또는 '+' 연산자를 기준으로 식을 토큰화
    uchar *token = strtok(expr, "-+");

    // 첫 번째 피연산자(BUFEND)를 구문 분석
    int operand1 = -1;
    if (token != NULL) {
        operand1 = search_symtab(token, sec);
        if (operand1 == -1) {
            // 심볼 테이블에서 심볼을 찾을 수 없는 경우 처리
            fprintf(stderr, "오류: 심볼을 찾을 수 없음 - %s₩n", token);
            return -1;
        }
    } else {
        // 식의 형식이 올바르지 않은 경우 처리
        fprintf(stderr, "오류: 잘못된 식 형식 - %s₩n", expr);
        return -1;
    }

    // 두 번째 피연산자를 구문 분석
    token = strtok(NULL, " ");
    int operand2 = -1;
    if (token != NULL) {
        operand2 = search_symtab(token, sec);
        if (operand2 == -1) {
            // 심볼 테이블에서 심볼을 찾을 수 없는 경우 처리
            fprintf(stderr, "오류: 심볼을 찾을 수 없음 - %s₩n", token);
            return -1;
        }
```

```
    } else {
        // 식의 형식이 올바르지 않은 경우 처리
        fprintf(stderr, "오류: 잘못된 식 형식 - %s₩n", expr);
        return -1;
    }

    // 부호를 기준으로 식을 평가합니다.
    if (sign == '+') {
        return operand1 + operand2;
    } else if (sign == '-') {
        return operand1 - operand2;
    }
}

// 오브젝트 코드 출력 파일과 리스팅 파일을 쓰기 위해 열기
void make_objectcode_output(uchar *file_name, uchar *list_name) {
    // 처음 쓰기인지 확인
    if (first_write) {
        // 오브젝트 코드 출력 파일을 쓰기 모드로 열기
        object_program_file = fopen(file_name, "w");
        // 리스팅 파일을 쓰기 모드로 열기
        listing_file = fopen(list_name, "w");

        // 두 파일이 성공적으로 열렸는지 확인
        if (object_program_file == NULL || listing_file == NULL) {
            fprintf(stderr, "오류: objfile 및 lstfile을 쓰기 위해 열 수 없습니다.₩n");
            // 오류가 발생하면 파일을 닫기
            fclose(object_program_file);
            fclose(listing_file);
            exit(1); // 오류 코드와 함께 프로그램을 종료
        }

        // 처음 쓰기 플래그 업데이트
        first_write = 0;
    } else {
        // 처음 쓰기가 아니면 오브젝트 코드 출력 파일 및 리스팅 파일을 추가 모드로 열기
        object_program_file = fopen(file_name, "a");
        listing_file = fopen(list_name, "a");

        // 두 파일이 성공적으로 열렸는지 확인
        if (object_program_file == NULL || listing_file == NULL) {
            fprintf(stderr, "오류: objfile 및 lstfile을 쓰기 위해 열 수 없습니다.₩n");
            // 오류가 발생하면 파일을 닫기
            fclose(object_program_file);
            fclose(listing_file);
            exit(1); // 오류 코드와 함께 프로그램을 종료
        }
    }
}
```

```
// 레지스터의 숫자 값을 가져오는 함수
int getREGnum(uchar *register_name) {
    // 레지스터 이름을 비교하고 해당하는 숫자 값을 반환
    if (strcmp(register_name, "A") == 0) {
        return 0;
    } else if (strcmp(register_name, "X") == 0) {
        return 1;
    } else if (strcmp(register_name, "L") == 0) {
        return 2;
    } else if (strcmp(register_name, "B") == 0) {
        return 3;
    } else if (strcmp(register_name, "S") == 0) {
        return 4;
    } else if (strcmp(register_name, "T") == 0) {
        return 5;
    } else if (strcmp(register_name, "F") == 0) {
        return 6;
    } else {
        // 잘못된 레지스터 이름 처리
        fprintf(stderr, "오류: 잘못된 레지스터 이름 : %s₩n", register_name);
        exit(EXIT_FAILURE); // 오류 코드와 함께 프로그램을 종료
    }
}

