숭실대학교 Soongsil University

시스템프로그래밍 2021 보고서

보고서 제출서약서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
   1.1. 나는 동료의 보고서를 베끼지 않았습니다.
   1.2. 나는 비공식적으로 얻은 해답/해설을 기초로 보고서를 작성하지 않았습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다. (나는 특히 인터넷에서 다운로드한 내용을 보고서에 거의 그대로 복사하여 사용하지 않았습니다.)
3. 나는 보고서를 제출하기 전에 동료에게 보여주지 않았습니다.
4. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

과목	시스템프로그래밍 2021
과제명	프로젝트1
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20192698 심원준(출석번호 124)
제출일	2021년 5월 5일

차 례

1장. 프로젝트 목적/동기

이 프로젝트는 SIC/XE머신의 어셈블러를 구현하여 입력된 SIC/XE코드를 Object Program Code로 바꾸는 것을 목적으로 한다.

2장. 설계/구현 아이디어

지난번 파서 과제에 작성한 pass1함수를 이용하여 각 라인의 주소를 할당하고, 리터럴 테이블을 만들어 출력할 수 있도록 하였다. 이를 위해서 token_table에 nixbpe 변수가 추가되었으며 locctr을 이용하여 각 명령어 별로 주소를 할당해 주고, 증가시킬 수 있도록 하였다. 이를 위하여 token_unit구조체에 addr int형 변수를 추가하여 각 token별로 주소값을 저장할 수 있도록 하였다. 또한 csect별로 구분해주고자 section변수를 추가해주었다. 주소 할당이 끝난 뒤, nixbpe를 정해주었는데, indirect addressing/immediate addressing/4형식 여부를 operator와 operand를 통하여 확인한 뒤, 비트연산을 이용하여 nixbpe값을 할당해 주었다. 그 뒤에는 리터럴 테이블을 구성해 주었는데, 이를 위하여 리터럴인지 여부를 파악할 수 있도록 하는 is_literal bool형 변수와 literal값을 저장할 수 있는 literal 변수 또한 token_unit 에 추가해주어 활용하였다. 그리고 이 값을 literal_unit에 저장하여, 리터럴 테이블을 구성해주었다. literal_unit에도 또한 section값을 넣을 수 있는 변수를 추가해주어, 추후 pass2과정에서 objectcode를 계산할 때에 section내에 포함되어있는지 여부를 파악할 수 있도록 해주고자 하였다.

이 밖에도 symbol의 개수, literal의 개수를 파악하여 출력시에 활용하고자 symbol_count변수와 litcount변수를 추가해주었으며, objectcode를 출력할 시에 프로그램 길이를 출력해 줄 수 있도록 proglength배열(control section별 길이 저장)과 sttadd(starting address)변수를 추가해주었고 control section을 구분하고자 eachsection배열을 만들어 주어 각 섹션의 시작 토큰번호를 저장해 주었다.

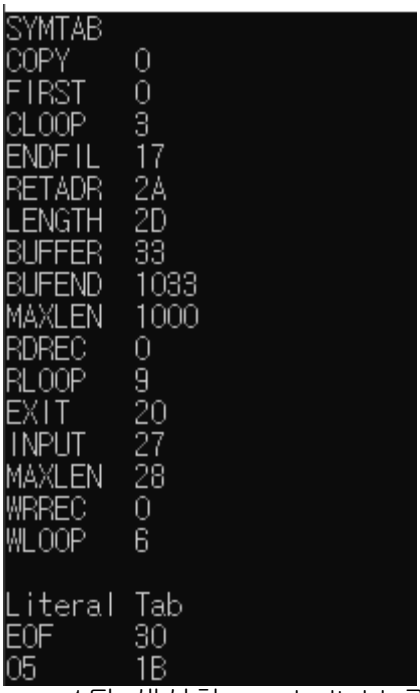
Pass2에서는 pass1에서 저장한 nixbpe값과 address값을 이용하여 object code값을 계산해 주고자 하였는데, 이 계산한 값을 저장할 수 있도록 objectcode unsigned char형 배열을 만들어주었다.(unsigned char의 저장 범위 때문에 배열로 나누어 저장함). 그 구성은 다음과 같다.

```
token_table[i]->objectcode[0] = (unsigned char)255;//op+ni부분
        token_table[i]->objectcode[1] = (unsigned char)255;//xbpe부분
        token_table[i]->objectcode[2] = (unsigned char)255;//3형식의 경우 나머지 세자리 4형식의경우 3자리
        token_table[i]->objectcode[3] = (unsigned char)255;//4형식의 경우 나머지 두자리
```

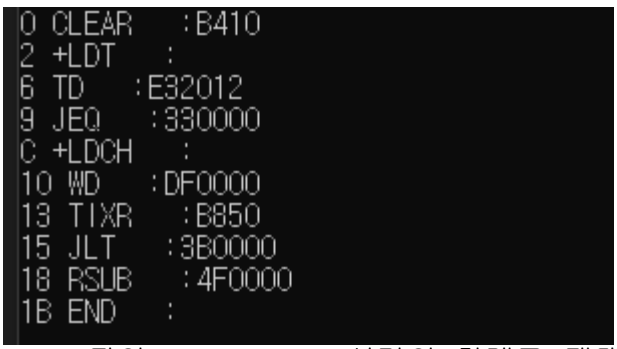
먼저 토큰에 operator가 존재하는지 여부를 파악한 뒤, search_opcode를 이용해 얻은 opcode를 가지고 해당 명령어의 형식을 얻어 명령어의 형식에 따라 분기하여 계산해주었다. 1형식의 경우 opcode를 그대로 objectcode로 저장해주었으며, 2형식의 경우 operand의 개수를 알아 낸 뒤, opcode+r1+r2(operand가 2개인 경우)의 구성으로 objectcode를 저장해주었다. 3형식과 4형식의 경우 indirect/immediate/simple addressing여부를 비트연산을 통해 알아낸 뒤, symbol을 검색하여 section내의 포함여부등을 바탕으로 objectcode를 저장하도록 하였다.

이 objectcode를 출력하기 위한 함수를 구현하여야 했으나 교내 코로나 확진자 발생으로 인해 시험이 미뤄지게 되면서 시험기간이 겹치는등 시간상의 이유로 제대로된 함수를 모두 구현하지 못하였고, token의 순서대로 그 object code만을 단순 print하도록 할 수 밖에 없었다.

3장 수행결과(구현 화면 포함)



pass1뒤 생성한 symboltable과 literal table



pass2뒤의 objectcode. 시간의 한계로 제대로된 출력함수를 모두 구현하지 못하고 주석처리 해두었으며, 단순 출력과 디버깅을 한 결과 objectcode가 계산되었으나 출력방법의 문제로 뒤의 FEF등이 표시되지 않는 것을 확인하였다.

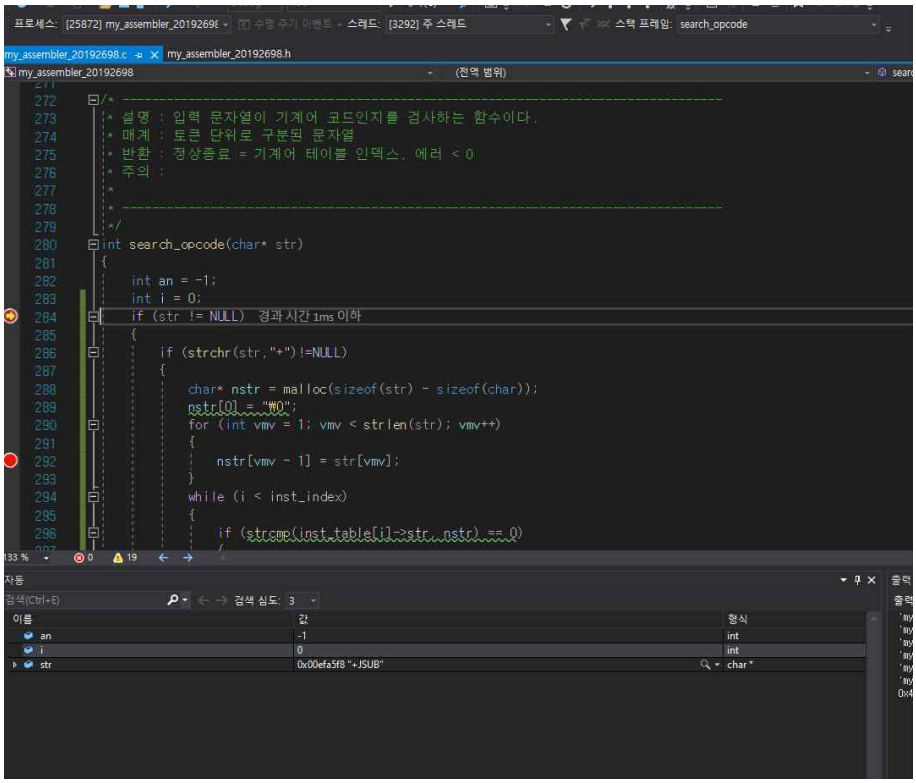
object코드를 양식에 맞춰 출력하는 함수를 실행하면 초기에 첫 섹션의 출력이 잘 이루어지던 중, 무한루프에 빠지는 현상이 발생하였으나 해결하지 못하였다.

