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	SIC/XE 어셈블러 구현 (Project #1)
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20163340 강원경 (출석번호 107번)
제출일	2021년 05월 05일

차 례

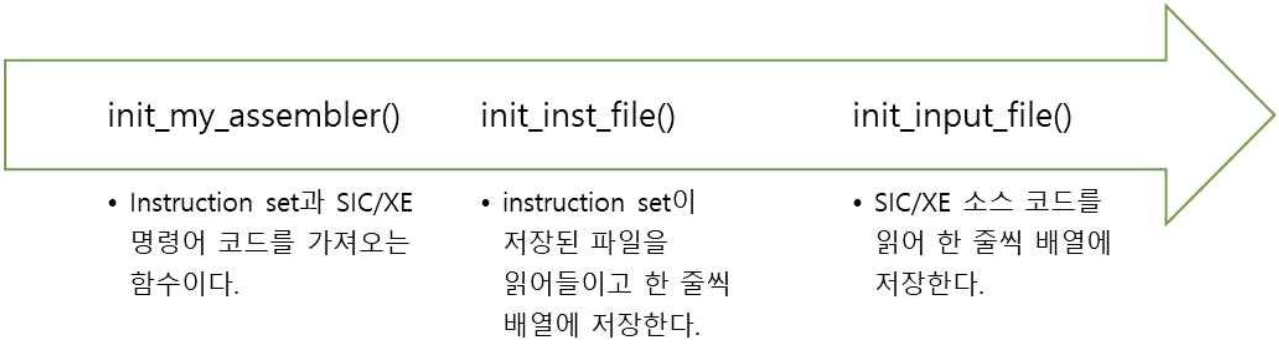
1장. 프로젝트 동기/목적

이번 프로젝트는 우리가 2021년 1학기 '시스템 프로그래밍' 시간에 배운 SIC/XE 머신의 어셈블러 프로그램 을 구현하는 것이다. 지난번 과제 3을 통해 명령어 코드를 받아들여 그 명령어의 op코드를 파싱하는 부분을 구현하였는데, 이번 프로젝트는 그 과제의 연장선으로 직접 object program을 만들어주는 프로그램을 구현하는 것이다.

수업시간에 개념적인 내용을 듣는 것은 어렵지 않다. 하지만 그 내용을 동작 구조까지 완벽하기 이해하기 위해서는 수업 이외의 본인이 직접 여러 가지 활동을 함으로써 받아들이는 것이 제일 빠르고 자세히 익힐 수 있다고 생각한다. 그렇기에 이번 과제를 통해 수업 시간에 배웠던 여러 가지 내용, 즉, Direct Addressing, Indirect Addressing, Immediate Addressing과 같은 Addressing의 방법과 Literal과 Symbol의 처리, 그리고 Control Section이 여러개 있을때의 처리 방법을 다시 한번 떠올리며 프로그램을 구현하기로 했다.

2장. 설계/구현 아이디어

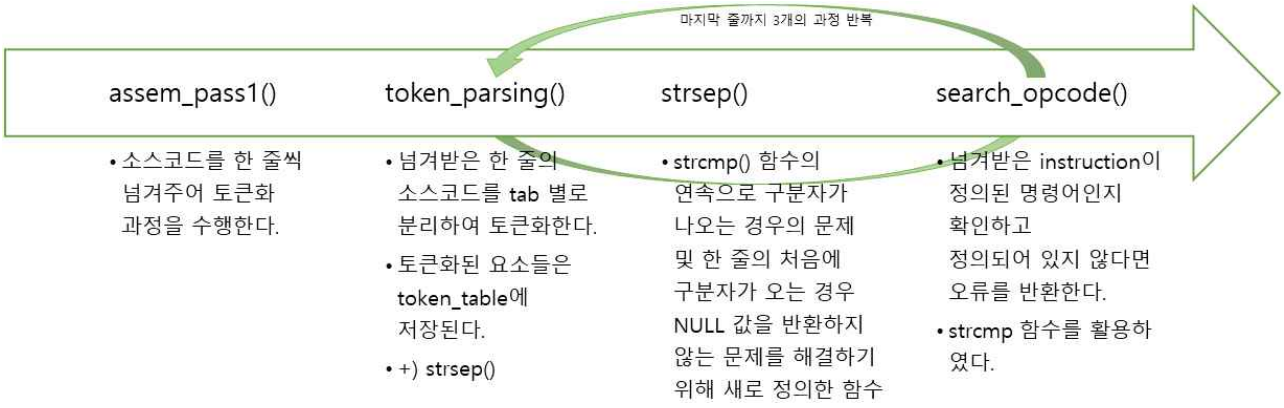
구현한 코드를 프로세스별로 살펴보도록 한다. 우선, 초기의 코드 파싱에 대한 부분이다.