// 명령어 형식에 기반한 오브젝트 코드 생성 함수
int generate_object_code(int format) {
    int dxx;     // 임시 저장 변수
    int disp;
    int loc;
    int loc2;
    int op_index = search_opcode(token_table[token_line]->operator); // 명령어 테이블에서 명령어의 인덱스
를 가져옴
    int opcode = inst_table[op_index]->op; // 명령어 값 가져오기
    token *ct = token_table[token_line]; // 현재 토큰
    token *nt = token_table[token_line +  1]; // 다음 토큰

    switch (format) {
        case 1: // 형식 1 명령어 (FIX, FLOAT, HIO, NORM, SIO, TIO)
            object_code[0] = opcode;
            object_code[1] = '₩0';
            break;

        case 2: // 형식 2 명령어 (피연산자는 r1 또는 r1, r2 또는 n 또는 r1, n 일 수 있음)
            object_code[0] = opcode;
            if (ct->operand[0][0] == '₩0')
                printf("오류! 피연산자가 필요합니다.₩n");

            if (ct->operand[0][0] < 'A') { // 피연산자가 상수 (n)인 경우
                sscanf(ct->operand[0], "%d", &dxx);
                object_code[1] = dxx << 4; // n
```

```
    } else {
        object_code[1] = getREGnum(ct->operand[0]) << 4; // r1
    }

    if (ct->operand[1][0] != '₩0') {
        if (ct->operand[1][0] < 'A') { // 피연산자 2가 상수 (n)인 경우
            sscanf(ct->operand[1]+1, "%d", &dxx);
            object_code[1] = object_code[1] | dxx; // n
        } else {
            object_code[1] = object_code[1] | getREGnum(ct->operand[1]); // r2
        }
    }
    break;

case 3: // 형식 3 명령어
    switch (ct->operand[0][0]) {
        case '#': // 즉시 주소 지정
            object_code[0] = opcode + 1; // n=0, i=1, 따라서 +1
            if (ct->operand[0][1] >= 'A') { // PC 상대 + 즉시 주소 지정 (예: #LENGTH)
                dxx = search_symtab(ct->operand[0]+1, sec);
                if (dxx == -1)
                    printf("오류: 정의되지 않은 심볼: %s₩n", ct->operand[0]+1);
                else
                    dxx -= nt->addr; // op1과 다음 명령어의 LOCCTR 간의 차이
                object_code[1] = (dxx >> 8) & 15; // TA 상위 4비트
                object_code[1] = object_code[1] | 32; // p=1 (PC 상대)
                object_code[2] = dxx & 255; // TA 하위 8비트
            } else { // 상수 값
                sscanf(ct->operand[0]+1, "%d", &dxx);
                object_code[1] = (dxx >> 8) & 15; // TA 상위 4비트
                object_code[2] = dxx & 255; // TA 하위 8비트
            }
            break;

        case '₩0': // 피연산자 없음 (RSUB)
            object_code[0] = opcode + 3; // n=1, i=1, 따라서 +3
            object_code[1] = 0;
            object_code[2] = 0;
            break;

        default: // 기타 경우
            if (ct->operand[0][0] == '@') { // 간접 주소 지정
                object_code[0] = opcode + 2; // n=1, i=0, 따라서 +2
                loc = search_symtab(ct->operand[0]+1, sec);
                disp = loc - nt->addr; // op1과 다음 명령어의 LOCCTR 간의 차이
            } else if (ct->operand[0][0] == '=') { // 리터럴
                object_code[0] = opcode + 3; // n=1, i=1, 따라서 +3
                loc = LTtab[current_pool-1].addr; // 리터럴 테이블 (풀)에서 검색
                disp = loc - nt->addr; // op1과 다음 명령어의 LOCCTR 간의 차이
            } else {
```

```
                    object_code[0] = opcode + 3; // n=1, i=1, 따라서 +3
                    loc = search_symtab(ct->operand[0], sec);
                    disp = loc - nt->addr; // op1과 다음 명령어의 LOCCTR 간의 차이
                }

                if (loc < 0) {
                    printf("오류: 정의되지 않은 심볼: %s₩n", ct->operand[0]);
                    loc = 0;
                    object_code[1] = 0;
                    object_code[2] = 0;
                    break;
                }