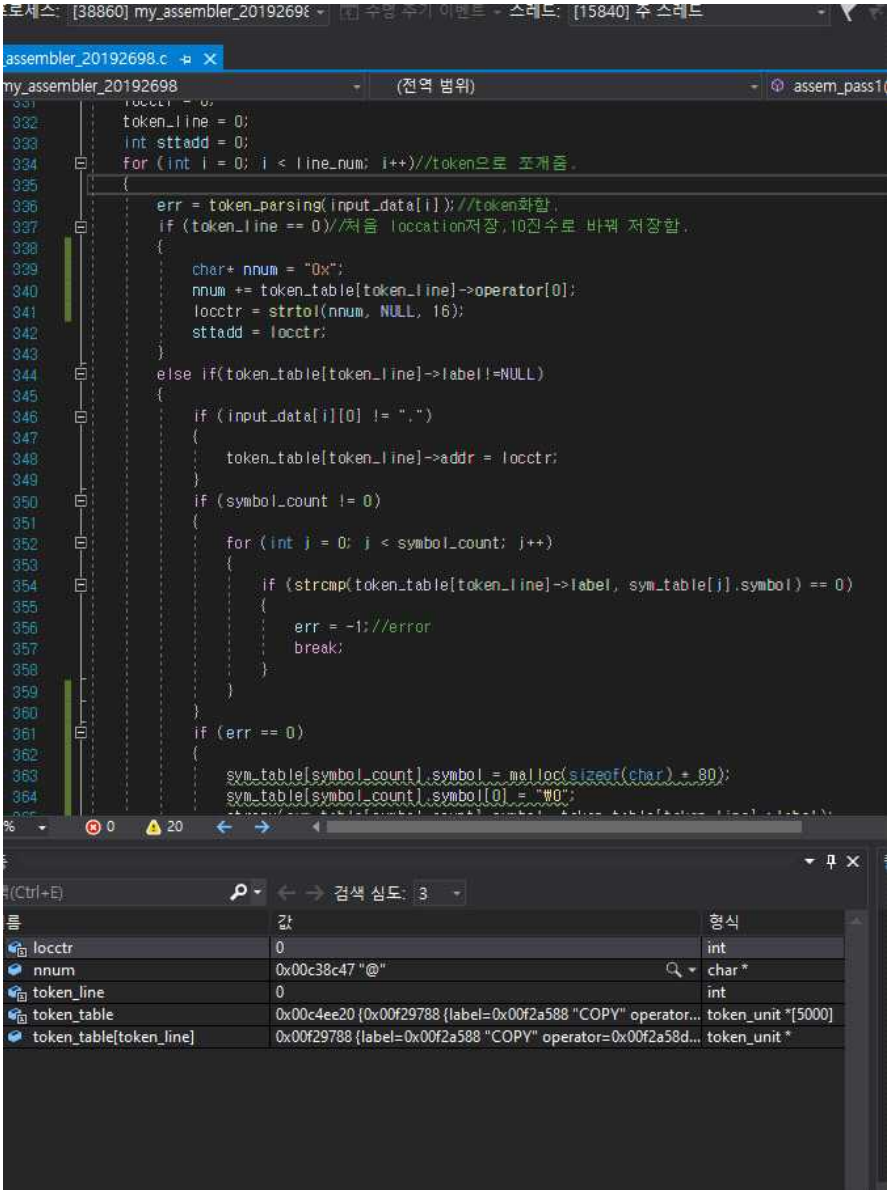
4장 결론 및 보충할 점

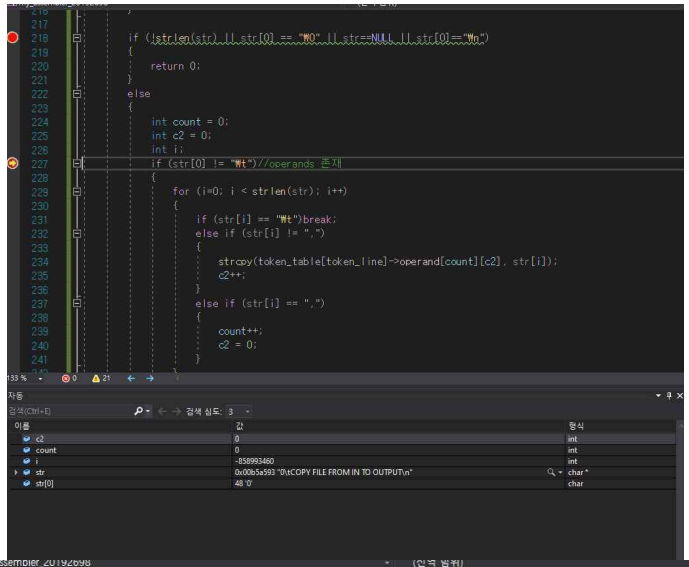
계산된 objectcode를 제대로 출력하지 못하였다. 시간의 한계도 있었으나 그 출력과정에 있어 Modification code의 시작위치, 길이등을 계산하는 것에 어려움을 겪었던 것이 영향을 주었다. 무한루프에 빠지는 현상의 해결책을 찾아야한다. 또한 분기문의 거듭된 사용으로 코드가 길어져 오류를 해결함에 어려움을 겪었는데, 코드를 줄이면서 성능은 향상시킬 수 있는 방법을 찾음이 필요해보인다.

5장 디버깅

문자열을 처리함에 있어 갑자기 변형되어버리는 오류가 발생하여 이를 해결하기 위해 디버깅을 실시하였다.

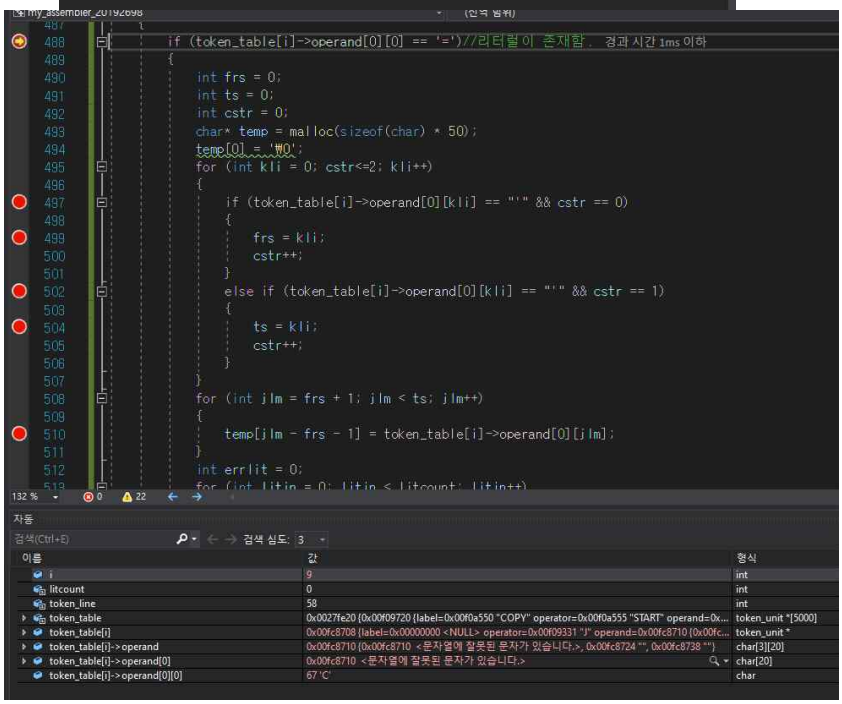



프로세스: [38860] my_assembler_20192698
스레드: [15840] 주 스레드
assembler_20192698.c
my_assembler_20192698
(전역 범위)
assem_pass1
token_line = 0;
int sttadd = 0;
for (int i = 0; i < line_num; i++)//token으로 쪼개줌.
{
err = token_parsing(input_data[i]);//token화함.
if (token_line == 0)//처음 location저장,10진수로 바꿔 저장함.
{
char* nnum = "0x";
nnum += token_table[token_line]->operator[0];
locctr = strtol(nnum, NULL, 16);
sttadd = locctr;
}
else if(token_table[token_line]->label!=NULL)
{
if (input_data[i][0] != ".")
{
token_table[token_line]->addr = locctr;
}
if (symbol_count != 0)
{
for (int j = 0; j < symbol_count; j++)
{
if (strcmp(token_table[token_line]->label, sym_table[j].symbol) == 0)
{
err = -1;//error
break;
}
}
}
if (err == 0)
{
sym_table[symbol_count].symbol = malloc(sizeof(char) * 80);
sym_table[symbol_count].symbol[0] = "₩0";
0
20
검색 심도: 3
이름
값
형식
locctr
0
int
nnum
0x00c38c47 "@"
char *
token_line
0
int
token_table
0x00c4ee20 {0x00f29788 {label=0x00f2a588 "COPY" operator...
token_unit *[5000]
token_table[token_line]
0x00f29788 {label=0x00f2a588 "COPY" operator=0x00f2a58d...
token_unit *



```
if (!strlen(str) || str[0] == '₩0' || str==NULL || str[0]=='₩n')
{
    return 0;
}
else
{
    int count = 0;
    int c2 = 0;
    int i;
    if (str[0] != '₩t')//operands 존재
    {
        for (i=0; i < strlen(str); i++)
        {
            if (str[i] == '₩t')break;
            else if (str[i] != ',')
            {
                strcpy(token_table[token_line]->operand[count][c2], str[i]);
                c2++;
            }
            else if (str[i] == ',')
            {
                count++;
                c2 = 0;
            }
```

