숭실대학교 Soongsil University

프로그램 보고서

나는 숭실대학교 전자정보공학부의 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 2025
프로젝트 명	Project #1 - C언어로 SIC/XE Assembler 만들기
교과목 교수	최 재 영
제출인	전자정보공학(학과) 학번:20203023 성명:우상욱(출석번호: 222)
제출일	2025년 4월 25일

차 례

1장 프로젝트 개요

1.1 동기 및 목적

2장 설계/구현 아이디어

2.1 기술적 배경지식

2.2 구현 아이디어

3장 수행 결과

3.1 전체 시스템 구현 내용

4장 결론 및 보충할 점

5장 소스코드(+주석)

1장 프로젝트 개요

1.1 동기 및 목적

본 프로젝트는 SIC/XE 머신 구조에 대한 이해를 바탕으로, 어셈블리 소스 코드를 object code로 변환하는 어셈블러를 직접 구현하는 것을 목표로 한다.

SIC/XE 머신의 어셈블리어의 문법과 명령어 포맷(format 1~4), nixbpe 비트의 역할, 그리고 Object Code 생성 과정을 단계별로 학습 및 구현하여 어셈블러에 대한 이해를 높이고자 한다.

Symbol Table 및 Literal Table의 생성과 관리, Control Section과 External Reference/Definition 처리, Modification Record 작성 등 어셈블러의 핵심 기능을 직접 구현함으로써 시스템 프로그래밍과 C 언어의 이해를 높인다.

2장 설계/구현 아이디어

2.1 기술적 배경지식

SIC/XE 머신은 1바이트부터 4바이트까지 총 4가지 포맷(format 1~4)을 지원하며, SIC 머신의 모든 명령어를 포함한다. 주소 지정 방식은 nixbpe 비트를 통해 구분된다.

- n = 1일 경우 indirect 주소 지정
- i = 1일 경우 immediate 주소 지정
- x는 index 주소 지정
- b, p는 각각 base-relative, PC-relative 주소 지정
- n=i=0 이면 SIC 형식, n=i=1이면 SIC/XE 형식의 명령어로 해석된다.

literal은 LTORG, END에서 정의되며 WORD, BYTE, RESB, RESW 등의 지시어는 메모리에 공간을 차지하게 된다. 본 프로젝트에서는 2-pass 방식의 어셈블러를 구현한다. 2-pass 방식에서 pass1은 Symbol Table과 Literal Table을 구성하고 pass2에서 pass1의 정보를 가지고 object code를 생성한다.

2.2 구현 아이디어

input.txt는 간격은 모두 \t, \n으로 명령어들이 구분된다. 한 줄씩 읽어와서 토큰화 할 수 있다.

Pass1: 소스 파일을 라인 단위로 읽고, 이를 토큰화하여 token_table에 저장하며, 동시에 Symbol Table과 Literal Table을 구성한다. 이때 LOCCTR을 관리하여 각 명령어 또는 데이터의 주소를 계산하고 저장한다. LOCCTR 값은 각 instruction마다 저장하여 Pass2에서 target address와 program counter 값의 차이를 구하여 format을 정할 때 사용한다.

Pass2: token_table, Symbol Table, literal Table을 기반으로 실제 Object Code를 생성한다. 명령어 포맷 2/3/4에 따라 기계어를 인코딩하고, nixbpe 비트를 설정하여 명령어 형태를 결정한다. External Symbol의 경우 Modification Record를 등록한다. 이를 위해 symbol table의 범위를 나눠서 검색한다.

3장 수행 결과

3.1 전체 시스템 구현 내용

input.txt를 입력받아 output_symtab.txt와 output_littab.txt를 생성하고 최종적으로 output_objectcode.txt를 생성한다.

input.txt

```
COPY    START   0   COPY FILE FROM IN TO OUTPUT
    EXTDEF  BUFFER,BUFEND,LENGTH
    EXTREF  RDREC,WRREC
FIRST   STL RETADR  SAVE RETURN ADDRESS
CLOOP   +JSUB   RDREC   READ INPUT RECORD
    LDA LENGTH  TEST FOR EOF (LENGTH = 0)
    COMP    #0
    JEQ ENDFIL  EXIT IF EOF FOUND
    +JSUB   WRREC   WRITE OUTPUT RECORD
    J   CLOOP   LOOP
ENDFIL  LDA =C'EOF' INSERT END OF FILE MARKER
    STA BUFFER
    LDA #3  SET LENGTH = 3
    STA LENGTH
    +JSUB   WRREC   WRITE EOF
    J   @RETADR RETURN TO CALLER
RETADR  RESW    1
LENGTH  RESW    1   LENGTH OF RECORD
    LTORG
BUFFER  RESB    4096    4096-BYTE BUFFER AREA
BUFEND  EQU *
MAXLEN  EQU BUFEND-BUFFER   MAXIMUM RECORD LENGTH
RDREC   CSECT
.
.   SUBROUTINE TO READ RECORD INTO BUFFER
.
    EXTREF  BUFFER,LENGTH,BUFEND
    CLEAR   X   CLEAR LOOP COUNTER
    CLEAR   A   CLEAR A TO ZERO
    CLEAR   S   CLEAR S TO ZERO
    LDT MAXLEN
RLOOP   TD  INPUT   TEST INPUT DEVICE
    JEQ RLOOP   LOOP UNTIL READY
    RD  INPUT   READ CHARACTER INTO REGISTER A
    COMPR   A,S TEST FOR END OF RECORD (X'00')
    JEQ EXIT    EXIT LOOP IF EOR
```

```
    +STCH   BUFFER,X    STORE CHARACTER IN BUFFER
    TIXR    T   LOOP UNLESS MAX LENGTH
    JLT RLOOP   HAS BEEN REACHED
EXIT    +STX    LENGTH  SAVE RECORD LENGTH
    RSUB        RETURN TO CALLER
INPUT   BYTE    X'F1'   CODE FOR INPUT DEVICE
MAXLEN  WORD    BUFEND-BUFFER
WRREC   CSECT
.
.   SUBROUTINE TO WRITE RECORD FROM BUFFER
.
    EXTREF  LENGTH,BUFFER
    CLEAR   X   CLEAR LOOP COUNTER
    +LDT    LENGTH
WLOOP   TD  =X'05'  TEST OUTPUT DEVICE
    JEQ WLOOP   LOOP UNTIL READY
    +LDCH   BUFFER,X    GET CHARACTER FROM BUFFER
    WD  =X'05'  WRITE CHARACTER
    TIXR    T   LOOP UNTIL ALL CHARACTERS
    JLT WLOOP   HAVE BEEN WRITTEN
    RSUB        RETURN TO CALLER
    END FIRST
```

