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	파서 구현 (Program 과제 #3)
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20192698 심원준(출석번호 124)
제출일	2021년 4월 12일

차 례

1장. 프로젝트 목적/동기

이 프로젝트는 SIC/XE머신의 어셈블러, 그 중에서 파싱 프로그램을 구현해봄을 목적으로 한다. input.txt파일과 명령어와 opcode가 적힌 inst.data파일을 입력받아, 명령어에 해당하는 OPCODE를 찾아 해당 명령어 옆에 출력하게 된다.
이를 통하여 기본적인 SIC/XE머신을 이해함을 목적으로 한다.

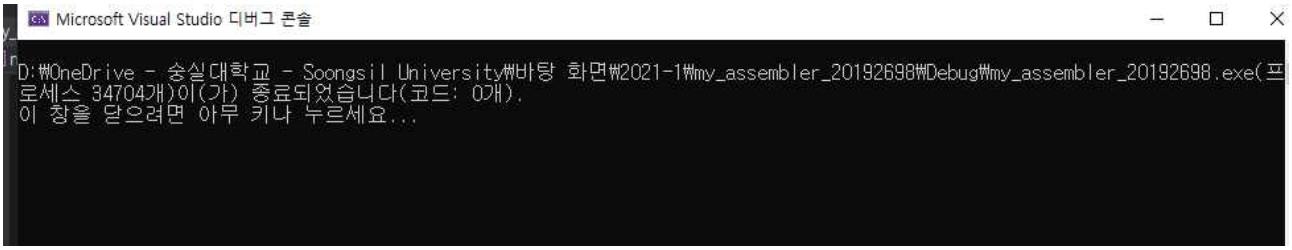
2장. 설계/구현 아이디어

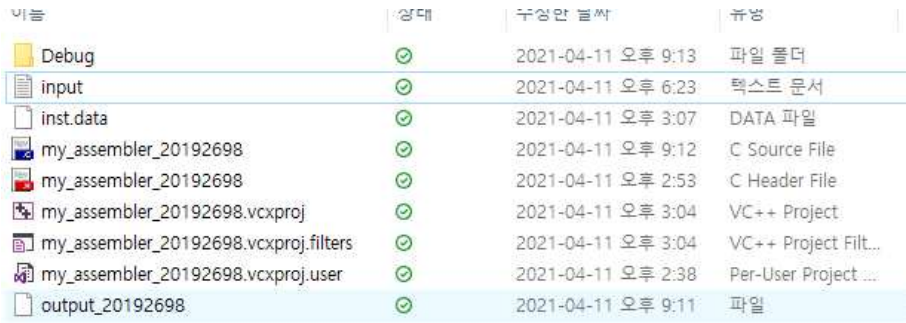
(1)주어진 바와 같이 assembler를 초기화 해주는 init_my_assembler함수 안에, instruction file에서 명령어와 각 명령어의 opcode등을 불러오는 init_inst_file함수, 그리고 input file에서 한줄씩 읽어오는 동작을 하는 init_input_file함수로 구성하였다. init_inst_file함수는 기본적인 file I/o를 이용하여 한 줄 씩 읽어오는데, 이때 구분자를 " "(띄어쓰기)로 두어 명령어와 opcode,형식,operand 개수 등을 구분하도록 하였다.

(2)input.txt를 받아서 input_data로 넘겨줄 때에, 각각의 배열에 input.txt의 한 줄 씩을 받아 입력해주었다. 이것을 token_parsing함수를 이용해 한 배열씩 받아, label의 존재여부, operator의 존재여부, operand의 개수와 commment존재여부를 파악하여 각각 token_table 구조체 배열에 넣어주도록 하였다. 이 과정에서 예외처리를 위하여 조건문을 활용하였다. 또한 오류를 방지하기 위하여 모든 구조체 배열은 미리 초기화해둔 뒤 사용하도록 하였다.