init_inst_file 함수는 inst_table에 각각의 instruction을 저장하고, init_input_file은 input_data 배열에 각각의 줄을 string의 형태로 저장한다.

지난 과제에서는 4형식 명령어의 경우 3형식 명령어와는 별개의 명령어로 취급하여 inst.data에 저장했었으나(즉, 3형식과 4형식 모두를 가질 수 있는 명령어는 3형식일 때 1번, 4형식일 때 1번 해서 총 2번이 저장되어있는 방식), 이번 프로젝트에서는 operand의 첫 번째 문자가 '+'인지 아닌지로 판단하여 3형식과 4형식을 판단하는 방법으로 바꿨고, 그에 따라 각각의 명령어는 한 번씩만 inst.data에 저장되어있다.

다음은 토큰 파싱에 대한 부분이다.



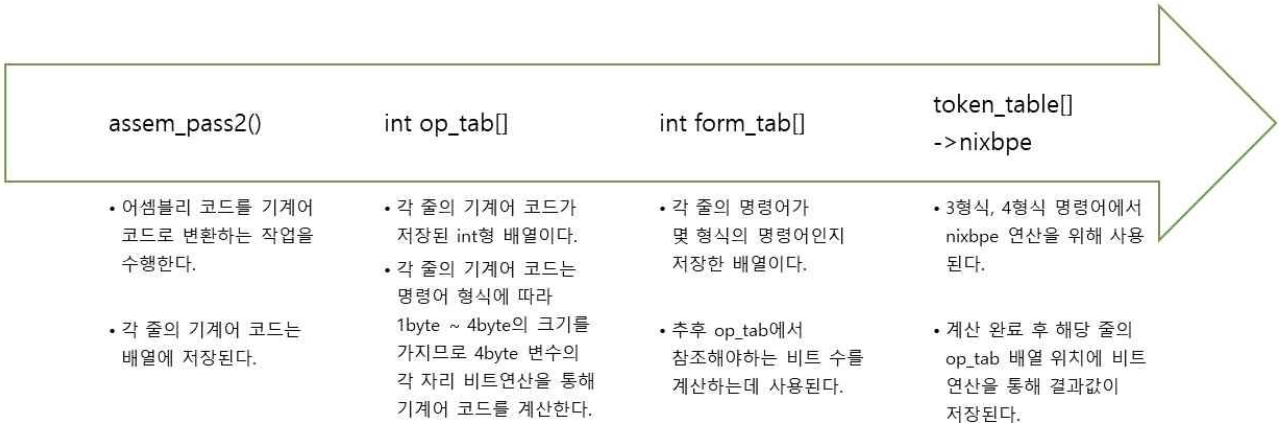
지난 과제에서 input.txt 파일의 토큰들이 '₩t' (탭)로 구분되어있었고, 대부분의 명령어가 string으로 되어있는 점에서 strtok 함수를 이용했으나, 연속 구분자에 대한 부분이나 해당 영역의 토큰이 없는 경우 NULL값을 반환하는 것이 아니라 그 연속된 구분자들을 무시해버리는 문제가 있었다. 따라서 이번 과제에서는 strtok 대신, Linux-C 기반의 strsep를 직접 구현하여 그 역할을 대신하기로 했다.

여기까지는 지난 과제 3에 구현한 내용이지만, 이번 프로젝트를 진행하면서 지난 과제에서 작성한 코드의 한계점을 수정해야 할 부분이 있어, 필요한 부분에 대해서는 재정의하거나 함수를 추가하는 과정을 진행하였다. 이 다음부터는 이번 프로젝트에 주요 부분으로 볼 수 있는 프로세스들이다.