output_symtab.txt

```
COPY    0000
FIRST   0000
CLOOP   0003
ENDFIL  0017
RETADR  002a
LENGTH  002d
BUFFER  0033
BUFEND  1033
MAXLEN  1033
RDREC   0000
RLOOP   0009
EXIT    0020
INPUT   0027
MAXLEN  0028
WRREC   0000
WLOOP   0006
```

```
COPY    0000
FIRST   0000
CLOOP   0003
ENDFIL  0017
RETADR  002a
LENGTH  002d
BUFFER  0033
BUFEND  1033
MAXLEN  1033

RDREC   0000
RLOOP   0009
EXIT    0020
INPUT   0027
MAXLEN  0028

WRREC   0000
WLOOP   0006
file generating is done: output_symtab.txt
```

output_littab.txt

```
=C'EOF' 0030
=X'05'  001b
```

```
=C'EOF' 0030
=X'05'  001b
file generating is done: output_littab.txt
```

output_objectcode.txt

```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002d
RRDREC WRREC
T0000001d1720274b1000000320232900003320074b1000003f2fec0320160f2016
T00001d0d0100030f200a4b1000003e2000
T00003003454f46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E000000

HRDREC 00000000002b
RBUFFERLENGTHBUFEND
T0000001db410b400b44077201fe3201b332ffadb2015a0043320095790000b850
T00001d0e3b2fe9131000004f0000f1000000
M00001805+BUFFER
M00002105+LENGTH
M00002806+BUFEND
M00002806-BUFFER
E

HWRREC 00000000001c
RLENGTHBUFFER
T0000001cb41077100000e32012332ffa53900000df2008b8503b2fee4f000005
```

```
M00000305+LENGTH
M00000d05+BUFFER
E
```

```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002d
RRDREC WRREC
T0000001d1720274b1000000320232900003320074b1000003f2fec0320160f2016
T00001d0d0100030f200a4b1000003e2000
T00003003454f46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E000000

HRDREC 00000000002b
RBUFFERLENGTHBUFEND
T0000001db410b400b44077201fe3201b332ffadb2015a00433200957900000b850
T00001d0e3b2fe9131000004f0000f1000000
M00001805+BUFFER
M00002105+LENGTH
M00002806+BUFEND
M00002806-BUFFER
E

HWRREC 00000000001c
RLENGTHBUFFER
T0000001cb41077100000e32012332ffa53900000df2008b8503b2fee4f000005
M00000305+LENGTH
M00000d05+BUFFER
E

file generating is done: output_objectcode.txt
```

4장 결론 및 보충할 점

control section 문법을 따르는 코드만 동작 가능한 어셈블러이다. 또한 LTORG, BYTE, WORD 등의 지시어에서 문자열이 길어지거나, 리터럴(literal)의 개수가 매우 많아지는 경우 예외 처리가 미흡하여 안정적인 동작을 할 수 없다. 또한 USE 지시어, 기본 SIC 머신 형식의 코드 등 다양한 문법에 대한 처리는 현재 지원되지 않는다. 추후 다음과 같은 기능 개선이 필요하다:

- 다양한 지시어(예: USE, ORG, BASE 등) 지원
- 기본 SIC 머신 명령어 포맷에 대한 호환성 확보
- 긴 문자열 처리 및 리터럴 다량 사용에 대한 예외 처리 강화
- 다양한 데이터 형식에 대한 유연한 대응

정리하자면 더 폭넓은 범위의 어셈블리 코드에 대응할 수 있는 어셈블러로 발전시켜야 한다.

5장 소스코드(+주석)

새로운 함수 / 기본 함수로 색을 구분하였습니다.

- **init_inst_file(char* inst_file)**

inst_file.txt를 입력으로 받아 inst_table을 초기화한다.

```
int init_inst_file(char* inst_file)
{
    FILE* file;
    int errno;
    char line[30] = { NULL };
    inst_index = 0;//inst_index init

    /* add your code here */
    fopen_s(&file, inst_file, "r");
    if (file == NULL) {//exception
        printf("Error opening inst_table.txt");
        return -1;
    }
    while (fgets(line, sizeof(line), file) != NULL) {
        inst_table[inst_index] = (inst*)malloc(sizeof(inst));
        sscanf_s(line, "%s %d %hhx %d",
            inst_table[inst_index]->str, sizeof(inst_table[inst_index]->str),
            &inst_table[inst_index]->format,
            &inst_table[inst_index]->op,
            &inst_table[inst_index]->ops);//using sscanf for instruction table
        inst_index++;
    }
    fclose(file);
}
```

- **init_input_file(char* input_file)**

input.txt를 입력으로 받아 input_data를 초기화한다.

```
int init_input_file(char* input_file)
{
    FILE* file;
    int errno;
    char line[100] = { NULL };
    line_num = 0;//input_data index init

    /* add your code here */
    fopen_s(&file, input_file, "r");
    if (file == NULL) {//exeption
        printf("Error opening inst_table.txt");
        return -1;
```

```
    }
    while (fgets(line, sizeof(line), file) != NULL) {
        if (line_num < MAX_LINES) {
            input_data[line_num] = (char*)malloc(100);
            if (input_data[line_num] == NULL) {
                return -1;
            }
            input_data[line_num] = _strdup(line);
            line_num++;
        }
    }
    return -1;
}
```