(3)이후 pass1에서 위의 과정을 실행한 뒤, 주어진 이름의 파일(output_20192698/NULL일 경우 cmd창에) 매핑된 opcode를 출력하도록 하였다. 이 과정에서 각 token_table의 operator와 inst_table의 명령어를 비교하고, inst_table에 없는 것들(START제외)은 opcode의 출력없이 그대로 출력하고, inst_table에 있는, 즉 operator인 것들은 opcode를 매핑하여 출력하도록 설계하고자 하였다.

3장 수행결과





수행 결과 output_20192698 파일이 출력되었다.

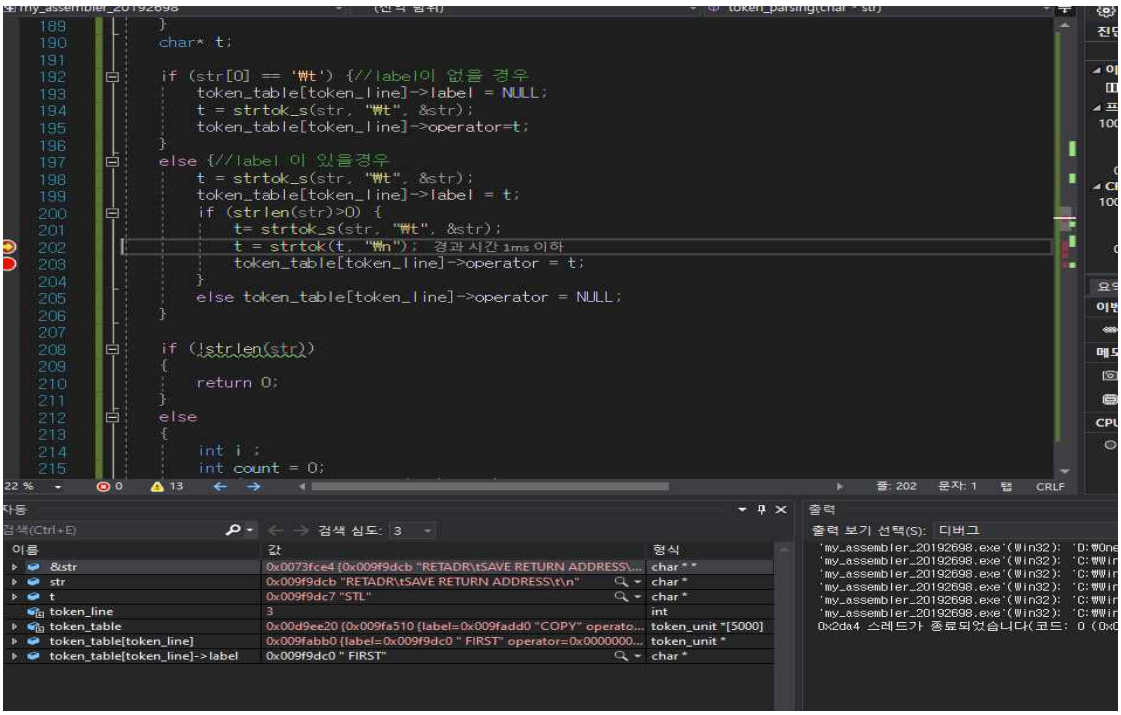
만약 파일 이름이 NULL일 시 위와같이 출력된다.