다음으로 살펴볼 부분은 심볼 테이블과 리터럴 테이블을 만드는 부분이다.

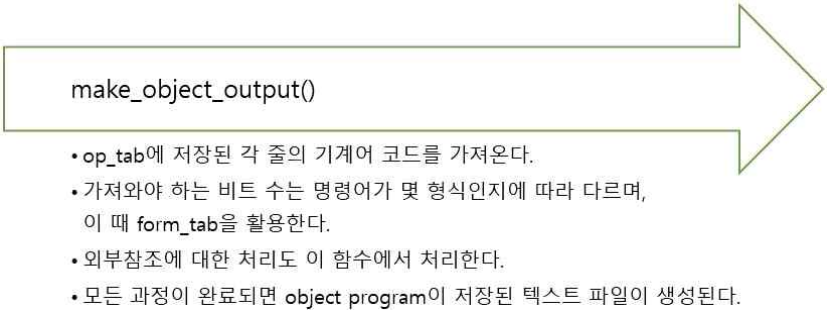
본인이 작성한 프로그램은 pass1 과정에서 토큰 파싱이 끝난 후 바로 이어서 symbol 테이블과 리터럴 테이블을 만드는 프로세스를 작성하였다. 이 과정이 정상적으로 완료되면 main 함수로 정상 종료 리턴값인 0을 리턴하고, main 함수 부분에서 뒤이어 make_symtab_output() 함수와 make_literaltab_output()함수를 호출하여 심볼 테이블과 리터럴 테이블에 저장된 값들을 각각의 파일로 만들어 생성하고, 콘솔창에도 해당 테이블을 표시한다. 생성되는 파일 명은 각각 symtab_20163340 (본인의 학번), literaltab_107 (본인의 출석번호)이다.

다음은 기계어 코드를 만드는 pass2에 대한 과정이다.



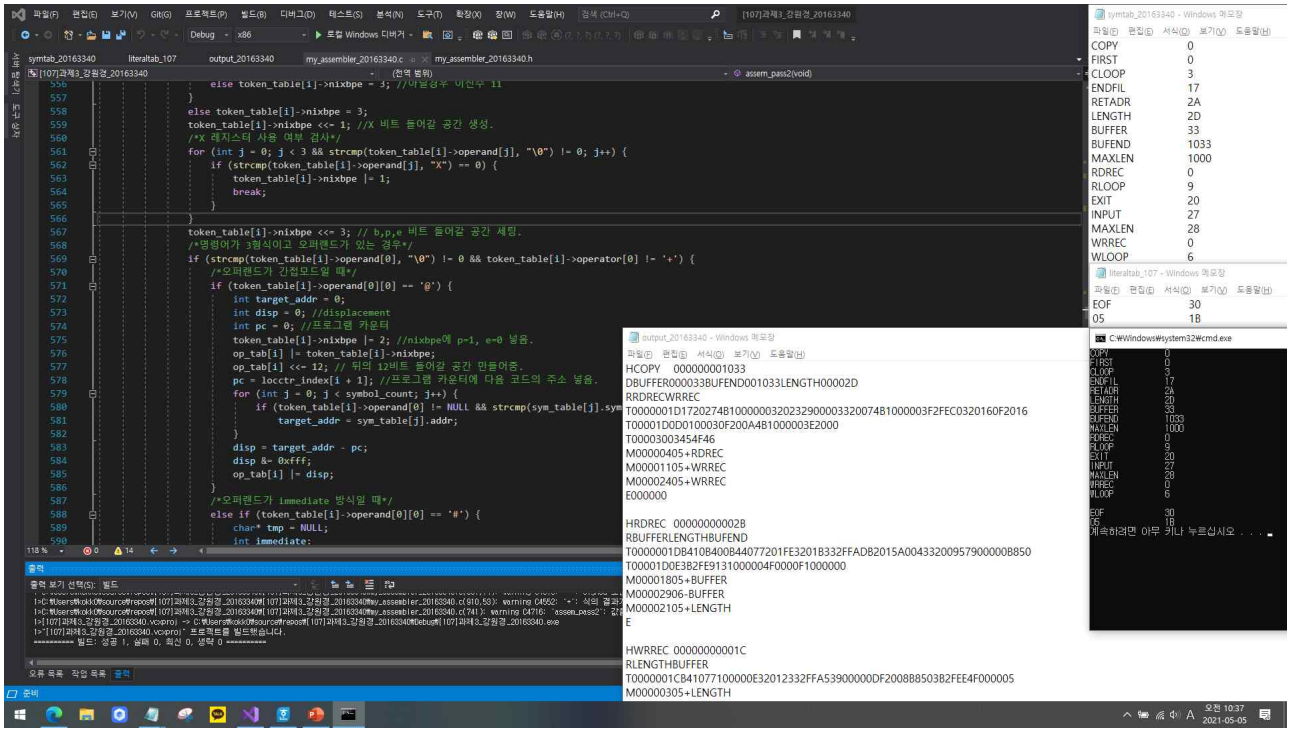
nixbpe는 1형식과 2형식 명령어에서는 사용할 필요가 없으므로, 이 경우에는 바로 op_tab 배열의 원소와 직접 비트연산을 실행하고, 3형식과 4형식 명령어에서는 op_tab에 명령어 코드 저장 → nixbpe 비트 연산 → op_tab과 nixbpe의 or 연산 → op_tab에 displacement 저장 의 순서로 실행하였다.