- **token_parsing(char* str)**

초기화된 input_data를 token 단위로 잘라서 저장한다. 패스 1에서 사용된다.

```
int token_parsing(char* str)
{
    int max_split = 8;//max split time
    char* arr_line[8] = { NULL };//init
    char* arr_operand[MAX_OPERAND];
    int len1 = 0;
    int len2 = 0;//operand length

    if (str == NULL) return -1;
    if (str == '.') return -1;//. is comment => skip the comment line

    // initial block
    token_table[token_line] = (token*)malloc(sizeof(token));
    if (token_table[token_line] == NULL) return -1; //NULL exception

    token_table[token_line]->label = (char*)malloc(6 * sizeof(char));
    token_table[token_line]->operator = (char*)malloc(6 * sizeof(char));

    if (token_table[token_line]->label == NULL ||
        token_table[token_line]-> operator == NULL) return -1;

    for (int i = 0; i < 6; i++) {
        token_table[token_line]->label[i] = NULL;
        token_table[token_line]->operator[i] = NULL;
    }
    for (int i = 0; i < MAX_OPERAND; i++) {
        token_table[token_line]->operand[i] = (char*)malloc(6 * sizeof(char));
        for (int j = 0; j < 6; j++) {
            token_table[token_line]->operand[j] = NULL;//each operand max length is 6
```

```
        }
    }
    /*
    ////////////////////////////////////////////////////////////////////////////////////////
    [0] => label [1] => operator [2]=> operand [3] => tap  split 이런 식으로 저장 됨.
    ////////////////////////////////////////////////////////////////////////////////////////
    */
    len1 = split_tap(str, arr_line); // split like the comment above

    token_table[token_line]->label = _strdup(arr_line[0]);//label parssing
    token_table[token_line]->operator = _strdup(arr_line[1]);//operator parssing

    if (!(arr_line[2] == NULL)) {
        len2 = split_comma(arr_line[2], arr_operand);// operand is divided by comma
        for (int i = 0; i < len2; i++) {
            token_table[token_line]->operand[i] = _strdup(arr_operand[i]);//operand parssing
        }
    }
    token_line++;
    return 0;
}
```

- **split_tap(char* data, char* arr[])**

label과 operator 들은 모두 tap으로 구분됨으로 tap을 기준으로 문자열을 자른다.

```
int split_tap(char* data, char* arr[]) {
    int cnt = 0;
    char* start = data;
    char* end;
    data[strcspn(data, "\n")] = '\0';//\n remove
    while (cnt < 8) {
        end = strchr(start, '\t');//\t pointer
        if (end == NULL) {//when \t all remove
            arr[cnt++] = _strdup(start);
            break;
        }
        else {
            int len = end - start;
            char* token = (char*)malloc(len + 1);
            strncpy_s(token, len + 1, start, len);
            token[len] = '\0';//string slicing
            arr[cnt++] = token;
        }
        start = end + 1;
    }
}
```

- **split_comma(char* data, char* arr[])**

operand는 comma로 구분되어 최대 3개 input으로 들어올 수 있다.

```
int split_comma(char* data, char* arr[]) {
    char* context = NULL;
    data[strcspn(data, "\n")] = '\0';//\n remove
    char* token = strtok_s(data, ",", &context);//, slicing
    int cnt = 0;
    while (token != NULL) {//token copy
        arr[cnt] = token;
        token = strtok_s(NULL, ",", &context);
        cnt++;
    }
    return cnt;
}
```

- **literal_check(char* temp)**

literal을 한 번 선언되지만 여러 번 사용할 수 있기 때문에 중복으로 table에 등록되는 것을 막기 위한 중복 체크 함수이다.

```
int literal_check(char* temp) {
    int i = 0;
    for (i = 0; i < literal_index; i++) {
        if (strcmp(literal_table[i].literal,temp) == 0) {
            return 0;
        }
    }
    return -1;
}
```