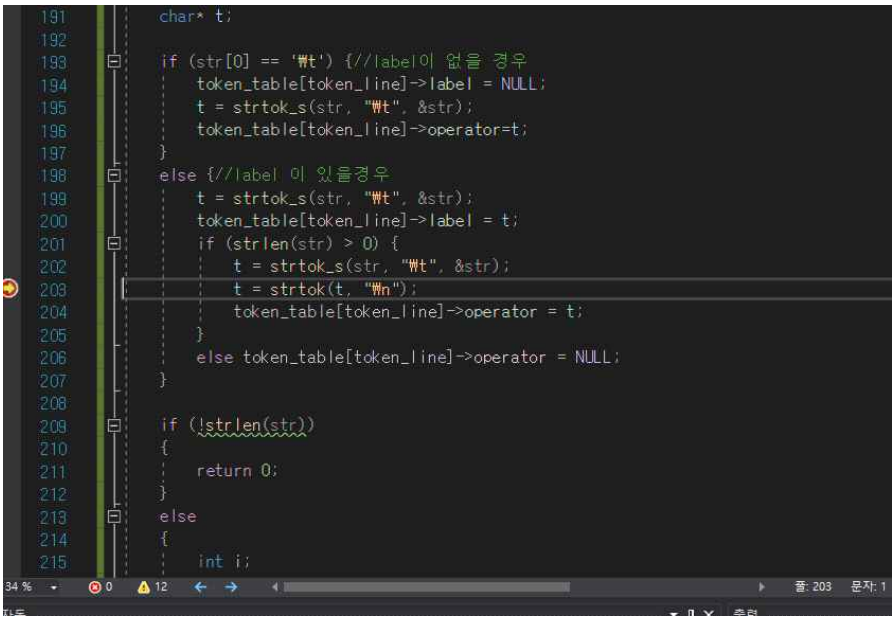
4장　결론 및 보충할 점

디버깅을 통하여 token으로 input이 나뉘어 들어가짐을 확인하였고, 또한 inst_input file또한 알맞게 저장됨을 확인하였으며 이 프로그램을 구현하는 과정에서 자주 쓰이는 포인터와 구조체 배열의 포인터/ 그들의 역참조등에 대해서 다시 공부하여 좀 더 원활한 프로그래밍이 가능토록 할 필요성이 보였다.
또한 파일 입출력 과정에서 16진수의 입출력에 관하여 다시 살펴볼 필요가 있어 보인다.
또 앞으로 pass2 등 프로젝트를 계속해 나갈 것인데, 미리 pass1과 pass1에서 만들어지는 중간 파일, pass2의 과정에 대해서 공부할 필요가 있어 보인다.

OPCODE를 매핑해줌에 넘어서 앞으로는 주소값을 계산하고, 이를 object코드로 만들어보는 과정을 거침이 필요하다.

5장 디버깅


189 }
190 char* t;
191
192 if (str[0] == '₩t') {//label이 없을 경우
193 token_table[token_line]->label = NULL;
194 t = strtok_s(str, "₩t", &str);
195 token_table[token_line]->operator=t;
196 }
197 else {//label 이 있을경우
198 t = strtok_s(str, "₩t", &str);
199 token_table[token_line]->label = t;
200 if (strlen(str)>0) {
201 t= strtok_s(str, "₩t", &str);
202 t = strtok(t, "₩n"); 경과 시간 1ms 이하
203 token_table[token_line]->operator = t;
204 }
205 else token_table[token_line]->operator = NULL;
206 }
207
208 if (!strlen(str))
209 {
210 return 0;
211 }
212 else
213 {
214 int i ;
215 int count = 0;
22 % 0 13 줄: 202 문자: 1 탭 CRLF
자동
검색(Ctrl+E) 검색 심도: 3
이름 값 형식
&str 0x0073fce4 {0x009f9dcb "RETADR\tSAVE RETURN ADDRESS\... char * *
str 0x009f9dcb "RETADR\tSAVE RETURN ADDRESS\t\n" char *
t 0x009f9dc7 "STL" char *
token_line 3 int
token_table 0x00d9ee20 {0x009fa510 {label=0x009fadd0 "COPY" operato... token_unit *[5000]
token_table[token_line] 0x009fabb0 {label=0x009f9dc0 " FIRST" operator=0x0000000... token_unit *
token_table[token_line]->label 0x009f9dc0 " FIRST" char *
출력
출력 보기 선택(S): 디버그
'my_assembler_20192698.exe'(Win32): 'D:₩One
'my_assembler_20192698.exe'(Win32): 'C:₩Win
'my_assembler_20192698.exe'(Win32): 'C:₩Win
'my_assembler_20192698.exe'(Win32): 'C:₩Win
'my_assembler_20192698.exe'(Win32): 'C:₩Win
'my_assembler_20192698.exe'(Win32): 'C:₩Win
0x2da4 스레드가 종료되었습니다(코드: 0 (0x0