마지막은 최종 결과 파일을 출력하는 make_object_output()에 대한 설명이다.



위의 일련의 과정을 모두 거치면 프로그램은 정상적으로 종료된다.

3장. 수행 결과



코드를 실행할 경우 위의 사진과 같이 심볼 테이블과 리터럴 테이블이 각각의 파일로 저장되고, 동일한 내용이 콘솔창에도 출력된다.

또한, 최종 결과 파일에도 프로세스를 거친 오브젝트 프로그램의 코드들이 저장되어있는 것을 확인할 수 있다.

4장. 결론 및 보충할 점.

이번 과제 수행을 통해 본인은 방대한 양의 프로그램에서 코드를 다루는 방법과 디버거를 통해 의도하지 않은 부분을 빠르게 찾아내는 방법을 습득할 수 있었다. 1000줄 가까이 되는 코드를 다뤄볼 일도 거의 없었고, 디버깅을 하더라도 직접 값을 찍어서 찾아보는 방법밖엔 하지 않았는데, 이번 과정을 통해 파일의 입출력에 대한 부분, 디버거를 다루는 부분등을 오랜시간 다루고 배운 것 같다.

또한, 이러한 설계 과정을 통해 수업시간에 배운 내용에 대해서도 다시한번 떠올리고, 잘 이해가 되지 않던 부분은 수업 교재를 참고하고 원리를 다시 이해하는 과정을 통해 어셈블러가 어떻게 동작하고 어떤 방식으로 구현해야 하는지 생각해보는 계기가 되었다.

하지만, 이번 프로젝트는 사실 완벽하지 못한 프로젝트이다. 두 가지 부분에 대한 코드가 완벽하지 못하다.

우선, 첫 번째로, 각 control section의 프로그램 길이에 대한 부분의 코드가 정상적으로 작동하지 않았다.

```
/*한 프로그램(섹션)을 시작할 때 프로그램의 이름, 시작주소 기록*/
fprintf(fp, "H%s\t", token_table[now]->label); //프로그램 이름 기록
if (strcmp(token_table[now]->operand[0], "\0") != 0)
    start_addr = atoi(token_table[now]->operand[0]);
fprintf(fp, "%06X", start_addr); // 프로그램 시작주소
/*프로그램(섹션) 하나당 몇줄의 코드가 있는지, literal은 몇개나 있는지 확인.*/
for (int i = now + 1; i < token_line; i++) {
    if (strcmp(token_table[i]->operator,"\0") == 0)
        continue;
    if (strcmp(token_table[i]->operator,"CSECT") == 0)
        break;
    else if (strcmp(token_table[i]->operator,"END") == 0) {
        end_line = i + 1;
        break;
    }
    end_line = i;
}

if (end_line == token_line)
    is_endline = 1;
temp_data = end_line;
if (end_line != token_line) {
    while (strcmp(token_table[temp_data]->operator,"EQU") == 0)
        temp_data--;

    end_addr = locctr_index[temp_data + 1];//마지막 줄의 PC 주소값.
}
else end_addr = locctr_index[temp_data];
```