이름	값	형식
c2	0	int
count	0	int
i	-858993460	int
str	0x00b5a593 "0\tCOPY FILE FROM IN TO OUTPUT\n"	char *
str[0]	48 '0'	char



```
if (token_table[i]->operand[0][0] == '=')//리터럴이 존재함. 경과 시간 1ms 이하
{
    int frs = 0;
    int ts = 0;
    int cstr = 0;
    char* temp = malloc(sizeof(char) * 50);
    temp[0] = '₩0';
    for (int kli = 0; cstr<=2; kli++)
    {
        if (token_table[i]->operand[0][kli] == "'" && cstr == 0)
        {
            frs = kli;
            cstr++;
        }
        else if (token_table[i]->operand[0][kli] == "'" && cstr == 1)
        {
            ts = kli;
            cstr++;
        }
    }
    for (int jlm = frs + 1; jlm < ts; jlm++)
    {
        temp[jlm - frs - 1] = token_table[i]->operand[0][jlm];
    }
    int errlit = 0;
```

이름	값	형식
i	9	int
litcount	0	int
token_line	58	int
token_table	0x0027fe20 {0x00f09720 {label=0x00f0a550 "COPY" operator=0x00f0a555 "START" operand=0x...	token_unit *[5000]
token_table[i]	0x00fc8708 {label=0x00000000 <NULL> operator=0x00f09331 "J" operand=0x00fc8710 {0x00fc...	token_unit *
token_table[i]->operand	0x00fc8710 {0x00fc8710 <문자열에 잘못된 문자가 있습니다.>, 0x00fc8724 "", 0x00fc8738 ""}	char[3][20]
token_table[i]->operand[0]	0x00fc8710 <문자열에 잘못된 문자가 있습니다.>	char[20]
token_table[i]->operand[0][0]	67 'C'	char