                if ((abs(disp) >= 4096) && (loc >= 0)) { // 베이스 상대 주소 지정 사용
                    disp = abs(BASEADDR - loc);
                    object_code[1] = (disp >> 8) & 15; // TA 상위 4비트
                    object_code[1] = object_code[1] | 64; // b=1 (베이스 상대)
                    object_code[2] = disp & 255; // TA 하위 8비트
                    printf("opcode->%s base->%X loc->%X disp->%d₩n", inst_table[op_index]->str,
BASEADDR, loc, disp);
                } else if ((disp < 4096) && (loc >= 0)) {
                    object_code[1] = (disp >> 8) & 15; // TA 상위 4비트
                    object_code[1] = object_code[1] | 32; // p=1 (PC 상대)
                    object_code[2] = disp & 255; // TA 하위 8비트
                } else {
                    int ddd;
                    if (ct->operand[0][0] == '@') // 간접 주소 지정
                        sscanf(ct->operand[0]+1, "%d", &ddd);
                    else
                        sscanf(ct->operand[0], "%d", &ddd);
                    object_code[1] = (ddd >> 8); // TA 상위 4비트
                    object_code[2] = ddd & 255; // TA 하위 8비트
                }

                if (ct->operand[1][0] == 'X') // 인덱스
                    object_code[1] = object_code[1] + 128; // x=1
            }
            printf("<형식 3> opcode->%s obj->%02X%02X%02X₩n", inst_table[op_index]->str,
object_code[0], object_code[1], object_code[2]);
            break;

        case 4: // 형식 4
            switch (ct->operand[0][0]) {
                case '#': // 즉시 주소 지정
                    object_code[0] = opcode + 1; // n=0, i=1, 따라서 +1
                    if (ct->operand[0][1] >= 'A') { // PC 상대 + 즉시 주소 지정 (예: #LENGTH)
                        dxx = search_symtab(ct->operand[0]+1, sec);
                        if (dxx == -1)
                            printf("오류: 정의되지 않은 심볼: %s₩n", ct->operand[0]+1);
                        else
```

```
                    dxx -= nt->addr; // op1과 다음 명령어의 LOCCTR 간의 차이
                } else  // 상수 값
                    sscanf(ct->operand[0]+1, "%d", &dxx);
                object_code[1] = 16; // e=1
                object_code[2] = (dxx >> 8) & 255; // TA 상위 1바이트
                object_code[3] = dxx & 255; // TA 하위 1바이트
                break;

            default:
                if (ct->operand[0][0] == '@') { // 간접 주소 지정
                    object_code[0] = opcode + 2; // n=1, i=0, 따라서 +2
                    loc2 = search_symtab(ct->operand[0]+1, sec);
                } else {
                    object_code[0] = opcode + 3; // n=1, i=1, 따라서 +3
                    loc2 = search_symtab(ct->operand[0], sec);
                    if (loc2 < 0) { // 현재 섹션에서 찾을 수 없음
                        loc2 = search_extRtab(ct->operand[0], sec); // 외부 참조인지 확인
                        if (loc2 >= 0) {
                            // 수정 레코드에 추가
                            // L2-STARTADDR+1 -> 현재 제어 섹션의 시작부터 수정 대상 필드까지의
상대적인 주소
                            // 05 -> 레코드의 반바이트로 표시된 길이 (여기서는 05로 고정)
                            // + -> 수정 플래그 (여기서는 +로 고정)
                            sprintf(mod_record[mod_record_count],                "M%06X05+%-6s₩n",
ct->addr-csect_start_address+1, ct->operand[0]);
                            mod_record_count++;
                        }
                    }
                    if (loc2 < 0) {
                        printf("오류: 정의되지 않은 심볼: %s₩n", ct->operand[0]);
                        loc2 = 0;
                    }
                }

                if (loc2 >= 0) {
                    object_code[1] = 16; // e=1
                    object_code[2] = (loc2 >> 8); // TA 상위 1바이트
                    object_code[3] = loc2 & 255;  // TA 하위 1바이트
                } else {
                    int ddd;
                    if (ct->operand[0][0] == '@') // 간접 주소 지정
                        sscanf(ct->operand[0]+1, "%d", &ddd);
                    else
                        sscanf(ct->operand[0], "%d", &ddd);
                    object_code[1] = 16; // e=1
                    object_code[2] = (ddd >> 8); // TA 상위 1바이트
                    object_code[3] = ddd & 255;  // TA 하위 1바이트
                }

                if (ct->operand[1][0] == 'X') // 색인화
```

```
                    object_code[1] = object_code[1] + 128; // x=1
            }
            printf("<형식 4> opcode->%s obj->%02X%02X%02X%02X₩n", inst_table[op_index]->str,
object_code[0], object_code[1], object_code[2], object_code[3]);
            break;
    }
}