해당 부분에 대한 코드인데, end_addr에 마지막 줄의 PC 주소값이 들어가야 하는데 배열에 저장되는 위치가 본인이 생각했던 대로 되지 않았던지, end_addr의 값이 0으로 반환되는 오류가 있었다. 이 부분은 결국 수동적으로 해당 control section의 코드 크기가 어느정도 되는지 직접 지정함으로써 input.txt파일에 한정된 프로그램으로 작성할 수 밖에 없었다.

두 번째 오류가 발생한 부분은 외부 참조에 대한 부분이다.

첫 번째 텍스트

```
HWRREC  00000000001C
RLENGTHBUFFER
T0000001CB41077100000E32012332FFA53900000DF2008B8503B2FEE4F000005
M00000305+LENGTH
M00000D05+BUFFER
E
```

두 번째 텍스트

```
HRDREC  00000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A00433200957900000B850
T00001D0E3B2FE9131000004F0000F1000000
M00001805+BUFFER
M00002806-BUFFER
M00002105+LENGTH
```

비교

결과

```
HCOPY 000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDRECWRREC
T0000001D1720274B10000003202329000003320074B1000003F2FEC0320160F2016
T00001D0D0100030F200A4B1000003E2000
T00003003454F46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E000000

HRDREC 00000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A00433200957900000B850
T00001D0E3B2FE9131000004F0000F1000000
M00001805+BUFFER
M00002906-BUFFER
M00002806-BUFFER
M00002105+LENGTH
M00002806+BUFEND
E

HWRREC 00000000001C
RLENGTHBUFFER
T0000001CB41077100000E32012332FFA53900000DF2008B8503B2FEE4F000005
M00000305+LENGTH
M00000D05+BUFFER
E
```

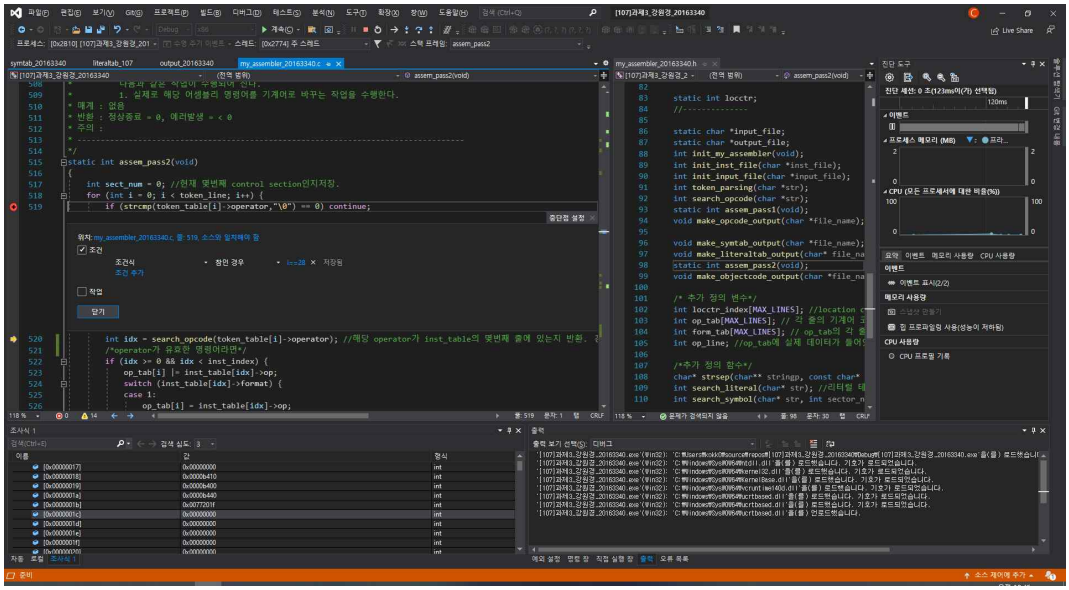
정상적으로 작성된 object 프로그램과 본인이 작성한 프로그램의 결과로 나온 object program의 텍스트 값을 비교해보면, 2번째 control section의 BUFFEND-BUFFER에 대한 부분이 정상적으로 출력되지 않음이 확인된다.