6장 소스코드(+주석)

```
/*
 * 화일명 : my_assembler_20192698.c
 * 설  명 : 이 프로그램은 SIC/XE 머신을 위한 간단한 Assembler 프로그램의 메인루틴으로,
 * 입력된 파일의 코드 중, 명령어에 해당하는 OPCODE를 찾아 출력한다.
 * 파일 내에서 사용되는 문자열 "00000000"에는 자신의 학번을 기입한다.
 */

 /*
  *
  * 프로그램의 헤더를 정의한다.
  *
  */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <math.h>
#include <stdbool.h>
  // 파일명의 "00000000"은 자신의 학번으로 변경할 것.
#include "my_assembler_20192698.h"

/                                                                *
------------------------------------------------------------------------------------------
 * 설명 : 사용자로 부터 어셈블리 파일을 받아서 명령어의 OPCODE를 찾아 출력한다.
 * 매계 : 실행 파일, 어셈블리 파일
 * 반환 : 성공 = 0, 실패 = < 0
 * 주의 : 현재 어셈블리 프로그램의 리스트 파일을 생성하는 루틴은 만들지 않았다.
 *                 또한 중간파일을 생성하지 않는다.
 *
------------------------------------------------------------------------------------------
 */
int main(int args, char* arg[])
{
    if (init_my_assembler() < 0)
    {
        printf("init_my_assembler: 프로그램 초기화에 실패 했습니다.₩n");
        return -1;
    }
    if (assem_pass1() < 0)
    {
        printf("assem_pass1:  패스1  과정에서 실패하였습니다.  ₩n");
        return -1;
    }
    make_symtab_output("symtab_20192698");


make_literaltab_output("literaltab_20192698");


    if (assem_pass2() < 0)
    {
        printf(" assem_pass2: 패스2 과정에서 실패하였습니다.  ₩n");
        return -1;
    }
```

```
make_objectcode_output("output_20192698");

    return 0;

}

/                                                                *
------------------------------------
------------------------------------
----------
 * 설명 : 프로그램 초기화를 위한 자료구조 생성 및 파일을 읽는 함수이다.
 * 매계 : 없음
 * 반환 : 정상종료 = 0 , 에러 발생 = -1
 * 주의 : 각각의 명령어 테이블을 내부에 선언하지 않고 관리를 용이하게 하기
 *                위해서 파일 단위로 관리하여 프로그램 초기화를 통해 정보를 읽어 올 수 있도록
 *                구현하였다.
 *
------------------------------------
------------------------------------
----------
 */
int init_my_assembler(void)
{
    int result;

    if ((result = init_inst_file("inst.data")) < 0)
        return -1;
    if ((result = init_input_file("input.txt")) < 0)
        return -1;
    return result;
}

/                                                                *
------------------------------------
------------------------------------
----------
 * 설명 : 머신을 위한 기계 코드목록 파일을 읽어 기계어 목록 테이블(inst_table)을
 *        생성하는 함수이다.
 * 매계 : 기계어 목록 파일
 * 반환 : 정상종료 = 0 , 에러 < 0
 * 주의 : 기계어 목록파일 형식은 자유롭게 구현한다. 예시는 다음과 같다.
 *
 *
==================================
==================================
=======
 *                | 이름 | 형식 | 기계어 코드 | 오퍼랜드의 갯수 | NULL|
 *
==================================
==================================
=======
 *
 *
------------------------------------
------------------------------------
----------
 */
int init_inst_file(char* inst_file)
{
    FILE* inst_file_to_read = fopen(inst_file, "r");//읽어야할inst.data파일

    int errno;
    int i = 0;
    errno = 0;//오류 잡기용
    if (inst_file_to_read == NULL)errno = -1;
    int dic[100];//딕셔너리에 16진수 저장해둠.
    dic['0'] = 0;
    dic['1'] = 1;
    dic['2'] = 2;
    dic['3'] = 3;
```

```
    dic['4'] = 4;
    dic['5'] = 5;
    dic['6'] = 6;
    dic['7'] = 7;
    dic['8'] = 8;
    dic['9'] = 9;
    dic['A'] = 10;
    dic['B'] = 11;
    dic['C'] = 12;
    dic['D'] = 13;
    dic['E'] = 14;
    dic['F'] = 15;
    char st[15];
    char o1, o2;
    int b;
    int d;
    inst_index = 0;
    while (!feof(inst_file_to_read)) {
        inst_table[i] = malloc(sizeof(struct
inst_unit));//저장공간 할당
        inst_table[i]->str[0] = "₩0";
        inst_table[i]->op =
malloc(sizeof(unsigned char));
        inst_table[i]->format = 0;
        inst_table[i]->ops = 0;
        fscanf(inst_file_to_read, "%s %d %c
%c %d", st, &b, &o1, &o2, &d);// inst로부터
입력받음
        strcpy(inst_table[i]->str, st);// 변수에
저장
        inst_table[i]->format = b;
        inst_table[i]->op = (unsigned
char)(dic[o1] * 16 + dic[o2]);
        inst_table[i]->ops = d;
        i++;
        inst_index++;
    }
    fclose(inst_file_to_read);// 파일닫기
    return errno;
}
/                                                *
-----------------------------------
-----------------------------------
----------
 * 설명 : 어셈블리 할 소스코드를 읽어 소스코
드 테이블(input_data)를 생성하는 함수이다.
 * 매계 : 어셈블리할 소스파일명
 * 반환 : 정상종료 = 0 , 에러 < 0
 * 주의 : 라인단위로 저장한다.
 *
 *
-----------------------------------
-----------------------------------
----------
 */
int init_input_file(char* input_file)
{
    int errno;
    errno = 0;//에러 파악
    FILE* file = fopen(input_file, "r");
    for (int i = 0; i < MAX_LINES; i++)
    {
        input_data[i] = malloc(sizeof(char) *
100);
        input_data[i][0] = "₩0";
    }
    if (file == NULL)errno = -1;
    line_num = 0;
    while (!feof(file))
    {
        fgets(input_data[line_num], 100, file);
        line_num++;
    }
    fclose(file);

    return errno;
}
```

```
/                                                                  *
------------------------------------------------------------------------------------------
 * 설명 : 소스 코드를 읽어와 토큰단위로 분석하고 토큰 테이블을 작성하는 함수이다.
 *        패스 1로 부터 호출된다.
 * 매계 : 파싱을 원하는 문자열
 * 반환 : 정상종료 = 0 , 에러 < 0
 * 주의 : my_assembler 프로그램에서는 라인단위로 토큰 및 오브젝트 관리를 하고 있다.
 *
------------------------------------------------------------------------------------------
 */
int token_parsing(char* str)
{
    if (str != NULL || str[0] != "₩0" || str[0] != "₩n")
    {
        token_table[token_line] = malloc(sizeof(struct token_unit));//토큰구조체 배열 초기화
        token_table[token_line]->label = malloc(sizeof(char) * 300);

token_table[token_line]->operator=malloc(sizeof(char) * 300);
        token_table[token_line]->label[0] = '₩0';
        token_table[token_line]->operator[0] = '₩0';
        token_table[token_line]->label = NULL;

token_table[token_line]->operator=NULL;

token_table[token_line]->operand[0][0] = '₩0';

token_table[token_line]->operand[1][0] = '₩0';

token_table[token_line]->operand[2][0] = '₩0';
        token_table[token_line]->comment[0] = '₩0';
        token_table[token_line]->addr = 0;
        token_table[token_line]->nixbpe = 0;
        token_table[token_line]->literal = NULL;
        int err = 0;
        if (str == NULL)
        {
            err = -1;
        }
        char* t;

        if (str[0] == '₩t') {//label이 없을 경우
            token_table[token_line]->label = NULL;
            t = strtok_s(str, "₩t", &str);

token_table[token_line]->operator=t;
        }
        else if (strchr(str, '.'))//comment만 존재하거나 .만 찍힌 줄일 경우
        {

strcpy(token_table[token_line]->comment, str);
        }
        else {//label 이 있을경우
            t = strtok_s(str, "₩t", &str);
            token_table[token_line]->label = t;
            if (strlen(str) > 0) {
                t = strtok_s(str, "₩t", &str);
                t = strtok(t, "₩n");
```

```
token_table[token_line]->operator = t;
            }
            e          l          s          e
token_table[token_line]->operator = NULL;
        }

        if (!strlen(str) || str[0] == '₩0' || str
== NULL || str[0] == '₩n' || str[0] == '.')
        {
            return 0;
        }
        else
        {
            int count = 0;
            int c2 = 0;
            int i = 0;
            if (str[0] != "₩t")//operands 존재
            {
                while (i < strlen(str) && str[i]
!= '₩n')
                {
                    if (str[i] == '₩t')
                    {
                        break;
                    }
                    else if (str[i] != ',' &&
str[i] != '₩t')
                    {

token_table[token_line]->operand[count][c2]
= str[i];
                        c2++;
                    }
                    else    if    (str[i]    ==
','||str[i]=="₩n"||str[i]=="₩t")
                    {

token_table[token_line]->operand[count][c2]
= '₩0';
                        count++;
                        c2 = 0;
                    }
                    i++;
                }
                count = 0;
                if (str[i] != '₩n')
                {
                    for (i = i + 1; i <
strlen(str); i++)
                    {

token_table[token_line]->comment[count]    =
str[i];

                        count++;
                    }

token_table[token_line]->comment[count]    =
'₩0';
                }

            }
            else if (str[0] == "₩t")//operand
미존재
            {
                int count = 0;
                i = 2;
                for (i; i < strlen(str); i++)
                {

strcpy(token_table[token_line]->comment[coun
t], str[i]);
                    count++;
                }
            }
        }

        return err;
    }
```

```
}



/                                                *
----------------------------------
----------------------------------
----------
 * 설명 : 입력 문자열이 기계어 코드인지를 검사하는 함수이다.
 * 매계 : 토큰 단위로 구분된 문자열
 * 반환 : 정상종료 = 기계어 테이블 인덱스, 에러 < 0
 * 주의 :
 *
 *
----------------------------------
----------------------------------
----------
 */
int search_opcode(char* str)
{
    int an = -1;
    int i = 0;
    if (str != NULL)
    {
        if (strchr(str,'+')!=NULL)
        {
            char* nstr = malloc(sizeof(str) - sizeof(char));
            nstr[0] = '₩0';
            for (int vmv = 1; vmv < strlen(str); vmv++)
            {
                nstr[vmv - 1] = str[vmv];
            }
            while (i < inst_index)
            {
                if    (strcmp(inst_table[i]->str, nstr) == 0)
                {
                    an = i;
                    break;
                }
                i++;
            }
        }
        else
        {
            while (i < inst_index)
            {
                if    (strcmp(inst_table[i]->str, str) == 0)
                {
                    an = i;
                    break;
                }
                i++;
            }
        }
    }

    return an;
}



/                                                *
----------------------------------
----------------------------------
----------
* 설명 : 어셈블리 코드를 위한 패스1과정을 수행하는 함수이다.
*                 패스1에서는..
*                 1. 프로그램 소스를 스캔하여 해당하는 토큰단위로 분리하여 프로그램 라인별 토큰
*                 테이블을 생성한다.
*
* 매계 : 없음
```

```
* 반환 : 정상 종료 = 0 , 에러 = < 0
* 주의 : 현재 초기 버전에서는 에러에 대한 검사를 하지 않고 넘어간 상태이다.
*          따라서 에러에 대한 검사 루틴을 추가해야 한다.
*
*
-----------------------------------
-----------------------------------
-----------
*/
static int assem_pass1(void)
{
    int err = 0;
    locctr = 0;
    token_line = 0;
    sttadd = 0;
    int section_count = 0;
    for (int i = 0; i < line_num; i++)//token으로 쪼개줌.
    {
        err = token_parsing(input_data[i]);//token화함.
        if (token_table[token_line]->operator!=NULL)
        {
            if (strcmp(token_table[token_line]->operator,"CSECT") == 0)
            {
                progrlength[section_count] = locctr;
                progrlength[section_count] -= sttadd;
                eachsection[section_count] = token_line;
                section_count++;