// 16진수 문자를 10진수로 변환하는 함수
int hexstr2dec(char H) {
    int i;
    for (i = 0; i <= 15; i++)
        if (HEXTAB[i] == H)
            return (i);
    return (-1);
}

// 리터럴을 오브젝트 코드에 작성하는 함수
int write_literal(void) {
    int n, len = 0;
    is_lt = 0; // 현재 명령어에 리터럴이 포함되어 있는지 나타내는 플래그
    token *ct = token_table[token_line]; // 현재 토큰

    // 리터럴 테이블에서 리터럴을 반복하여 확인
    for (; (LT_num < current_pool) && (LTtab[LT_num].value == LT_num); LT_num++) {
        len = 0; // 리터럴의 길이 초기화

        // 리터럴의 유형을 확인 (문자 또는 16진수)
        if (LTtab[LT_num].name[1] == 'C') { // 문자 리터럴
            n = 0;
            // 따옴표 내의 문자를 추출하여 바이트로 변환
            while (LTtab[LT_num].name[n + 3] != '₩') {
                object_code[len] = LTtab[LT_num].name[n + 3]; // 문자를 바이트로 변환
                n++;
                len++;
            }
            object_code[len] = '₩0'; // 문자열을 종료하는 null 문자 추가
        } else if (LTtab[LT_num].name[1] == 'X') { // 16진수 리터럴
            n = 0;
            // 16진수 쌍을 추출하여 바이트로 변환
            while (LTtab[LT_num].name[n + 3] != '₩') {
                // 두 16진수 문자를 하나의 바이트로 변환
                object_code[len] = hexstr2dec(LTtab[LT_num].name[n + 3]) * 16 +
hexstr2dec(LTtab[LT_num].name[n + 4]);
                n += 2;
                len++;
            }
            object_code[len] = '₩0'; // 문자열을 종료하는 null 문자 추가
        } else {
            // 문자 또는 16진수 리터럴이 아닌 경우, 10진수 리터럴로 간주
```

```
            sscanf(ct->operand[0] + 1, "%d", object_code);
            len++;
        }

        is_lt++; // 리터럴 플래그 설정
        write_listing_line(len); // 리터럴을 리스팅 파일에 작성
        is_lt = 0; // 리터럴 플래그 재설정
    } // for

    return (len); // 작성된 리터럴의 총 길이 반환
}

// 어셈블러의 두 번째 패스를 처리하는 함수
int handle_pass2(void) {
    int n = 0; // 리터럴 또는 상수의 길이를 저장하는 변수
    token *ct = token_table[token_line]; // 현재 토큰
    token *nt = token_table[token_line + 1]; // 다음 토큰
    inst_index = search_opcode(ct->operator); // 명령어 테이블에서 명령어의 인덱스
    int format = inst_table[inst_index]->format; // 명령어의 길이 (지시문의 길이는 0)

    // 명령어가 형식 4인지 확인
    if (ct->operator[0] == '+') {
        format = 4;
    }

    // 형식이 0보다 크면 어셈블러 지시문이 아님
    if (format > 0) {
        generate_object_code(format); // 명령어 형식에 따라 오브젝트 코드 생성
        write_listing_line(format); // 명령어에 대한 리스팅 라인 작성
    }

    // 형식이 0이면 어셈블러 지시문
    if (format == 0) {
        if (strcmp(ct->operator, "BASE") == 0) { // 상대 주소 지원을 위한 베이스 레지스터 값 설정
            write_listing_line(format);
            BASEADDR = search_symtab(ct->operand[0], sec); // 심볼 테이블에서 베이스 주소 검색
            if (BASEADDR < 0)
                BASEADDR = 0;
        } else if (strcmp(ct->operator, "NOBASE") == 0) { // 베이스 레지스터 값 해제
            write_listing_line(format);
            BASEADDR = 0;
        } else if (strcmp(ct->operator, "WORD") == 0) { // WORD 지시문 처리
            if (ct->operand[0][0] > 'A') { // 심볼이 사용됨
                int W3, W4 = 0;
                W3 = search_symtab(ct->operand[0], sec); // 심볼 테이블에서 심볼 검색
                // 외부 참조 처리
                if (W3 < 0) {
                    W3 = search_extRtab(ct->operand[0], sec); // 외부 참조 테이블에서 심볼 검색
                    if (W3 < 0) {
                        printf("Error: Undefined symbol: %s₩n", ct->operand[0]);
```