191 char* t;
192
193 if (str[0] == '₩t') {//label이 없을 경우
194 token_table[token_line]->label = NULL;
195 t = strtok_s(str, "₩t", &str);
196 token_table[token_line]->operator=t;
197 }
198 else {//label 이 있을경우
199 t = strtok_s(str, "₩t", &str);
200 token_table[token_line]->label = t;
201 if (strlen(str) > 0) {
202 t = strtok_s(str, "₩t", &str);
203 t = strtok(t, "₩n");
204 token_table[token_line]->operator = t;
205 }
206 else token_table[token_line]->operator = NULL;
207 }
208
209 if (!strlen(str))
210 {
211 return 0;
212 }
213 else
214 {
215 int i;
34 % 0 12 줄: 203 문자: 1

```
270     /*
271     * ------------------------------------------------------------------------
272     */
273     int search_opcode(char* str)
274     {
275         int an = -1;
276         for (int i = 0; i < inst_index; i++)
277         {
278             if (strcmp(inst_table[i]->str, str) == 0) 경과시간 1ms 이하
279             {
280                 an = i;
281                 break;
282             }
283         }
284         return an;
285     }
286
287     /* ------------------------------------------------------------------------
288     * 설명 : 어셈블리 코드를 위한 패스1과정을 수행하는 함수이다.
289     *        패스1에서는..
290     *        1. 프로그램 소스를 스캔하여 해당하는 토큰단위로 분리하여 프로그램 라인별 토큰
```

134 % 0 12 줄: 278 문자: 1 탭 CRLF

자동 출력

검색(Ctrl+E) 검색 심도: 3

```
119     char o1, o2;
120     int b;
121     int d;
122     inst_index = 0;
123     while (!feof(inst_file_to_read)) {
124         inst_table[i] = malloc(sizeof(struct inst_unit));//저장공간 할당
125         inst_table[i]->str[0] = "#0";
126         inst_table[i]->op=malloc(sizeof(unsigned char));
127         inst_table[i]->format=0;
128         inst_table[i]->ops=0;
129
130         fscanf(inst_file_to_read, "%s %d %c%c %d", st, &b, &o1, &o2, &d);// inst로부터 입력받음
131         strcpy(inst_table[i]->str, st);// 변수에 저장
132         inst_table[i]->format = b;
133         inst_table[i]->op = dic[o1] * 16 + dic[o2]; 경과시간 1ms 이하
134         inst_table[i]->ops = d;
135         i++;
136         inst_index++;
137     }
138     fclose(inst_file_to_read);// 파일닫기
139     return errno;
140 }
```

134 % 0 12

자동

검색(Ctrl+E) 검색 심도: 3

이름	값	형식
dic[o1]	12	int
dic[o2]	4	int
i	11	int
inst_file_to_read	0x00f86c38 {_Placeholder=0x00f89be6 }	_iobuf *
inst_table	0x006b8a60 {0x00f80550 {str=0x00f80550 "ADD" op=24 '\x18' format=3 ...}, 0x00f859f8 {str=0x...	inst_unit *[256]
inst_table[i]	0x00f8f720 {str=0x00f8f720 "FIX" op=136 '?' format=1 ...}	inst_unit *
inst_table[i]->format	1	int
inst_table[i]->op	136 '?'	unsigned char
inst_table[i]->str	0x00f8f720 "FIX"	char[10]
o1	67 'C'	char
o2	52 '4'	char
st	0x00affa6c "FIX"	char[15]