                locctr = 0;//controlsection이 넘어가 초기화해줌.
                token_table[token_line]->addr = locctr;
            }
        }
        if (input_data[i][0] != '.')
        {
            token_table[token_line]->addr = locctr;
            token_table[token_line]->section = (section_count);
        }
        if (token_line == 0)//처음 loccation저장,10진수로 바꿔 저장함.
        {
            char* nnum = "0x";
            nnum += token_table[token_line]->operator[0];
            locctr = strtol(nnum, NULL, 16);
            sttadd = locctr;
            token_table[token_line]->addr = locctr;
            sym_table[symbol_count].symbol = malloc(sizeof(char) * 80);

sym_table[symbol_count].symbol[0] = "₩0";

strcpy(sym_table[symbol_count].symbol, token_table[token_line]->label);
            sym_table[symbol_count].addr = locctr;
            sym_table[symbol_count].section = 0;
            token_table[token_line]->section = 0;
            eachsection[section_count ] = token_line;
            symbol_count++;
            section_count++;
        }
        e l s e
```

```
if(token_table[token_line]->label!=NULL)
        {

            if (strcmp(token_table[token_line - 1]->operator,"EQU") == 0 && strcmp(token_table[token_line]->operator,"EQU") == 0)
            {
                token_table[token_line]->addr = atoi(token_table[token_line - 2]->operand[0]);

sym_table[symbol_count].symbol = malloc(sizeof(char) * 80);

sym_table[symbol_count].symbol[0] = "₩0";

strcpy(sym_table[symbol_count].symbol, token_table[token_line]->label);

sym_table[symbol_count].addr = atoi(token_table[token_line - 2]->operand[0]);

sym_table[symbol_count].section = section_count - 1;
                symbol_count++;
            }
            else
            {

sym_table[symbol_count].symbol = malloc(sizeof(char) * 80);

sym_table[symbol_count].symbol[0] = "₩0";

strcpy(sym_table[symbol_count].symbol, token_table[token_line]->label);

sym_table[symbol_count].addr = locctr;

sym_table[symbol_count].section = section_count - 1;
                symbol_count++;

            }

        }
        if (token_table[token_line]->operator!=NULL)
        {
            token_table[token_line]->section = (section_count);
            int retid = search_opcode(token_table[token_line]->operator);
            if (strchr(token_table[token_line]->operator,'+')!=NULL)
            {
                locctr += 4;
            }
            else if (retid != -1)
            {
                if (inst_table[retid]->format == 2)//2형식
                {
                    locctr += 2;
                }
                else if (inst_table[retid]->format == 1)//1형식
                {
                    locctr += 1;
                }
                else
                {
                    locctr += 3;
                }
            }
            else if (strcmp(token_table[token_line]->operator,"WO
```

```
RD") == 0)
            {
                locctr += 3;
            }
            else if (strcmp(token_table[token_line]->operator,"RESW") == 0)
            {
                locctr += (3 * atoi(token_table[token_line]->operand[0]));
            }
            else if (strcmp(token_table[token_line]->operator,"RESB") == 0)
            {
                locctr += atoi(token_table[token_line]->operand[0]);
            }
            else if (strcmp(token_table[token_line]->operator,"LTORG₩n") == 0)
            {
                locctr += 3;
            }
            else if (strcmp(token_table[token_line]->operator,"BYTE") == 0)
            {
                int s = 0;
                int e = 0;
                int count = 0;
                //byte constant의 길이를 구할것임.
                for (int k = 0; k < strlen(token_table[token_line]->operand[0]); k++)
                {
                    if (token_table[token_line]->operand[0][k] == '₩'' && count == 0)
                    {
                        s = k;
                        count++;
                    }
                    else if (token_table[token_line]->operand[0][k] == '₩'' && count == 1)
                    {
                        e = k;
                        count++;
                    }
                }
                char* numbbb = malloc(sizeof(char) * (e - s)+2);
                numbbb[0] = "0";
                numbbb[1] = "x";
                count = 2;
                for (int k = s + 1; k < e; k++)
                {
                    numbbb[count] = token_table[token_line]->operand[0][k];
                }
                int nuk = strtol(numbbb, NULL, 16);
                locctr += (nuk / (int)pow(2.0, 8.0)) + 1;//2^8으로 나눈 몫 +1을 하여 length(byte수)를 구함.
            }
        }
        token_line++;

    }
    for (int i = 0; i < token_line; i++)
    {
        if (token_table[i]->operator)
        {
            if (token_table[i]->operator[0] == '+')
            {
```

```
                token_table[i]->nixbpe      +=
1;//e
            }
            if  (token_table[i]->operand[0][0]
!= '₩0')
            {
                i                               f
(token_table[i]->operand[0][0]              ==
'#')token_table[i]->nixbpe += 16;//i
                else                            if
(token_table[i]->operand[0][0]              ==
'@')token_table[i]->nixbpe += 32;//n
                else
                {
                    token_table[i]->nixbpe
+= 48;//simple addressing
                }
            }
            else   token_table[i]->nixbpe   +=
48;//simple addressing
            if  (token_table[i]->operand[1][0]
!= '₩0')
            {
                i                               f
(token_table[i]->operand[1][0] == 'X')
                {
                    token_table[i]->nixbpe
+= 8;
                }
            }
        }
    }
    //리터럴 테이블
    litcount = 0;
    for (int i = 0; i < token_line; i++)
    {
        if  (token_table[i]->operand[0][0]  ==
'=')//리터럴이 존재함.
        {
            int frs = 0;//앞부분
            int  ts  =  0;//뒷부분,  tail(리터럴의
끝)
            int cstr = 0;
            char*  temp  =  malloc(sizeof(char)
* 50);
            for (int fk = 0; fk < 50; fk++)
            {
                temp[fk] = '0';
            }
            for (int kli = 0; cstr <= 1; kli++)
            {
                i                               f
(token_table[i]->operand[0][kli]  ==  '₩''  &&
cstr == 0)
                {
                    frs = kli;
                    cstr++;
                }
                else                            if
(token_table[i]->operand[0][kli]  ==  '₩''  &&
cstr == 1)
                {
                    ts = kli;
                    cstr++;
                }
            }
            for  (int  jlm  =  frs  +  1;  jlm  <  ts;
jlm++)
            {
                temp[jlm  -  frs  -  1]  =
token_table[i]->operand[0][jlm];
            }
            temp[ts - frs - 1] = '₩0';
            int errlit = 0;
            for  (int  litin  =  0;  litin  <  litcount;
litin++)
            {
                i                               f
(strcmp(literal_table[litin].literal, temp) == 0)
                {
```

```
                    errlit = -1;
                }
            }
            if (errlit == 0)
            {
                token_table[i]->is_literal = true;
                literal_table[litcount].literal = malloc(sizeof(char) * 50);
                literal_table[litcount].literal[0] = '₩0';
                literal_table[litcount].addr = 0;
                int j = i;
                while (token_table[j]->addr != 0 && j < token_line)
                {
                    if (token_table[j]->operator!=NULL)
                    {
                        if (strcmp(token_table[j]->operator,"LTORG₩n") == 0)
                        {
                            break;
                        }
                    }
                    if (j + 1 < token_line)
                    {
                        j++;
                    }
                    else if (j + 1 == token_line)
                    {
                        break;
                    }
                }

strcpy(literal_table[litcount].literal, temp);
                token_table[i]->literal = malloc(sizeof(char) * 50);
                token_table[i]->literal[0] = '₩0';
                strcpy(token_table[i]->literal, temp);
                literal_table[litcount].addr = token_table[j]->addr;
                literal_table[litcount].section = token_table[j]->section;
                litcount++;
            }
        }
        else
        {
            token_table[i]->is_literal = false;
            token_table[i]->literal = NULL;
        }
    }
    return err;
}

/                                                                    *
------------------------------------------------------------------------
----------
* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
*          여기서 출력되는 내용은 명령어 옆에 OPCODE가 기록된 표(과제 3번) 이다.
* 매계 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프로그램의 결과를 표준출력으로 보내어
*          화면에 출력해준다.
*           또한 과제 4번에서만 쓰이는 함수이므로 이후의 프로젝트에서는 사용되지 않는다.
*
------------------------------------------------------------------------
-----------
```

```
/                                                    *
------------------------------------
------------------------------------
----------
* 설명 : 입력된 문자열의 이름을 가진 파일에
프로그램의 결과를 저장하는 함수이다.
*          여기서 출력되는 내용은 SYMBOL별
주소값이 저장된 TABLE이다.
* 매계 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프
로그램의 결과를 표준출력으로 보내어
*         화면에 출력해준다.
*
*
------------------------------------
------------------------------------
-----------
*/
void make_symtab_output(char* file_name)
{
    if (*file_name == NULL)
    {
        printf("SYMTAB₩n");
        for (int i = 0; i < symbol_count; i++)
        {
            printf("%s₩t", sym_table[i].symbol);
            printf("%X₩n", sym_table[i].addr);
        }
    }
    else
    {
    FILE* f = fopen(file_name, "w");
    for (int i = 0; i < symbol_count; i++)
    {
        fwrite(sym_table[i].symbol, sizeof(char) * strlen(sym_table[i].symbol), 1, f);
        fwrite("₩t", sizeof("₩t"), 1, f);
        fprintf(f, "%02X", sym_table[i].addr);
        fwrite("₩n", sizeof("₩n"), 1, f);
    }
    fclose(f);
    }
}

/                                                    *
------------------------------------
------------------------------------
----------
* 설명 : 입력된 문자열의 이름을 가진 파일에
프로그램의 결과를 저장하는 함수이다.
*          여기서 출력되는 내용은 LITERAL별
주소값이 저장된 TABLE이다.
* 매계 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프
로그램의 결과를 표준출력으로 보내어
*         화면에 출력해준다.
*
*
------------------------------------
------------------------------------
-----------
*/
void make_literaltab_output(char* filen_ame)
{
    if (*filen_ame == NULL)
    {

        printf("₩nLiteral Tab₩n");
        for (int i = 0; i < litcount; i++)
        {
            printf("%s₩t", literal_table[i].literal);