- **assem_pass1(void)**

token_parsing() 함수를 이용하여 input_data를 label, operator, operand로 잘라 token에 저장 후 symbol table, literal table을 만들고 주솟값을 지정한다.

```
static int assem_pass1(void)
{
    int i = 0;
    int j = 0;

    int index;//instruction table index
    sym_index = 0;// symbol table index
    char* temp = NULL;//space for operator => argument for search_opcode
    int isFormat4 = 0;

    /*literal 관련 변수 */
    literal_index = 0;
```

```
    int literal_len = 0;
    int literal_flag = 0;

    int file_sec = 0;//0, 1, 2

    for (i = 0; i < line_num; i++) {// whole line process => tokenization
        isFormat4 = 0;//format4 init
        temp = NULL;//temp init

        /*token_parsing 잘 안됐을 때 예외 처리*/
        if (token_parsing(input_data[i]) == -1) return -1;
        if (token_table[token_line - 1] == NULL) return -1;

        //START일때
        if (token_table[token_line - 1]->operator != NULL &&
            (strcmp("START",token_table[token_line - 1]->operator) == 0)) {
            locctr = 0;//locctor init
            token_table_addr[token_line-1] = locctr;//ta-pc를 위한 값 저장
            strcpy_s(sym_table[sym_index].symbol, sizeof(sym_table[sym_index].symbol),
                    token_table[token_line - 1]->label);
            sym_table[sym_index].addr = locctr;
            sym_index++;
            continue;
        }

        //CSEC과 END일 때 literal 처리
        if (token_table[token_line - 1]->operator != NULL) {
            if ((strcmp("CSECT", token_table[token_line - 1]->operator) == 0)
                || (strcmp("END", token_table[token_line - 1]->operator) == 0)) {
                if (literal_flag) {
                    literal_flag = 0;
                    for (; j < literal_index; j++) {
                        literal_table[j].addr = locctr;
                        literal_len = strlen(literal_table[j].literal);
                        if (literal_table[j].literal[1] == 'C') {//char 처리
                            locctr += (literal_len - 4);
                        }
                        else if (literal_table[j].literal[1] == 'X') {//byte 처리
                            locctr += (literal_len - 4) / 2;
                        }
                    }
                }
                code[file_sec++].p_length = locctr;//program length 처리(header)
            }
        }
```

```
        if (token_table[token_line - 1]->operator != NULL &&
            (strcmp(token_table[token_line - 1]->operator,"CSECT") == 0)) { //if CSECT
            locctr = 0;//if CSECT locctr init

            //symbol table 저장
            strcpy_s(sym_table[sym_index].symbol, sizeof(sym_table[sym_index].symbol),
                    token_table[token_line - 1]->label);
            sym_table[sym_index].addr = locctr;
            sym_index++;

            //address 저장
            token_table_addr[token_line - 1] = locctr;

            continue;
        }

        //symbol table register
        // NULL pointer check and non valid value cheking and comment check
        if (token_table[token_line - 1]->label != NULL
            && token_table[token_line - 1]->label[0] != '\0'
            && token_table[token_line - 1]->label[0] != '.'
            && token_table[token_line - 1]->label[0] != '*') {
            //if label is valid, store in symbol table
            strcpy_s(sym_table[sym_index].symbol, sizeof(sym_table[sym_index].symbol),
                    token_table[token_line - 1]->label);
            sym_table[sym_index].addr = locctr;//label 주소값 및 label 문자열 저장
            sym_index++;
        }

        //= literal checking 및 locctor update
        if ((token_table[token_line - 1]->operator != NULL)
            && literal_flag && (strcmp("LTORG", token_table[token_line - 1]->operator) == 0)) {
            literal_flag = 0;
            for (; j < literal_index; j++) {
                literal_table[j].addr = locctr;
                token_table_addr[token_line - 1] = locctr;
                literal_len = strlen(literal_table[j].literal);
                if (literal_table[j].literal[1] == 'C') {
                    locctr += (literal_len-4);
                }
                else if (literal_table[j].literal[1] == 'X') {
                    locctr += (literal_len - 4) / 2;
                }
            }
```

```
            continue;
        }

        //literal 일 때 literal table에 저장하고 주소값 나중에 업데이트하여 literal table에 등록
        if (token_table[token_line - 1]->operand[0] != NULL
            && (token_table[token_line - 1]->operand[0][0] == '=')) {
            if (literal_check(token_table[token_line - 1]->operand[0])) { // duplicate checking
                literal_flag = 1;
                literal_table[literal_index].literal    =    _strdup(token_table[token_line    -
1]->operand[0]);
                literal_index++;
            }
        }

        //+ fromat4 checking
        if (token_table[token_line - 1]->operator != NULL) {
            temp = token_table[token_line - 1]->operator;
            if (token_table[token_line - 1]->operator[0] == '+') {
                temp++;
                isFormat4 = 1;
            }
        }

        token_table_addr[token_line - 1] = locctr;//주소값 저장

        //search_opcode
        if (temp == NULL) {
            index = -1;
        }
        else {
            index = search_opcode(temp);//if it doesn't exist, return -1;
            if (index != -1) {
                if (inst_table[index]->format == 2) {//format 2
                    locctr += 2;
                }
                else {
                    if (isFormat4) {//format 4
                        locctr += 4;
                    }
                    else {//format 3
                        locctr += 3;
                    }
                }
            }
            if (strcmp(temp, "RESW") == 0) {//if RESW ,add word size
```

```
                if (token_table[token_line - 1]->operand == NULL) {
                    return -1;
                }
                else {
                    locctr += atoi(token_table[token_line - 1]->operand[0]) * 3;//word
                }
            }
            else if (strcmp(temp, "RESB") == 0) {//if RESW ,add byte size
                if (token_table[token_line - 1]->operand == NULL) {
                    return -1;
                }
                else {
                    locctr += atoi(token_table[token_line - 1]->operand[0]);//byte
                }
            }
            if (strcmp(temp, "WORD") == 0) {//if RESW ,add word size
                locctr += 3;
            }
            else if (strcmp(temp, "BYTE") == 0) {//if RESW ,add byte size
                locctr++;
            }
        }
    }
    return 0;
}
```

- **search_opcode(char* str)**

token의 operator를 opcode롤 변환하기 위해 사용하는 함수이다. inst_table의 index를 반환한다.

```
int search_opcode(char* str)
{
    int i = 0;
    for (i = 0; i < inst_index; i++) {
        if (strcmp(inst_table[i]->str, str) == 0) {
            return i;//return index
        }
    }
    return -1;
}
```

- **make_symtab_output(char* file_name)**

pass1 작업에서 만들어진 token들을 활용하여 output_symtab.txt를 만든다.

```
void make_symtab_output(char* file_name)
{
    /* add your code here */
```

```
    FILE* fp;
    int i = 0;

    fopen_s(&fp, file_name, "w");//open file
    if (fp == NULL) {//NULL exception
        printf("Failure");
        return;
    }
    //function 3개 이므로 3번 나눠서
    for (i = 0; i < 2; i++) {
        fprintf(fp, "%s\t%04x\n", sym_table[i].symbol, sym_table[i].addr);
        printf("%s\t%04x\n", sym_table[i].symbol, sym_table[i].addr);
    }
    for (i = 2; i < sym_index; i++) {
        if (sym_table[i].addr == 0) printf("\n");
        fprintf(fp, "%s\t%04x\n", sym_table[i].symbol, sym_table[i].addr);
        printf("%s\t%04x\n", sym_table[i].symbol, sym_table[i].addr);
    }
    fclose(fp); //file close
    printf("file generating is done: %s\n\n", file_name);
}
```

- **make_literaltab_output(char* filename)**

pass1 작업에서 만들어진 token들을 활용하여 output_littab.txt를 만든다.

```
void make_literaltab_output(char* filename)
{
    /* add your code here */
    FILE* fp;
    int i = 0;

    fopen_s(&fp, filename, "w");//open file
    if (fp == NULL) {//NULL exception
        printf("Failure");
        return;
    }
    for (i = 0; i < literal_index; i++) {
        fprintf(fp, "%s\t%04x\n", literal_table[i].literal, literal_table[i].addr);
        printf("%s\t%04x\n", literal_table[i].literal, literal_table[i].addr);
    }
    fclose(fp); //file close
    printf("file generating is done: %s\n\n", filename);
}
```

이 다음부터 나오는 함수는 pass2에서 사용하는 함수이다.