```
                    }
                    // + 또는 -가 있는 표현식 처리
                    if (strchr(ct->operand[0], '+') || strchr(ct->operand[0], '-')) {
                        if (strchr(ct->operand[0], '+')) {
                            sign2 = '+';
                        } else if (strchr(ct->operand[0], '-')) {
                            sign2 = '-';
                        }
                        int operand_index = 0;
                        uchar *optok = strtok(ct->operand[0], "+-");
                        while (optok != NULL) {
                            strcpy(ct->operand[operand_index], optok);
                            optok = strtok(NULL, ",₩t");
                            operand_index++;
                        }
                        W3 = search_symtab(ct->operand[0], sec);
                        if (W3 < 0) {
                            W3 = search_extRtab(ct->operand[0], sec);
                            if (W3 < 0) {
                                printf("Error: Undefined symbol: %s, %s₩n", ct->operand[0],
ct->operand[1]);
                            }
                            sprintf(mod_record[mod_record_count], "M%06X06%c%-6s₩n", ct->addr -
csect_start_address, '+', ct->operand[0]);
                            mod_record_count++;
                        }
                        W4 = search_symtab(ct->operand[1], sec);
                        if (W4 < 0) {
                            W4 = search_extRtab(ct->operand[1], sec);
                            if (W4 < 0) {
                                printf("Error: Undefined symbol: %s, %s₩n", ct->operand[0],
ct->operand[1]);
                            }
                            sprintf(mod_record[mod_record_count], "M%06X06%c%-6s₩n", ct->addr -
csect_start_address, sign2, ct->operand[1]);
                            mod_record_count++;
                        }
                        if (sign2 == '+') {
                            W3 = W3 + W4;
                        }
                        if (sign2 == '-') {
                            W3 = W3 - W4;
                        }
                    }
                    // 상수 값을 메모리에 쓰기
                    object_code[0] = (W3 >> 16) & 255;
                    object_code[1] = (W3 >> 8) & 255;
                    object_code[2] = W3 & 255;
                } else {
                    // 상수 값
```

```
                sscanf(ct->operand[0], "%d", &n);
                object_code[0] = 0;
                object_code[1] = (n >> 8) & 255;
                object_code[2] = n & 255;
            }
            // 리스팅 파일에 쓰기
            write_listing_line(3);
            return 3;
        }
    } else if (strcmp(ct->operator, "RESW") == 0) { // RESW 지시문 처리
        write_listing_line(format);
        write_text_record(); // 이전 텍스트 레코드 출력
        text_record_start = ct->addr; // 새로운 텍스트 레코드의 시작 주소 설정
    } else if (strcmp(ct->operator, "RESB") == 0) { // RESB 지시문 처리
        write_listing_line(format);
        write_text_record(); // 이전 텍스트 레코드 출력
        text_record_start = ct->addr; // 새로운 텍스트 레코드의 시작 주소 설정
    } else if (strcmp(ct->operator, "BYTE") == 0) { // BYTE 지시문 처리
        if (ct->operand[0][0] == 'C') { // 문자 리터럴
            n = 0;
            while (ct->operand[0][n + 2] != '₩') {
                object_code[n] = ct->operand[0][n + 2]; // 문자를 바이트로 변환
                n++;
            }
            object_code[n] = '₩0'; // 문자열을 종료하는 null 문자 추가
            write_listing_line(n);
            return n;
        } else if (ct->operand[0][0] == 'X') { // 16진수 리터럴
            int len = 0;
            n = 0;
            while (ct->operand[0][n + 2] != '₩') {
                // 두 16진수 문자를 하나의 바이트로 변환
                object_code[len] = hexstr2dec(ct->operand[0][n + 2]) * 16 +
hexstr2dec(ct->operand[0][n + 3]);
                n += 2;
                len++;
            }
            object_code[len] = '₩0'; // 문자열을 종료하는 null 문자 추가
            write_listing_line(len);
            return len;
        } else { // 숫자 상수
            sscanf(ct->operand[0], "%d", &n);
            object_code[0] = n;
            write_listing_line(n);
            return 1;
        }
    } else if (strcmp(ct->operator, "LTORG") == 0) { // LTORG 지시문 처리
        text_record_start = ct->addr; // 텍스트 레코드의 시작 주소 설정
        write_listing_line(format);
        n = write_literal(); // 리터럴 테이블 (풀) 출력
```