            printf("%X₩n", literal_table[i].addr);
        }
```

```
    }
    else
    {
        FILE* f = fopen(filen_ame, "w");
        for (int i = 0; i < litcount; i++)
        {
            fwrite(literal_table[i].literal, sizeof(char) * strlen(literal_table[i].literal), 1, f);
            fwrite("₩t", sizeof("₩t"), 1, f);
            fprintf(f, "%X", literal_table[i].addr);
            fwrite("₩n", sizeof("₩n"), 1, f);
        }
        fclose(f);
    }
}

/ *
-----------------------------------
-----------------------------------
----------
* 설명 : 어셈블리 코드를 기계어 코드로 바꾸기 위한 패스2 과정을 수행하는 함수이다.
*                 패스 2에서는 프로그램을 기계어로 바꾸는 작업은 라인 단위로 수행된다.
*                 다음과 같은 작업이 수행되어 진다.
*                 1. 실제로 해당 어셈블리 명령어를 기계어로 바꾸는 작업을 수행한다.
* 매계 : 없음
* 반환 : 정상종료 = 0, 에러발생 = < 0
* 주의 :
*
-----------------------------------
-----------------------------------
-----------
*/
static int assem_pass2(void)
{
    int err = 0;
    for (int i = 0; i < token_line; i++)
    {
        bool start = false;
        for (int isstart = 0; isstart < 5; isstart++)//section의 시작인지 검사
        {
            if (eachsection[isstart] == i)
            {
                start = true;
            }
        }
        int currsection = 0;//현재 섹션검사
        currsection = token_table[i]->section;//현재 섹션
        token_table[i]->objectcode[0] = (unsigned char)255;//op+ni부분
        token_table[i]->objectcode[1] = (unsigned char)255;//xbpe부분
        token_table[i]->objectcode[2] = (unsigned char)255;//3형식의 경우 나머지 세자리 4형식의경우 3자리
        token_table[i]->objectcode[3] = (unsigned char)255;//4형식의 경우 나머지 두자리
        if (start)//START of each section
        {
            token_table[i]->objectcode[0] = token_table[i]->addr;
        }
        else if (token_table[i]->operator!=NULL)
        {
            int opc = search_opcode(token_table[i]->operator);
            if (opc != -1)
            {
                if ((token_table[i]->nixbpe & 1) == 1)//4형식
                {
```

```
                    if (token_table[i]->operand[0][0] != '₩0')
                    {
                        int temporarymarker = 0;
                        if (token_table[i]->operand[0][0] == '@')//indirect
                        {
                            for (int issym = 0; issym < symbol_count; issym++)//현재 section에 symbol이 있는지 파악 후 계산
                            {

                                if (strcmp(sym_table[issym].symbol, strtok(token_table[i]->operand[0], "@")) == 0 && sym_table[issym].section == currsection - 1)
                                {

token_table[i]->nixbpe += 2;//pc relatvie

temporarymarker = 1;
                                    char tempni = 0;

token_table[i]->objectcode[0] = inst_table[opc]->op + (unsigned char)2;
                                    tempni = token_table[i]->nixbpe - 32;

token_table[i]->objectcode[1] = (unsigned char)tempni;

token_table[i]->objectcode[2] = (unsigned char)((sym_table[issym].addr - token_table[i + 1]->addr) / 256);

token_table[i]->objectcode[3] = (unsigned char)((sym_table[issym].addr - token_table[i + 1]->addr) % 256);

                                }

                            }
                            if (temporarymarker == 0)
                            {
                                char tempni = 0;

token_table[i]->objectcode[0] = inst_table[opc]->op + (unsigned char)2;
                                tempni = token_table[i]->nixbpe - 32;

token_table[i]->objectcode[1] = (unsigned char)tempni;

token_table[i]->objectcode[2] = 0;

token_table[i]->objectcode[3] = 0;
                                }
                            }
                            else//indirect가 0아님.(simple or immediate)
                            {
                                for (int issym = 0; issym < symbol_count; issym++)//현재 section에 symbol이 있는지 파악 후 계산
                                {

                                    if (strcmp(sym_table[issym].symbol, token_table[i]->operand[0]) == 0 && sym_table[issym].section == currsection - 1)
                                    {
```

```
temporarymarker = 1;
                                    char
tempni = 0;

token_table[i]->nixbpe += 2;//pc relatvie
                                    if
((token_table[i]->nixbpe & 32) == 32 &&
(token_table[i]->nixbpe & 16) == 16)//simple
addressing
                                    {

token_table[i]->objectcode[0]              =
inst_table[opc]->op + (unsigned char)1;

tempni = token_table[i]->nixbpe - 48;

token_table[i]->objectcode[1]   =   (unsigned
char)tempni;

token_table[i]->objectcode[2]   =   (unsigned
char)((sym_table[issym].addr   -   token_table[i
+ 1]->addr) / 256);

token_table[i]->objectcode[3]   =   (unsigned
char)((sym_table[issym].addr   -   token_table[i
+ 1]->addr) % 256);
                                    }


                                }
                            }
                            if
(temporarymarker == 0)
                            {
                                if
((token_table[i]->nixbpe & 16) == 16 &&
(token_table[i]->nixbpe      &      32)      !=
32)//immediate
                                {
                                    char
tempni = 0;

token_table[i]->objectcode[0]              =
inst_table[opc]->op + (unsigned char)1;
                                    tempni
= token_table[i]->nixbpe - 16;

token_table[i]->objectcode[1]   =   (unsigned
char)tempni;
                                    int
templength                         =
atoi(strtok(token_table[i]->operand[0], "#"));

token_table[i]->objectcode[2]   =   (unsigned
char)(templength / 256);

token_table[i]->objectcode[3]   =   (unsigned
char)(templength % 256);

                                }
                                else        if
(token_table[i]->is_literal == true)//literal
                                {
                                    for (int j
= 0; j < litcount; j++)
                                    {
                                        if
(strcmp(literal_table[j].literal,
token_table[i]->literal) == 0)
                                        {

token_table[i]->nixbpe += 2;//pc relatvie

char tempni = 0;

token_table[i]->objectcode[0]              =
inst_table[opc]->op + (unsigned char)3;

tempni = token_table[i]->nixbpe - 48;
```

```
token_table[i]->objectcode[1] = (unsigned char)tempni;

token_table[i]->objectcode[2] = (unsigned char)((literal_table[j].addr - token_table[i + 1]->addr) / 256);

token_table[i]->objectcode[3] = (unsigned char)((literal_table[j].addr - token_table[i + 1]->addr) % 256);
                                        }

                                    }
                                }
                                else//simple && not in section
                                {
                                    char tempni = 0;

token_table[i]->objectcode[0] = inst_table[opc]->op + (unsigned char)3;
                                    tempni = token_table[i]->nixbpe - 48;

token_table[i]->objectcode[1] = (unsigned char)tempni;

token_table[i]->objectcode[2] = 0;

token_table[i]->objectcode[3] = 0;
                                }
                            }
                        }
                    }
                }
                else if (inst_table[opc]->format == 1)//1형식
                {

token_table[i]->objectcode[0] = inst_table[opc]->op;//opcode만 들어감.
                }
                else if (inst_table[opc]->format == 2)//2형식
                {

                    if (inst_table[opc]->ops == 1)//operand가 1개
                    {
                        if (token_table[i]->operand[0][0] == 'X')
                        {

token_table[i]->objectcode[0] = inst_table[opc]->op;

token_table[i]->objectcode[1] = (unsigned char)16;
                        }
                        else if (token_table[i]->operand[0][0] == 'A')
                        {

token_table[i]->objectcode[0] = inst_table[opc]->op;

token_table[i]->objectcode[1] = (unsigned char)00;
                        }
                        else if (token_table[i]->operand[0][0] == 'S')
                        {

token_table[i]->objectcode[0] = inst_table[opc]->op;

token_table[i]->objectcode[1] = (unsigned
```

```
char)16 * 4;
                        }
                        else if
(token_table[i]->operand[0][0] == 'T')
                        {

token_table[i]->objectcode[0]                =
inst_table[opc]->op;

token_table[i]->objectcode[1]   =   (unsigned
char)16 * 5;
                        }
                    }
                    else if
(inst_table[opc]->ops == 2)
                    {
                        i                    f
(token_table[i]->operand[0][0] == 'X')
                        {

token_table[i]->objectcode[0]                =
inst_table[opc]->op;

token_table[i]->objectcode[1]   =   (unsigned
char)16;
                        }
                        else if
(token_table[i]->operand[0][0] == 'A')
                        {

token_table[i]->objectcode[0]                =
inst_table[opc]->op;

token_table[i]->objectcode[1]   =   (unsigned
char)00;
                        }
                        else if
(token_table[i]->operand[0][0] == 'S')
                        {

token_table[i]->objectcode[0]                =
inst_table[opc]->op;

token_table[i]->objectcode[1]   =   (unsigned
char)16 * 4;
                        }
                        else if
(token_table[i]->operand[0][0] == 'T')
                        {

token_table[i]->objectcode[0]                =
inst_table[opc]->op;

token_table[i]->objectcode[1]   =   (unsigned
char)16 * 5;
                        }
                        i                    f
(token_table[i]->operand[1][0] == 'X')
                        {

token_table[i]->objectcode[1]   +=   (unsigned
char)1;
                        }
                        else if
(token_table[i]->operand[1][0] == 'A')
                        {

token_table[i]->objectcode[1]   +=   (unsigned
char)0;
                        }
                        else if
(token_table[i]->operand[1][0] == 'S')
                        {

token_table[i]->objectcode[1]   +=   (unsigned
char)4;
                        }
                        else if
(token_table[i]->operand[1][0] == 'T')
```

```
                    {
token_table[i]->objectcode[1] += (unsigned char)1;
                    }
                }
            }

            else if (inst_table[opc]->format == 3)
            {
                if (token_table[i]->operand[0][0] != '₩0')
                {
                    int temporarymarker = 0;
                    if (token_table[i]->operand[0][0] == '@')//indirect
                    {
                        for (int issym = 0; issym < symbol_count; issym++)//현재 section에 symbol이 있는지 파악 후 계산
                        {