- **split_table()**

symbol table 안에는 CSECT마다 0으로 구분되어 한 배열 안에 저장되어 있으므로 다음과 같이 범위를 나눠준다.

```
void split_table() {
    int i = 0, n = 0;

    sym_len[0] = 2;
    for (i = 1; i < MAX_SEC; i++) {
        sym_len[i] = 0;
    }
    for (i = 2; i < sym_index; i++) {
        if (sym_table[i].addr != 0) {//CSECT일때 symbol table의 주소값이 0으로 바뀐다.
            sym_len[n]++;
        }
        else {
            sym_len[++n]++;
        }
    }
}
```

- **search_sym(char* s, int mode)**

symbol table 안에는 함수 3개의 symbol들이 모두 저장되어 있다. 함수별로 검색 범위를 정해 검색한다.

```
int search_sym(char* s, int mode) {//mode : file 범위

    int i = 0, start = 0, end = 0;

    if (mode == 0) {//file 0번 검색 범위
        start = 0;
        end = sym_len[0];
    }
    else if (mode == 1) {//file 1번 검색 범위
        start = sym_len[0];
        end = start + sym_len[1];
    }
    else if (mode == 2) {//file 2번 검색 범위
        start = sym_len[0] + sym_len[1];
        end = start + sym_len[2];
    }

    for (i = start; i < end; i++) {//위에서 정한 범위 만큼만 검색
        if (strcmp(&sym_table[i].symbol, s) == 0) {
```

```
            return sym_table[i].addr;
        }
    }
    return -1;
}
```

- **search_lit(char* s)**

literal table도 위와 같이 symbol table처럼 관리해야 하지만 이번 프로젝트에서는 literal의 주소값이 겹치지 않으므로 생략하였다.

```
int search_lit(char* s) {
    for (int i = 0; i < literal_index; i++) {
        if (strcmp(literal_table[i].literal, s) == 0) {
            return literal_table[i].addr;
        }
    }
    return -1;
}
```

- **reg_num(char c)**

reg_num함수는 2형식 처리를 위해 register 번호를 알아야하기 때문에 이 함수를 통해서 register 번호를 반환해준다.

```
int reg_num(char c) {
    switch (c){
    case 'A': return 0; break;
    case 'X': return 1; break;
    case 'B': return 3; break;
    case 'S': return 4; break;
    case 'T': return 5; break;
    case 'F': return 6; break;
    default: return 0;  break;
    }
}
```

- **assem_pass2()**

pass2 작업을 수행한다. token구조체에 정보를 읽어와 nixbpe값을 만들고 완성된 token구조체를 사용하여 object_code 구조체의 정보를 채운다.

token_table에서 한 줄씩 읽고 operand의 @,# 정보로 n, i 값을 정하고, x값을 정한다. operator의 + 가 붙어있는지 여부로 e 값을 정하고 pass1에서 저장한 token_table_addr 배열을 보고 TA-PC값을 계산하여 b,p 값을 정한다. 그 후 shift 연산을 통해 24bit objectcode를 구조체에 저장한다. (format4일 경우에는 32bit) TA가 없는 경우와 literal 처리, header 선언 등을 위해 strcmp를 활용하여 문자열을 비교하여 해당 작업을 수행하였다. 반복문은 token_line 수만큼 반복하게 된다.