```
            current_pool++;
            return n;
        } else if (strcmp(ct->operator, "CSECT") == 0) { // CSECT 지시문 처리
            text_record_start = 0; // TS -> 텍스트 레코드의 시작 주소
            csect_start_address = ct->addr; // 제어 섹션의 시작 주소 설정
            fprintf(listing_file, "₩n");
            write_listing_line(format);

            sec++; // 제어 섹션 번호 증가
            write_text_record(); // 이전 텍스트 레코드 출력
            // 수정 레코드 출력
            while (mod_last < mod_record_count) {
                fprintf(object_program_file, "%s", mod_record[mod_last]);
                mod_last++;
            }
            // 종료 레코드 출력
            if (sec == 1) { // 기본 섹션
                fprintf(object_program_file, "E%06X₩n₩n", starting_address);
            } else {
                fprintf(object_program_file, "E₩n₩n");
            }
            // 헤더 레코드 출력
            fprintf(object_program_file, "H%-6s%06X%06X₩n", ct->label, 0,
csect_table[sec].program_length);
        } else if (strcmp(ct->operator, "EQU") == 0) { // EQU 지시문 처리
            line_num += 5;
            fprintf(listing_file, "%4d     %04X  %-10s ", line_num, ct->addr, input_data[token_line]);
            fprintf(listing_file, "₩n");
        } else if (strcmp(ct->operator, "EXTDEF") == 0) { // EXTDEF 지시문 처리 (외부 정의)
            // EXTDEF 지시문의 각 피연산자 처리
            for (int i = 0; i < MAX_OPERAND && ct->operand[i][0] != '₩0'; ++i) {
                handle_extdef(ct->operand[i]);
            }
            fprintf(object_program_file, "D"); // 정의 레코드
            // 외부 심볼과 해당 상대 주소 출력
            for (int i = 0; i < MAX_OPERAND && ct->operand[i][0] != '₩0'; ++i) {
                int a1 = search_extDtab(ct->operand[i]);
                fprintf(object_program_file, "%-6s%06X", ct->operand[i], a1);
            }
            line_num += 5;
            fprintf(listing_file, "%4d          %-10s ", line_num, input_data[token_line]);
            fprintf(object_program_file, "₩n");
            fprintf(listing_file, "₩n");
        } else if (strcmp(ct->operator, "EXTREF") == 0) { // EXTREF 지시문 처리 (외부 참조)
            fprintf(object_program_file, "R"); // 참조 레코드
            // 현재 제어 섹션에서 참조된 외부 심볼 출력
            for (int i = 0; i < extRefCount; ++i) {
                if (extRef[i].sec == sec) {
                    fprintf(object_program_file, "%-6s", extRef[i].symbol);
                }
```

```
            }
            line_num += 5;
            fprintf(listing_file, "%4d              %-10s", line_num, input_data[token_line]);
            fprintf(object_program_file, "₩n");
            fprintf(listing_file, "₩n");
        }
    }

    return (format);
}

// 오브젝트 프로그램에 텍스트 레코드를 작성하는 도우미 함수
void write_text_record(void) {
    if (text_record_ctr > 0) { // 텍스트 레코드에 내용이 있는지 확인
        // 시작 주소, 길이 및 내용과 함께 텍스트 레코드를 오브젝트 프로그램 파일에 작성
        fprintf(object_program_file, "T%06X%02X%s₩n", text_record_start, text_record_ctr / 2,
text_record);
        text_record_ctr = 0; // 다음 레코드를 위해 텍스트 레코드 카운터 초기화
    }
}

// 명령어 형식에 따라 리스팅 파일에 리스팅 라인을 작성하는 함수
// 다양한 형식 (0, 1, 2, 3, 4)을 처리하고 해당 정보를 리스팅 파일에 작성한다
int write_listing_line(int format) {
    line_num += 5; // 라인 번호를 5 증가시킴

    // 토큰 테이블에서 현재 토큰과 다음 토큰을 가져옴
    token *current_token = token_table[token_line];
    token *next_token = token_table[token_line + 1];