자동 로컬 조사식 1

6장 소스코드(+주석)

my_assembler_20192698.c

```
/*
 * 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 "my_assembler_20192698.h"
token_line = 0;//토큰 저장에 사용할 line전역변수 초기화
/                                                                *
------------------------------------------------------------------------------------
 * 설명 : 사용자로 부터 어셈블리 파일을 받아서 명령어의 OPCODE를 찾아 출력한다.
 * 매계 : 실행 파일, 어셈블리 파일
 * 반환 : 성공 = 0, 실패 = < 0
 * 주의 : 현재 어셈블리 프로그램의 리스트 파일을 생성하는 루틴은 만들지 않았다.
 *                  또한 중간파일을 생성하지 않는다.
 *
------------------------------------------------------------------------------------
 */
int main(int args, char* arg[])
{
        if (init_my_assembler() < 0)
        {
                printf("init_my_assembler:  프로그램 초기화에 실패 했습니다.₩n");
                return -1;
        }

        if (assem_pass1() < 0)
        {
                printf("assem_pass1: 패스1 과정에서 실패하였습니다.  ₩n");
                return -1;
        }


make_opcode_output("output_20192698");

        /*
         * 추후 프로젝트에서 사용되는 부분
         *

make_symtab_output("symtab_00000000");
        if (assem_pass2() < 0)
        {
                printf("  assem_pass2:  패스2 과정에서 실패하였습니다.  ₩n");
                return -1;
        }


make_objectcode_output("output_00000000");
        */
        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)//inst.data 입력받기.
{
        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
%c %c %c %d",st,&b,&o1,&o2,&d);// inst로부
터 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 소스코드
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 분석하
고 토큰 테이블을 작성하는 함수이다.
 *        패스 1로 부터 호출된다.
 * 매계 : 파싱을 원하는 문자열
 * 반환 : 정상종료 = 0 , 에러 < 0
 * 주의 : my_assembler 프로그램에서는 라인단
위로 토큰 및 오브젝트 관리를 하고 있다.
 *
------------------------------------
------------------------------------
----------
 */
```

```
int token_parsing(char* str)
{
	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 = 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';
	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 {//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;
		}
		else token_table[token_line]->operator = NULL;
	}

	if (!strlen(str))
	{
		return 0;
	}
	else
	{
		int i;
		int count = 0;
		for (i = 0; i < strlen(str); i++)
		{
			if (str[i] == "₩t")count += 1;
			if (count == 2)break;
		}
		i = i + 1;
		count = 0;
		int c2 = 0;
		if (str[i] != "₩t")//operands 존재
		{
			for (i; 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;
                                }
                        }
                        count = 0;
                        for (i = i + 1; i <
strlen(str); i++)
                        {

strcpy(token_table[token_line]->comment[cou
nt], str[i]);

                                count++;
                        }
                }
                else if (str[i] ==
"₩t")//operand 미존재
                {
                        int count = 0;
                        i = i + 1;
                        for (i; i < strlen(str);
i++)
                        {

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

        return err;
}

/                                               *
-----------------------------------
-----------------------------------
----------
 * 설명 : 입력 문자열이 기계어 코드인지를 검사
하는 함수이다.
 * 매계 : 토큰 단위로 구분된 문자열
 * 반환 : 정상종료 = 기계어 테이블 인덱스, 에
러 < 0
 * 주의 :
 *
 *
-----------------------------------
-----------------------------------
----------
 */
int search_opcode(char* str)
{
        int an = -1;
        int i = 0;
        while(i<inst_index)
        {
                if (strcmp(inst_table[i]->str,
str) == 0)
                {
                        an = i;
                        break;
                }
                i++;
        }
        return an;
}

/                                               *
-----------------------------------
-----------------------------------
----------
* 설명 : 어셈블리 코드를 위한 패스1과정을 수행
하는 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 검사
```

```
를 하지 않고 넘어간 상태이다.
*          따라서 에러에 대한 검사 루틴을 추가해야 한다.
*
*
------------------------------------
------------------------------------
----------
*/
static int assem_pass1(void)
{
        int err = 0;
        for (int i = 0; i < line_num; i++)//token으로 쪼개줌.
        {
                i                       f (token_parsing(input_data[i]) < 0)