REDREC를 처리할 때 BUFFER-BUFFEND의 값을 알 수 없으므로 000000을 적어야하나 0을 shift하여

도 그대로 0이므로 길이를 알 수 없다. 따라서 1000000을 넣어서 조건문으로 처리하였다.

```
static int assem_pass2(void)
{
    /* add your code here */
    int ta = 0, pc = 0;// target address and program counter
    int index = 0;
    char* temp = NULL;
    int format = 0;//format 구분
    int last = 0;//last 3byte or 5byte
    int file_num = 0;//file 3개 0 : main 1 : REDREC 2: WEREC
    int reg1 = 0, reg2 = 0;//format 2
    int op = 0;//opcode
    int sec = 0;//control section literal processing variable
    int sec_first = 0;// control section literal processing variable (arragne's first value)
    int m_index[MAX_SEC] = { 0, };//modified용

    /*file별 검색 범위 정하기*/
    split_table();

    for (int i = 0; i < token_line; i++) {
        code[file_num].t_code[sec_len[file_num]] = 0;
        code[file_num].t_addr[sec_len[file_num]] = token_table_addr[i];
        op = -1;
        format = -1;// literal_flag = 0;
        ta = 0; pc = 0;
        if (token_table[i]->operator != NULL) {
            //HEADER저장
            if (strcmp("START", token_table[i]->operator) == 0) {
                strcpy((code[file_num].p_name), token_table[i]->label);
                sec_len[file_num]++;
                continue;
            }
            if (strcmp("EXTDEF", token_table[i]->operator) == 0) {
                for (int j = 0; j < 3; j++) {
                    if (token_table[i]->operand[j] != NULL) {
                        strcpy(code[file_num].d_name[j], token_table[i]->operand[j]);
                        code[file_num].d_addr[j]
                            = search_sym(token_table[i]->operand[j], file_num);
                    }
                }
                sec_len[file_num]++;
                continue;
            }
            if (strcmp("EXTREF", token_table[i]->operator) == 0) {
                for (int j = 0; j < 3; j++) {
```

```
                if (token_table[i]->operand[j] != NULL) {
                    strcpy(code[file_num].r_name[j], token_table[i]->operand[j]);
                }
            }
            sec_len[file_num]++;
            continue;
        }
        if (strcmp("CSECT", token_table[i]->operator) == 0) {
            file_num++;
            strcpy((code[file_num].p_name), token_table[i]->label);
        }
    }


    //n, i value setting
    token_table[i]->nixbpe = 0B00000000;
    if (token_table[i]->operand[0] != NULL) {
        if (token_table[i]->operand[0][0] == '@') {
            token_table[i]->nixbpe += 0B00100000;//0010_0000
        }
        else if (token_table[i]->operand[0][0] == '#') {
            token_table[i]->nixbpe += 0B00010000;//0001_0000;
        }
        else {
            token_table[i]->nixbpe += 0B00110000;//0011_0000
        }
    }
    //x setting
    if (token_table[i]->operand[1] != NULL && (token_table[i]->operand[1][0] == 'X')) {
        token_table[i]->nixbpe += 0B00001000;//00_1000
    }

    //e setting
    if (token_table[i]->operator != NULL && (token_table[i]->operator[0] == '+')) {
        token_table[i]->nixbpe += 0B00000001;//0000_0001
    }

    //bp setting and last
    if (token_table[i]->operator != NULL) {
        temp = token_table[i]->operator;
        if (token_table[i]->operator[0] == '+') {
            temp++;
            format = 4;
            index = search_opcode(temp);
            if (index != -1) {//if it doesn't exist, return -1;
```

```
                op = inst_table[index]->op;
            }
        }
        else {
            index = search_opcode(temp);
            if (index != -1) {//if it doesn't exist, return -1;
                format = inst_table[index]->format;
                op = inst_table[index]->op;
            }
        }
    }
    code[file_num].t_format[sec_len[file_num]] = format;

    //foramt 2 3 4 and others condition
    if (format == -1) {

        /*continue 조건*/
        if (token_table[i]->operator == NULL) continue;
        if ((strcmp(token_table[i]->operator,"EQU") == 0)) continue;

        // END LTORG SETTING
        if ( (strcmp(token_table[i]->operator,"LTORG") == 0)
                || (strcmp(token_table[i]->operator,"END") == 0)) {
            char* temp2;
            temp2 = literal_table[sec_first].literal + 1;
            if (*temp2 == 'X') {
                temp2 += 2;
                code[file_num].t_code[sec_len[file_num]] += strtol(temp2, NULL, 16);
                sec_first++;
            }
            else if (*temp2 == 'C') {
                temp2 += 2;
                sec_first++;
                while (*temp2 != '\'') {
                    code[file_num].t_code[sec_len[file_num]] += *temp2;
                    code[file_num].t_code[sec_len[file_num]]
                                = code[file_num].t_code[sec_len[file_num]] << 8;
                    temp2++;
                }
                //while loop 로직 상 뒤로 한번 shift해줘야함
                code[file_num].t_code[sec_len[file_num]]
                    = code[file_num].t_code[sec_len[file_num]] >> 8;
            }
        }
```

```
            if ((strcmp(token_table[i]->operator,"WORD") == 0)
                        || strcmp(token_table[i]->operator,"BYTE") == 0) {

                if (token_table[i]->operand[0] == NULL) continue;

                /*\' 처리*/
                char* temp2 = token_table[i]->operand[0];
                while (*temp2 != '\'') temp2++;//\' 찾기
                if (temp2 == NULL) {//\' 없을 대
                    code[file_num].t_code[sec_len[file_num]] = 0;
                }
                else {
                    if (token_table[i]->operand[0][0] == 'X') {
                        temp2++;
                        code[file_num].t_code[sec_len[file_num]]
                                                += strtol(temp2, NULL, 16);//16진수 변환
                    }
                    else if (token_table[i]->operand[0][0] == 'C') {
                        temp2++;
                        while (*temp2 != '\'') {
                            code[file_num].t_code[sec_len[file_num]] += *temp2;
                            code[file_num].t_code[sec_len[file_num]]
                                        = code[file_num].t_code[sec_len[file_num]] << 8;
                            temp2++;
                        }
                        code[file_num].t_code[sec_len[file_num]]
                                = code[file_num].t_code[sec_len[file_num]] >> 8;