    // 명령어 형식에 따라 switch 문 실행
    switch (format) {
        case 0: // 어셈블러 디렉티브
            // 라벨이 있는지 확인하고 그에 따라 리스팅 라인 형식 지정
            if (current_token->label == NULL) {
                fprintf(listing_file, "%4d              %-10s ₩n", line_num, input_data[token_line]);
            } else {
                fprintf(listing_file, "%4d              %04X      %-10s ₩n", line_num, current_token->addr,
input_data[token_line]);
            }
            break;

        case 1: // 형식 1
            // 새로운 텍스트 레코드를 시작해야 하는지 확인
            if ((text_record_ctr + 2) > 60) {
                write_text_record();
                text_record_start = current_token->addr;
            }
            // 텍스트 레코드를 객체 코드로 업데이트
            sprintf(text_record + text_record_ctr, "%02X", object_code[0]);
```

```
            text_record_ctr += 2;

            // 리터럴이 있는지 확인하고 그에 따라 리스팅 라인 형식 지정
            if (is_lt > 0) {
                fprintf(listing_file, "%4d  %04X  *  %s  ", line_num, current_token->addr,
LTtab[current_pool - 1].name);
            } else {
                fprintf(listing_file, "%4d  %04X  %-10s  ", line_num, current_token->addr,
input_data[token_line]);
            }
            // 객체 코드를 리스팅 파일에 작성
            fprintf(listing_file, "%02X₩n", object_code[0]);
            break;

        case 2: // 형식 2
            // 새로운 텍스트 레코드를 시작해야 하는지 확인
            if ((text_record_ctr + 4) > 60) {
                write_text_record();
                text_record_start = current_token->addr;
            }
            // 텍스트 레코드를 객체 코드로 업데이트
            sprintf(text_record + text_record_ctr, "%02X%02X", object_code[0], object_code[1]);
            text_record_ctr += 4;

            // 리터럴이 있는지 확인하고 그에 따라 리스팅 라인 형식 지정
            if (is_lt > 0) {
                fprintf(listing_file, "%4d  %04X  *  %s  ", line_num, current_token->addr,
LTtab[current_pool - 1].name);
            else {
                fprintf(listing_file, "%4d  %04X  %-10s  ", line_num, current_token->addr,
input_data[token_line]);
            }
            // 객체 코드를 리스팅 파일에 작성
            fprintf(listing_file, " %02X%02X₩n", object_code[0], object_code[1]);
            break;

        case 3: // 형식 3
            // 새로운 텍스트 레코드를 시작해야 하는지 확인
            if ((text_record_ctr + 6) > 60) {
                write_text_record();
                text_record_start = current_token->addr;
            }
            // 텍스트 레코드를 객체 코드로 업데이트
            sprintf(text_record + text_record_ctr, "%02X%02X%02X", object_code[0], object_code[1],
object_code[2]);
            text_record_ctr += 6;

            // 리터럴이 있는지 확인하고 그에 따라 리스팅 라인 형식 지정
            if (is_lt > 0) {
                fprintf(listing_file, "%4d  %04X  *  %s  ", line_num, current_token->addr,
```

```
LTtab[current_pool - 1].name);
            } else {
                fprintf(listing_file,  "%4d          %04X      %-10s  ",  line_num,  current_token->addr,
input_data[token_line]);
            }
            // 객체 코드를 리스팅 파일에 작성
            fprintf(listing_file, " %02X%02X%02X₩n", object_code[0], object_code[1], object_code[2]);
            break;

        case 4: // 형식 4
            // 새로운 텍스트 레코드를 시작해야 하는지 확인
            if ((text_record_ctr + 8) > 60) {
                write_text_record();
                text_record_start = current_token->addr;
            }
            // 텍스트 레코드를 객체 코드로 업데이트
            sprintf(text_record + text_record_ctr, "%02X%02X%02X%02X", object_code[0], object_code[1],
object_code[2], object_code[3]);
            text_record_ctr += 8;