                {
                        err = -1;
                }
                token_line++;
        }
        return err;
}

/                                       *
------------------------------------
------------------------------------
----------
* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
*          여기서 출력되는 내용은 명령어 옆에 OPCODE가 기록된 표(과제 3번) 이다.
* 매개 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프로그램의 결과를 표준출력으로 보내어
*        화면에 출력해준다.
*          또한 과제 3번에서만 쓰이는 함수이므로 이후의 프로젝트에서는 사용되지 않는다.
*
------------------------------------
------------------------------------
----------
*/
void make_opcode_output(char* file_name)
{
        if (*file_name == NULL)//파일명이 NULL값일땐 CMD창에 출력
        {
                for (int i = 0; i < token_line; i++)
                {
                        i                       f (token_table[i]->label != NULL)
                        {
                                printf("%s", token_table[i]->label);

                        }
                        printf("₩t");
                        i                       f (token_table[i]->operator!=NULL)
                        {
                                printf("%s", token_table[i]->operator);
                        }
                        printf("₩t");
                        i                       f (token_table[i]->operand[0][0] != "₩0")
                        {
                                for (int j = 0; j < 3; j++)
                                {
                                        i       f (token_table[i]->operand[j][0] != "₩0")
                                        {

                        p r i n t f ( " % s " , token_table[i]->operand[j]);
                                        }

                                }
                        }
                        printf("₩t");
                        int a = -1;
                        i                       f (token_table[i]->operator!=NULL)
```

```
                {
                        a = search_opcode(token_table[i]->operator);
                        if (a >= 0)//opcode가 존재한다면
                        {
                                printf("%02X", inst_table[a]->op);
                        }
                        else if (strcmp("START",token_table[i]->operator)==0)
                        {
                                printf("00");
                        }
                }
                printf("₩n");

        }
    }
    else
    {
        FILE* f = fopen(file_name, "w");
        for (int i = 0; i < token_line; i++)
        {
                if (token_table[i]->label != NULL)
                {
                        fwrite(token_table[i]->label, sizeof(char) * strlen(token_table[i]->label), 1, f);
                }
                fwrite("₩t", sizeof("₩t"), 1, f);
                if (token_table[i]->operator!=NULL)
                {
                        fwrite(token_table[i]->operator,sizeof(char) * strlen(token_table[i]->operator), 1, f);
                }
                fwrite("₩t", sizeof("₩t"), 1, f);
                if (token_table[i]->operand[0][0] != "₩0")
                {
                        for (int j = 0; j < 3; j++)
                        {
                                if (token_table[i]->operand[j][0] != "₩0")
                                {
                                        fwrite(token_table[i]->operand[j], sizeof(char) * strlen(token_table[i]->operand[j]), 1, f);
                                }
                        }
                }
                fwrite("₩t", sizeof("₩t"), 1, f);
                int a = -1;
                if (token_table[i]->operator!=NULL)
                {
                        a = search_opcode(token_table[i]->operator);
                        if (a >= 0)
                        {
                                fprintf(f, "%02X", inst_table[a]->op);
                        }
                        else if (strcmp("START", token_table[i]->operator) == 0)
                        {
                                fwrite("00", sizeof("00"), 1, f);
                        }
```

```
                        }

                        f w r i t e ( " ₩ n " ,
sizeof("₩n"), 1, f);
                }
                fclose(f);
        }
}
```