                    }
                    else {//BUFFEDN-BUFFER 처리
                        code[file_num].t_code[sec_len[file_num]] = 0x1000000;
                        ta = search_sym(token_table[i]->operand[0], file_num);
                        if (ta == -1) {
                            char symbol1[20] = { 0 }, symbol2[20] = { 0 };
                            char operator = 0;
                            char* operand = token_table[i]->operand[0];
                            char* op_pos = NULL;

                            // 연산자 위치 탐색
                            op_pos = strchr(operand, '+');
                            if (op_pos) operator = '+';
                            else {
                                op_pos = strchr(operand, '-');
                                if (op_pos) operator = '-';
                            }
```

```
                        if (op_pos == NULL) continue;

                        // 연산자 앞 부분 -> symbol1
                        strncpy(symbol1, operand, op_pos - operand);
                        symbol1[op_pos - operand] = '\0';

                        // 연산자 뒤 부분 -> symbol2
                        strcpy(symbol2, op_pos + 1);
                        if (search_sym(symbol1, file_num) == -1) {
                            code[file_num].m_addr[m_index[file_num]] = token_table_addr[i];
                            code[file_num].m_length[m_index[file_num]] = 6;
                            char sign[10];
                            sprintf(sign, "+%s", symbol1);
                            code[file_num].m_name[m_index[file_num]++] = _strdup(sign);
                        }
                        if ((search_sym(symbol2, file_num) == -1)) {
                            code[file_num].m_addr[m_index[file_num]] = token_table_addr[i];
                            code[file_num].m_length[m_index[file_num]] = 6;
                            char sign[10];
                            sprintf(sign, "%c%s", operator, symbol2);
                            code[file_num].m_name[m_index[file_num]++] = _strdup(sign);
                        }
                    }
                }
            }
        }
        sec_len[file_num]++;
        continue;
    }
    else {//format != -1 일 때
        if (format == 2) {//format2 처리
            reg1 = 0; reg2 = 0;
            if (token_table[i]->operand[0] != NULL) {
                reg1 = reg_num(token_table[i]->operand[0][0]);
            }
            if (token_table[i]->operand[1] != NULL) {
                reg2 = reg_num(token_table[i]->operand[1][0]);
            }
            code[file_num].t_code[sec_len[file_num]] = (op << 8);
            last = (reg1 << 4) + (reg2);
            code[file_num].t_code[sec_len[file_num]] += last;
            sec_len[file_num]++;
            continue;//format3와 format4와 다름
        }
        else if (format == 3) {
```

```
            if (token_table[i]->operand[0] != NULL) {
                ta = search_sym(token_table[i]->operand[0], file_num);//target address
                if (token_table[i]->operand[0][0] == '@') {//@ 때고 찾기
                    ta = search_sym(token_table[i]->operand[0] + 1, file_num);
                }
                if (token_table[i]->operand[0][0] == '=') {//= literal table에서 찾기
                    ta = search_lit(token_table[i]->operand[0]);
                }
                if (ta == -1) {//target address 모를 때 modified 처리해주기
                    last = 0;
                    if (token_table[i]->operand[0][0] == '#') {
                        last = atoi(token_table[i]->operand[0] + 1);
                    }
                    else {
                        code[file_num].m_addr[m_index[file_num]] = token_table_addr[i] + 1;
                        code[file_num].m_length[m_index[file_num]] = 5;
                        char sign[10];
                        sprintf(sign, "+%s", token_table[i]->operand[0]);
                        code[file_num].m_name[m_index[file_num]++] = _strdup(sign);
                    }
                }
                else {
                    pc = token_table_addr[i + 1];
                    last = ta - pc;
                    if (last < 2048 && last >= -2048) {//12bit arrange check
                        token_table[i]->nixbpe += 0B00000010;//0000_0010
                        last &= (0xFFF);//signed bit 오염 방지
                    }
                    else {
                        //In this project doesn't have base addressing
                        token_table[i]->nixbpe += 0B00000100;/
                    }
                }
            }
            else {//format3이면서 operand 없을 때에도 ni bit 업데이트
                token_table[i]->nixbpe += 0B00110000;
                last = 0;
            }
        }
        else if (format == 4) {
            ta = search_sym(token_table[i]->operand[0], file_num);
            if (ta == -1) {//ta없을 때 modified 처리 해주기
                last = 0;
                if (token_table[i]->operand[0][0] == '#') {
                    last = atoi(token_table[i]->operand[0] + 1);
```

```
                    }
                    else {
                        code[file_num].m_addr[m_index[file_num]] = token_table_addr[i] + 1;
                        code[file_num].m_length[m_index[file_num]] = 5;
                        char sign[10];
                        sprintf(sign, "+%s", token_table[i]->operand[0]);
                        code[file_num].m_name[m_index[file_num]++] = _strdup(sign);
                    }
                }
                else {//format 4는 target address 그대로 둘어감
                    last = ta;
                }
            }

            //format update
            code[file_num].t_format[sec_len[file_num]] = format;

            /*code 업데이트*/
            code[file_num].t_code[sec_len[file_num]] = (op << 4) + token_table[i]->nixbpe;
            if (format == 3) {
                code[file_num].t_code[sec_len[file_num]]
                        = code[file_num].t_code[sec_len[file_num]] << 12;//3*4
            }
            else if (format == 4) {
                code[file_num].t_code[sec_len[file_num]]
                        = code[file_num].t_code[sec_len[file_num]] << 20;//5*4
            }
            code[file_num].t_code[sec_len[file_num]] += last;

            sec_len[file_num]++;//length update
        }
    }
}
```