            // 리터럴이 있는지 확인하고 그에 따라 리스팅 라인 형식 지정
            if (is_lt > 0) {
                fprintf(listing_file,  "%4d          %04X     *     %s  ",  line_num,  current_token->addr,
LTtab[current_pool - 1].name);
            } else {
                fprintf(listing_file,  "%4d          %04X      %-10s  ",  line_num,  current_token->addr,
input_data[token_line]);
            }
            // 객체 코드를 리스팅 파일에 작성
            fprintf(listing_file, "%02X%02X%02X%02X₩n", object_code[0], object_code[1], object_code[2],
object_code[3]);
            break;
    }
}

// 어셈블러의 패스 2를 수행하는 함수
int assem_pass2() {
    // 변수 초기화
    line_num = 0;
    LT_num = 0;
    token_line = 0;
    inst_index = 0;
    sec = 0;

    // 객체 코드 출력 및 리스팅 파일 초기화
    make_objectcode_output("objectprogram.txt", "list.txt");

    // 토큰 테이블에서 첫 번째 토큰을 가져옴
    token *fl = token_table[0];
```

```
    // 첫 번째 명령어가 START인지 확인
    if (strcmp(fl->operator, "START") == 0) {
        // 피연산자에서 시작 주소를 추출하고 16진수로 변환
        starting_address = strtol(fl->operand[0], NULL, 16);

        // START 디렉티브에 대한 리스팅 라인 작성
        write_listing_line(0); // 리스팅 파일 출력 시작

        // 객체 프로그램 파일에 헤더 레코드 작성
        fprintf(object_program_file, "H%-6s%06X%06X₩n", fl->label, starting_address,
csect_table[sec].program_length);

        // 다음 토큰으로 이동
        token_line++;
    } else {
        // START 디렉티브를 찾을 수 없는 경우 오류 표시
        printf("Error: START 디렉티브를 찾을 수 없습니다.₩n");
    }

    // END 디렉티브가 나타날 때까지 토큰 처리
    while (1) {
        // 현재 토큰에 대한 패스 2 처리
        handle_pass2();

        // 현재 토큰이 END 디렉티브인지 확인
        if (strcmp(token_table[token_line]->operator, "END") == 0) {
            // END 디렉티브에 대한 리스팅 라인 작성
            write_listing_line(0);

            // 현재 섹션에 대한 텍스트 레코드 작성
            write_text_record();

            // 리터럴을 위한 텍스트 레코드 시작 주소 업데이트
            text_record_start = token_table[token_line]->addr;

            // 리터럴 풀을 텍스트 레코드에 작성
            write_literal();

            // 리터럴 풀을 위한 텍스트 레코드 작성
            write_text_record();

            // 객체 프로그램 파일에 수정 레코드 작성
            while (mod_last < mod_record_count) {
                fprintf(object_program_file, "%s", mod_record[mod_last]);
                mod_last++;
            }

            // 객체 프로그램 파일에 END 레코드 작성
            if (sec == 1) { // 기본 섹션
                fprintf(object_program_file, "E%06X₩n₩n", starting_address);
```

```
            } else {
                fprintf(object_program_file, "E₩n₩n");
            }

            // 루프 종료: END 디렉티브를 만나면
            break;
        }

        // 다음 토큰으로 이동
        token_line++;
    }

    // 리스팅 및 객체 프로그램 파일 닫기
    fclose(listing_file);
    fclose(object_program_file);

    // 성공적으로 실행되었음을 나타내기 위해 0 반환
    return (0);
}
```

5장 기대효과 및 결론

SIC/XE Assembler를 구현함으로서 컴퓨터 아키텍쳐 및 시스템 프로그램의 개념을 이해할 수 있다. 더 나아가 어셈블러를 직접 구현함으로서 하드웨어와 소프트웨어 간 상호작용에 대한 개념도 이해할 수 있다. 코드를 생성하고 최적화하는 과정을 연구하고 연습하여 프로그래밍 역량을 키울 수 있다. 또한, Low Level 프로그래밍에 대한 이해와 복잡한 시스템을 다루는 능력을 기를 수 있었다. 어셈블리 언어의 동작 원리와 문법에 대한 이해를 목적으로 더 나아가 복잡한 시스템에서 어셈블리 코드를 이해하는 능력을 기른다. C언어로 본 프로젝트를 수행하며 C 라이브러리에 대한 활용과 C 프로그래밍 역량을 강화한다.

첨부　프로그램 소스파일

Github Repository: my_assembler_20194318

my_assembler_20194318.h

my_assembler_20194318.c

my_assembler_20194318_main.c

input.txt

inst.data