                            if (strcmp(sym_table[issym].symbol, strtok(token_table[i]->operand[0], "@")) == 0 && sym_table[issym].section == currsection - 1)
                            {

token_table[i]->nixbpe += 2;//pc relatvie

temporarymarker = 1;
                                char tempni = 0;

token_table[i]->objectcode[0] = inst_table[opc]->op + (unsigned char)2;
                                tempni = token_table[i]->nixbpe - 32;

token_table[i]->objectcode[1] = (unsigned char)tempni;

token_table[i]->objectcode[2] = (unsigned char)((sym_table[issym].addr - token_table[i + 1]->addr));

                            }

                        }
                        if (temporarymarker == 0)
                        {
                            char tempni = 0;

token_table[i]->objectcode[0] = inst_table[opc]->op + (unsigned char)2;
                            tempni = token_table[i]->nixbpe - 32;

token_table[i]->objectcode[1] = (unsigned char)tempni;

token_table[i]->objectcode[2] = 0;
                        }
                    }
                    else//indirect가 0아님.(simple or immediate)
                    {
                        for (int issym = 0; issym < symbol_count; issym++)//현재 section에 symbol이 있는지 파악 후 계산
                        {

                            if (strcmp(sym_table[issym].symbol,
```

```
token_table[i]->operand[0]) == 0 &&
sym_table[issym].section == currsection - 1)
                                {

temporarymarker = 1;
                                    char
tempni = 0;

token_table[i]->nixbpe += 2;//pc relatvie
                                    if
((token_table[i]->nixbpe & 32) == 32 &&
(token_table[i]->nixbpe & 16) == 16)//simple
addressing
                                    {

token_table[i]->objectcode[0] =
inst_table[opc]->op + (unsigned char)1;

tempni = token_table[i]->nixbpe - 48;

token_table[i]->objectcode[1] = (unsigned
char)tempni;

token_table[i]->objectcode[2] = (unsigned
char)((sym_table[issym].addr - token_table[i
+ 1]->addr));
                                    }


                                }
                            }
                            if
(temporarymarker == 0)
                            {
                                if
((token_table[i]->nixbpe & 16) == 16 &&
(token_table[i]->nixbpe & 32) !=
32)//immediate
                                {
                                    char
tempni = 0;

token_table[i]->objectcode[0] =
inst_table[opc]->op + (unsigned char)1;
                                    tempni
= token_table[i]->nixbpe - 16;

token_table[i]->objectcode[1] = (unsigned
char)tempni;
                                    int
templength =
atoi(strtok(token_table[i]->operand[0], "#"));

token_table[i]->objectcode[2] = (unsigned
char)(templength);


                                }
                                else if
(token_table[i]->is_literal == true)//literal
                                {
                                    for (int j
= 0; j < litcount; j++)
                                    {
                                        if
(strcmp(literal_table[j].literal,
token_table[i]->literal) == 0)
                                        {

token_table[i]->nixbpe += 2;//pc relatvie

char tempni = 0;

token_table[i]->objectcode[0] =
inst_table[opc]->op + (unsigned char)3;

tempni = token_table[i]->nixbpe - 48;
```

```
token_table[i]->objectcode[1] = (unsigned char)tempni;

token_table[i]->objectcode[2] = (unsigned char)((literal_table[j].addr - token_table[i + 1]->addr));

                                    }


                                }
                                else//simple && not in section
                                {
                                    char tempni = 0;

token_table[i]->objectcode[0] = inst_table[opc]->op + (unsigned char)3;
                                    tempni = token_table[i]->nixbpe - 48;

token_table[i]->objectcode[1] = (unsigned char)tempni;

token_table[i]->objectcode[2] = 0;

                                }
                            }
                        }
                    }
                    else
                    {
                        char tempni = 0;

token_table[i]->objectcode[0] = inst_table[opc]->op + (unsigned char)3;
                        tempni = token_table[i]->nixbpe - 48;

token_table[i]->objectcode[1] = (unsigned char)tempni;

token_table[i]->objectcode[2] = 0;
                    }
                }


            }

            else if (strcmp(token_table[i]->operator,"WORD") == 0 || strcmp(token_table[i]->operator,"BYTE"))
            {
                if (strcmp(token_table[i]->operator,"WORD") == 0)
                {
                    char *temp=malloc(sizeof(token_table[i]->operand[0]));
                    strcpy(temp, token_table[i]->operand[0]);
                    char* temp2 = strtok(temp, "-");
                    char* temp3 = strtok(NULL, "-");
                    int found1 = 0;
                    int found2 = 0;
                    for (int ksearchsym=0;ksearchsym<symbol_count;ksearchsym++)
                    {
                        if (strcmp(sym_table[ksearchsym].symbol, temp2) == 0 && sym_table[ksearchsym].section == token_table[i]->section)
```

```
                        {
                            found1 =
ksearchsym;
                        }
                        else if
(strcmp(sym_table[ksearchsym].symbol,
temp3) == 0 &&
sym_table[ksearchsym].section ==
token_table[i]->section)
                        {
                            found2 =
ksearchsym;
                        }
                    }
                    if(found1!=0 &&
found2!=0)
                    {

token_table[i]->objectcode[0] = (unsigned
char)(sym_table[found1].addr -
sym_table[found2].addr);

token_table[i]->objectcode[1] = 0;

token_table[i]->objectcode[2] = 0;
                    }
                    else
                    {

token_table[i]->objectcode[0] = 0;

token_table[i]->objectcode[1] = 0;

token_table[i]->objectcode[2] = 0;
                    }
                }
                else if
(strcmp(token_table[i]->operator,"BYTE") ==
0)
                {
                    int s = 0;
                    int e = 0;
                    int count = 0;

                    for (int k = 0; k <
strlen(token_table[token_line]->operand[0]);
k++)
                    {
                        i f
(token_table[token_line]->operand[0][k] ==
'₩'' && count == 0)
                        {
                            s = k;
                            count++;
                        }
                        else if
(token_table[token_line]->operand[0][k] ==
'₩'' && count == 1)
                        {
                            e = k;
                            count++;
                        }
                    }
                    unsigned char* numbbb
= malloc(sizeof(char) * (e - s) + 2);
                    numbbb[0] = "0";
                    numbbb[1] = "x";
                    count = 2;
                    for (int k = s + 1; k <
e; k++)
                    {
                        numbbb[count] =
token_table[token_line]->operand[0][k];
                    }
                    int temper =
strtol(numbbb, NULL, 16);

token_table[i]->objectcode[0] = (unsigned
char)temper;
                }
```

```
            }
        }
    }
    return err;
}


/                                                   *
------------------------------------
------------------------------------
----------
* 설명 : 입력된 문자열의 이름을 가진 파일에
프로그램의 결과를 저장하는 함수이다.
*          여기서 출력되는 내용은 object code
(프로젝트 1번) 이다.
* 매개 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프
로그램의 결과를 표준출력으로 보내어
*          화면에 출력해준다.
*
*
------------------------------------
------------------------------------
-----------
*/
void make_objectcode_output(char* file_name)
{
    /*
    for (int i = 0; i < token_line; i++)
    {
        printf("%X       %s                    :",
t o k e n _ t a b l e [ i ] - > a d d r ,
token_table[i]->operator);
        for (int j = 0; j < 4; j++)
        {

            if   (token_table[i]->objectcode[j]
!= (unsigned char)255)
            {
                if (j == 0)
                {
                    p r i n t f ( " % 0 2 X " ,
token_table[i]->objectcode[j]);
                }
                else if (j == 1)
                {
                    p r i n t f ( " % X " ,
token_table[i]->objectcode[j]);
                }
                else if (j == 2)
                {
                    p r i n t f ( " % 0 3 X " ,
token_table[i]->objectcode[j]);
                }
                else   if   (j   ==   3   &&
token_table[i]->operator[0]=='+')
                {
                    p r i n t f ( " % 0 2 X " ,
token_table[i]->objectcode[j]);
                }
            }
        }
        printf("₩n");
    }
    */

    if (*file_name == NULL)
    {
        int refindex = 0;
        int countforsection = 0;
        for (int i = 0; i < token_line; i++)
        {
            int linecount = 0;//줄마다 길이 측
정.
            if              (i              ==
eachsection[countforsection])
            {

                if (i != 0)
```

```
                {

                    for (int modi = eachsection[countforsection - 1];modi<eachsection[countforsection];modi++)
                    {
                        if (token_table[refindex]->operand[0][0] != '₩0')
                        {
                            if (strstr(token_table[modi]->operand[0], token_table[refindex]->operand[0]) !=NULL)
                            {
                                if (strchr(token_table[modi]->operand[0], "-") != NULL)
                                {
                                    printf("M %X %02X +%s₩n", token_table[modi]->addr,6,strtok(token_table[modi]->operand[0], "-"));
                                    printf("M %X %02X -%s₩n", token_table[modi]->addr, 6, strtok(NULL, "-"));
                                }
                                else
                                {
                                    printf("M %X %02X +%s₩n", token_table[modi]->addr + 1, 5, token_table[modi]->operand[0]);
                                }
                            }
                        }


                    }
                }

                printf("H%s", token_table[i]->label);
                printf("%06X %06X₩n", sttadd, progrlength[countforsection]);
                countforsection++;
            }
            else if (token_table[i]->operator!=NULL)
            {
                if (strcmp(token_table[i]->operator,"EXTDEF") == 0)
                {
                    printf("D");
                    for (int j = 0; j < MAX_OPERAND; j++)
                    {
                        if (token_table[i]->operand[j][0] != '₩0')
                        {
                            for (int searsym = 0; searsym < symbol_count; searsym++)