- **make_objectcode_output(char* file_name)**

pass2 작업으로 완성된 대부분의 objectcode를 읽어 파일 형식에 맞게 저장한다.

```
void make_objectcode_output(char* file_name)
{
    /* add your code here */
    FILE* fp;
    fopen_s(&fp, file_name, "w");//open file
    if (fp == NULL) {//NULL exception
        printf("Failure");
        return;
```

```
    }
    int i = 0, j = 0;

    int file_sec = 0;
    int zflag = 0;//zero flag 0 0 0 연속 나올 때 RESW와 그냥 0 구분
    int start = 0;//start address
    int t_line = 0;//start, length indexing

    int line_length = 0;
    int line_num = 0;

    /*HEADER와 DEF Line 처리*/
    for (i = 0; i < 3; i++) {
        printf("H%-6s%06x%06x\n", code[i].p_name, code[i].h_start_addr, code[i].p_length);
        fprintf(fp, "H%-6s%06x%06x\n", code[i].p_name, code[i].h_start_addr, code[i].p_length);
        if (code[i].d_name[0][0] != '\0') {
            printf("D");
            fprintf(fp,"D");
            for (int j = 0; j < 10; j++) {
                if (code[i].d_addr[j] != 0) {
                    printf("%6s%06x", code[i].d_name[j], code[i].d_addr[j]);
                    fprintf(fp, "%6s%06x", code[i].d_name[j], code[i].d_addr[j]);
                }
            }
            printf("\n");
            fprintf(fp,"\n");
        }

        /*REF Line process*/
        printf("R");
        fprintf(fp, "R");
        for (int j = 0; j < 10; j++) {
            if (code[i].r_name[j][0] != '\0') {
                printf("%-6s", code[i].r_name[j]);
                fprintf(fp,"%-6s", code[i].r_name[j]);
            }

        }
        printf("\n");
        fprintf(fp, "\n");


        /*Setting start address and length for T line */
        if (code[i].p_length <= 30) {
            code[i].t_staddr[0] = 0;
```

```
            code[i].t_length[0] = code[i].p_length;
        }
        else {
            zflag = 0;
            start = 0;
            t_line = 0;
            for (j = 0; j < sec_len[i]; j++) {

                if (code[i].t_code[j] == 0 && (code[i].t_addr[j] != 0)) {
                    zflag++;
                }
                else {
                    if (zflag > 1) {//0이었다가 값들어왔을 때 빈칸 일 때 줄 바꿈
                        code[i].t_staddr[t_line] = code[i].t_addr[j];
                        code[i].t_length[t_line] = code[i].t_addr[j + 1] - code[i].t_addr[j];
                        start = code[i].t_addr[j];
                    }
                }
                if ((code[i].t_addr[j] - start > 28) || zflag == 1) {//한 줄 길이 제한

                    code[i].t_staddr[t_line] = start;
                    code[i].t_length[t_line] = code[i].t_addr[j] - start;
                    start = code[i].t_addr[j];
                    t_line++;
                }
                if (code[i].t_code[j] == 0x1000000) {//마지막에 word 있을 때 000000 처리
                    code[i].t_staddr[t_line] = start;
                    code[i].t_length[t_line] = code[i].t_addr[j] - start +3 ;
                    start = code[i].t_addr[j];
                }
            }
        }

        /*Writing T Line*/
        line_length = 0;
        line_num = 0;
        printf("T%06x%02x", code[i].t_staddr[line_num], code[i].t_length[line_num]);
        fprintf(fp,"T%06x%02x", code[i].t_staddr[line_num], code[i].t_length[line_num]);

        for (j = 0; j < sec_len[i]; j++) {
            if (code[i].t_code[j] != 0) {
                if (line_length >= code[i].t_length[line_num]) {
                    if (code[i].t_length[line_num] != 0) {
                        printf("\n");
                        fprintf(fp,"\n");
```

```
                        line_num++;
                        line_length = 0;
                        printf("T%06x%02x",                          code[i].t_staddr[line_num],
code[i].t_length[line_num]);
                        fprintf(fp, "T%06x%02x", code[i].t_staddr[line_num],
                            code[i].t_length[line_num]);
                    }

                }
                if (code[i].t_format[j] == 2) {
                    printf("%04x", code[i].t_code[j]);
                    fprintf(fp,"%04x", code[i].t_code[j]);
                    line_length += 2;
                }
                else if (code[i].t_format[j] == 3) {
                    printf("%06x", code[i].t_code[j]);
                    fprintf(fp,"%06x", code[i].t_code[j]);
                    line_length += 3;
                }
                else if (code[i].t_format[j] == 4) {
                    printf("%08x", code[i].t_code[j]);
                    fprintf(fp,"%08x", code[i].t_code[j]);
                    line_length += 4;
                }
                else {
                    if (code[i].t_code[j] == 0x1000000) {
                        printf("%06x", 0);
                        fprintf(fp,"%06x", 0);
                    }
                    else {
                        printf("%02x", code[i].t_code[j]);
                        fprintf(fp,"%02x", code[i].t_code[j]);
                    }
                }
            }
        }
        printf("\n");
        fprintf(fp,"\n");

        /*modulation line 처리*/
        for (int j = 0; j < 5; j++) {
            if (code[i].m_name[j] == NULL) continue;
            if (code[i].m_name[j][0] != '\0') {
                printf("M%06x%02x%-6s \n", code[i].m_addr[j],
                        code[i].m_length[j], code[i].m_name[j]);
```

```
                fprintf(fp,"M%06x%02x%-6s \n", code[i].m_addr[j],
                        code[i].m_length[j], code[i].m_name[j]);
            }

        }

        /*end line 처리*/
        if (i == 0) {
            printf("E000000");
            fprintf(fp,"E000000");
        }
        else {
            printf("E");
            fprintf(fp,"E");
        }
        printf("\n\n");
        fprintf(fp,"\n\n");
    }

    printf("file generating is done: %s\n\n", file_name);
}
```

- **assembler.h**

assembler의 header파일이다. 프로그램에 필요한 정의, 구조체, 함수들이 선언되어있다.

```
#define MAX_INST 256
#define MAX_LINES 5000

#define MAX_COLUMNS 4
#define MAX_OPERAND 3

/*새로 추가한 define*/
#define MAX_SEC 3 // file section 3개 MAX 추가 가능

 /*
 * instruction 목록을 저장하는 구조체이다.
 * instruction 목록 파일로부터 정보를 받아와서 생성한다.
 * instruction 목록 파일에는 라인별로 하나의 instruction을 저장한다.
 *
 */
typedef struct _inst
{
    char str[10];//name
    unsigned char op;//opcode
    int format;//formant
    int ops;//number of oprend
```

```
} 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;
/*
 * 어셈블리 할 소스코드를 토큰으로 변환하여 저장하는 구조체 변수이다.
 * operator 변수명은 renaming을 허용한다.
 */
typedef struct _token {
    char* label;
    char* operator;
    char* operand[MAX_OPERAND];
    char comment[100];
    char nixbpe;
} 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];
static int sym_index;

literal literal_table[MAX_LINES];
static int literal_index;

/**
 * 오브젝트 코드 전체에 대한 정보를 담는 구조체이다.
 * Header Record, Define Recode,
 * Modification Record 정보가 들어있다.
 */
typedef struct _object_code {
     // H' line
    char p_name[7];//program name
    int h_start_addr;//start adress
    int p_length;//program length

    //D' line
    char d_name[10][7];//10개??
    int d_addr[10];

    //R' line
    char r_name[10][7];

    //T' line
    int t_staddr[MAX_LINES];//each line  start address
    int t_addr[MAX_LINES];//each addr
    int t_length[MAX_LINES]; //each line length
    int t_code[MAX_LINES];//code
    int t_format[MAX_LINES];

    // M' line
    int m_addr[MAX_LINES];
    int m_length[MAX_LINES];
    char* m_name[MAX_LINES];
    char m_sign[MAX_LINES];

} object_code;
object_code code[MAX_SEC];

static int locctr;
```

```c
static char* input_file;
static char* output_file;

/*새로 추가한 변수*/
static int token_table_addr[MAX_LINES];//보고서에 적어야함 TA - PC를 위해서
static int sym_len[MAX_SEC];//symbol 검색 범위
static int sec_len[MAX_SEC] = { 0, };//section별 length

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);

/*새로 추가한 함수*/
int split_tap(char* data, char* arr[]);// tap 제거
int split_comma(char* data, char* arr[]);// comma 제거
int literal_check(char* temp);// literal 중복 체크
int search_opcode(char* str);//operator -> opcode로 반환

void split_table();//symbol table split
int reg_num(char c);//register table
int search_sym(char* s, int mode);//symbol search return symbol address
int search_lit(char* s);//literal search return address
```