                            {
                                if (strcmp(sym_table[searsym].symbol, token_table[i]->operand[j]) == 0)
                                {

                                    printf("%s%X", token_table[i]->operand[j], sym_table[searsym].addr);
                                }
                            }
                        }
                    }
                    printf("₩n");
                }
                else if (strcmp(token_table[i]->operator,"EXTREF") == 0)
                {
                    refindex = i;
```

```
                printf("R");
                for (int j = 0; j < MAX_OPERAND; j++)
                {
                    if (token_table[i]->operand[j][0] != '₩0')
                    {
                        printf("%s₩t", token_table[i]->operand[j]);
                    }
                }
                printf("₩n");
            }
            else
            {
                int j = i;
                int endofthisline = 0;
                int lineco = 0;
                for (j; j < eachsection[countforsection]; j++)
                {
                    if (token_table[j]->objectcode[0] != 255)
                    {
                        linecount += 2;
                    }
                    if (token_table[j]->objectcode[1] != 255)
                    {
                        linecount += 1;
                    }
                    if (token_table[j]->objectcode[2] != 255)
                    {
                        linecount += 3;
                    }
                    if (token_table[j]->objectcode[3] != 255)
                    {
                        linecount += 2;
                    }
                    if (linecount > 30)
                    {
                        endofthisline = j-1;
                        j = token_line - 1;
                    }
                    if(linecount<=30)
                    {
                        lineco = linecount;
                        endofthisline = j - 1;
                    }
                }
                printf("T %X %02X", token_table[i]->addr,lineco);
                for (int k = i; k <= endofthisline; k++)
                {
                    for (int l = 0; l < 4; l++)
                    {
                        if (token_table[k]->objectcode[l] != 255)
                        {
                            if (l == 0)
                            {
                                printf("%02X", token_table[k]->objectcode[l]);
                            }
                            else if (l == 1)
                            {
                                printf("%X", token_table[k]->objectcode[l]);
                            }
                            else if (l == 2)
```

```
                                {
printf("%03X", token_table[k]->objectcode[l]);
                                }
                                else if (l == 3)
                                {
printf("%02X", token_table[k]->objectcode[l]);
                                }
                            }
                        }
                        printf("₩n");
                        i = endofthisline;
                    }
                }
            }
        }
        else
        {
            FILE* f = fopen(file_name, "w");
            int refindex = 0;
            int countforsection = 0;
            for (int i = 0; i < token_line; i++)
            {
                int linecount = 0;//줄마다 길이 측정.
                if (i == eachsection[countforsection])
                {
                    if (i != 0)
                    {
                        for (int modi = eachsection[countforsection - 1]; modi < eachsection[countforsection]; modi++)
                        {
                            if (token_table[refindex]->operand[0][0] != '₩0')
                            {
                                if (strstr(token_table[modi]->operand[0], token_table[refindex]->operand[0]) != NULL)
                                {
                                    if (strchr(token_table[modi]->operand[0], "-") != NULL)
                                    {
fprintf(f,"M %X %02X +%s₩n", token_table[modi]->addr, 6, strtok(token_table[modi]->operand[0], "-"));
fprintf(f,"M %X %02X -%s₩n", token_table[modi]->addr, 6, strtok(NULL, "-"));
                                    }
                                    else
                                    {
fprintf(f,"M %X %02X +%s₩n", token_table[modi]->addr + 1, 5, token_table[modi]->operand[0]);
                                    }
                                }
                            }
                        }
                    }
                    fprintf(f,"H%s", token_table[i]->label);
                    fprintf(f,"%X%X₩n", sttadd, progrlength[countforsection]);
                    countforsection++;
```

```
            }
            else if (token_table[i]->operator!=NULL)
            {
                if (strcmp(token_table[i]->operator,"EXTDEF") == 0)
                {
                    fprintf(f,"D");
                    for (int j = 0; j < MAX_OPERAND; j++)
                    {
                        if (token_table[i]->operand[j][0] != '₩0')
                        {
                            for (int searsym = 0; searsym < symbol_count; searsym++)
                            {
                                if (strcmp(sym_table[searsym].symbol, token_table[i]->operand[j]) == 0)
                                {

fprintf(f,"%s%X", token_table[i]->operand[j], sym_table[searsym].addr);
                                }
                            }
                        }
                    }
                    fprintf(f,"₩n");
                }
                else if (strcmp(token_table[i]->operator,"EXTREF") == 0)
                {
                    refindex = i;
                    fprintf(f,"R");
                    for (int j = 0; j < MAX_OPERAND; j++)
                    {
                        if (token_table[i]->operand[j][0] != '₩0')
                        {
                            fprintf(f,"%s", token_table[i]->operand[j]);
                        }
                    }
                    fprintf(f,"₩n");
                }
                else
                {
                    int j = i;
                    int endofthisline = 0;
                    int lineco = 0;
                    for (j; j < eachsection[countforsection]; j++)
                    {
                        if (token_table[j]->objectcode[0] != 255)
                        {
                            linecount += 2;
                        }
                        if (token_table[j]->objectcode[1] != 255)
                        {
                            linecount += 1;
                        }
                        if (token_table[j]->objectcode[2] != 255)
                        {
                            linecount += 3;
                        }
                        if (token_table[j]->objectcode[3] != 255)
                        {
                            linecount += 2;
                        }
                        if (linecount > 30)
                        {
                            endofthisline = j
```

```
- 1;
                    j = token_line -
1;
                }
                if (linecount <= 30)
                {
                    lineco =
linecount;
                    endofthisline = j
- 1;
                }
            }
            fprintf(f,"T%X%02X",
token_table[i]->addr, lineco);
            for (int k = i; k <=
endofthisline; k++)
            {
                for (int l = 0; l < 4;
l++)
                {
                    i f
(token_table[k]->objectcode[l] != 255)
                    {
                        if (l == 0)
                        {

f p r i n t f ( f , " % 0 2 X " ,
token_table[k]->objectcode[l]);
                        }
                        else if (l ==
1)
                        {

fprintf(f,"%X", token_table[k]->objectcode[l]);
                        }
                        else if (l ==
2)
                        {

f p r i n t f ( f , " % 0 3 X " ,
token_table[k]->objectcode[l]);
                        }
                        else if (l ==
3)
                        {

f p r i n t f ( f , " % 0 2 X " ,
token_table[k]->objectcode[l]);
                        }
                    }
                }
            }
            fprintf(f,"₩n");
            i = endofthisline;

        }
    }
    fclose(f);
}
}
```

```c
/*
 * my_assembler 함수를 위한 변수 선언 및 매크로를 담고 있는 헤더 파일이다.
 *
 */
#define MAX_INST 256
#define MAX_LINES 5000
#define MAX_OPERAND 3

 /*
  * instruction 목록 파일로 부터 정보를 받아와서 생성하는 구조체 변수이다.
  * 구조는 각자의 instruction set의 양식에 맞춰 직접 구현하되
  * 라인 별로 하나의 instruction을 저장한다.
  */
struct inst_unit
{
    char str[10];
    unsigned char op;
    int format;
    int ops;
};

// instruction의 정보를 가진 구조체를 관리하는 테이블 생성
typedef struct inst_unit inst;
inst* inst_table[MAX_INST];
int inst_index;

/*
 * 어셈블리 할 소스코드를 입력받는 테이블이다. 라인 단위로 관리할 수 있다.
 */
char* input_data[MAX_LINES];
static int line_num;

/*
 * 어셈블리 할 소스코드를 토큰단위로 관리하기 위한 구조체 변수이다.
 * operator는 renaming을 허용한다.
 * nixbpe는 8bit 중 하위 6개의 bit를 이용하여 n,i,x,b,p,e를 표시한다.
 */
struct token_unit
{
    char* label;
    char* operator;
    char operand[MAX_OPERAND][20];
    char comment[100];
    char nixbpe;
    unsigned char objectcode[4];//오브젝트코드 저장
    int addr;//locctr값을 모두 적어주기 위함.
    int section;//section값 지정.
    bool is_literal ;//literal인지
    char* literal;
};

typedef struct token_unit token;
token* token_table[MAX_LINES];
static int token_line;

/*
 * 심볼을 관리하는 구조체이다.
 * 심볼 테이블은 심볼 이름, 심볼의 위치로 구성된다.
 */
struct symbol_unit
{
    char* symbol;
    int addr;
    int section;//몇번째 섹션인지
};

/*
* 리터럴을 관리하는 구조체이다.
* 리터럴 테이블은 리터럴의 이름, 리터럴의 위치로 구성된다.
* 추후 프로젝트에서 사용된다.
*/
struct literal_unit
{
    char* literal;
    int addr;
    int section;//몇번째 섹션인지.저장해 주기 위하여 추가.
};

typedef struct symbol_unit symbol;
symbol sym_table[MAX_LINES];
```

```
static int symbol_count = 0;//symbol의 갯수 저장 변수
typedef struct literal_unit literal;
literal literal_table[MAX_LINES];
static int litcount;//리터럴 테이블의 갯수를 새어주기위한 변수.
static int progrlength[3];
int sttadd = 0;//starting address
int eachsection[5] = { 0,0,0,0,0 };//각 섹션의 시작 토큰번호 저장.
static int locctr;
//---------------

static char* input_file;
static char* output_file;
int init_my_assembler(void);
int init_inst_file(char* inst_file);
int init_input_file(char* input_file);
int token_parsing(char* str);
int search_opcode(char* str);
static int assem_pass1(void);
void make_opcode_output(char* file_name);

void make_symtab_output(char* file_name);
void make_literaltab_output(char* file_name);
static int assem_pass2(void);
void make_objectcode_output(char* file_name);
```