이 부분은 제출 직전까지도 계속 디버깅을 하였으나, 끝내 잡아내지 못하였다.

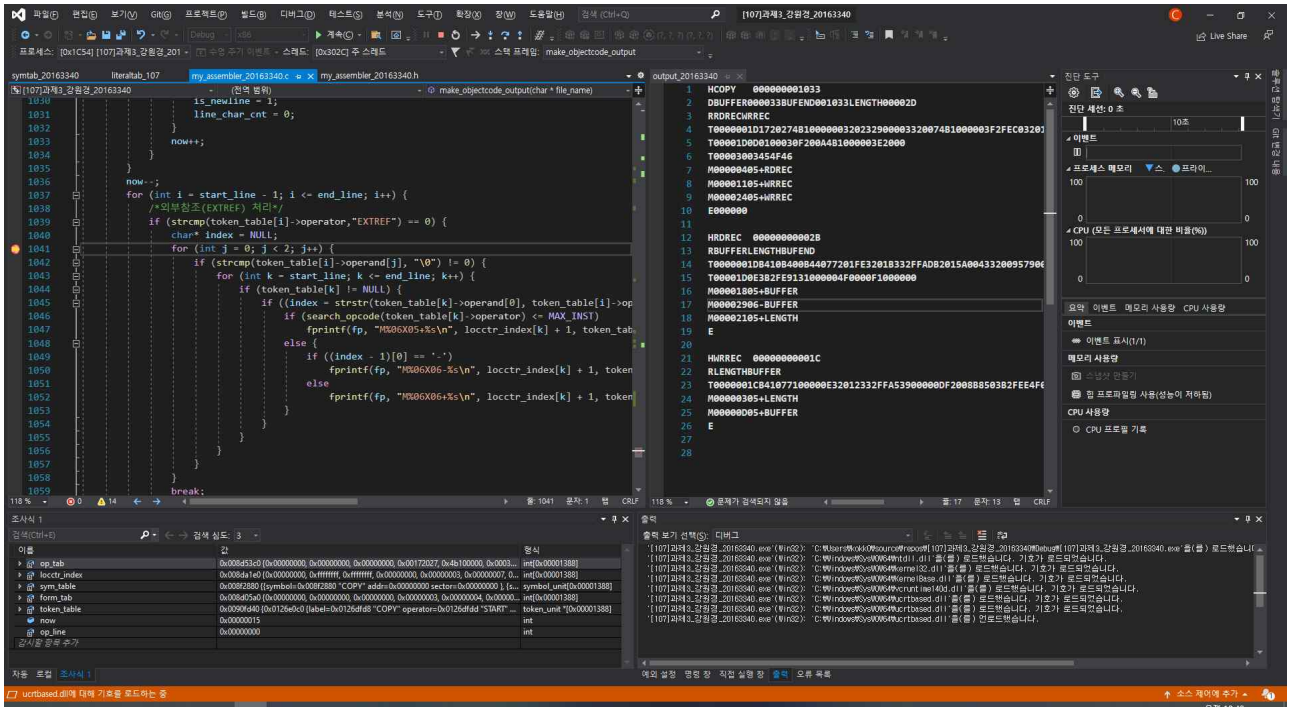
프로젝트를 마무리하면서 고치지 못한 두 부분에 대해서는 상당히 많은 아쉬움이 남았으나, 아직 내가 많이 부족하다는 것을 깨달을 수 있는 기회였고, 또 이렇게 큰 코드를 다루면서 어떤 방식으로 어떻게 프로그램을 작성하는 계획을 세워야 하는지에 대해 고민해보고 직접 부딪혀보는 좋은 계기가 되었다.

5장. 디버깅

지난번 과제와 마찬가지로 내가 의도하지 않은 동작에 대해서는 디버거를 통해 결과를 확인하였다. 이번에는 지난번 디버깅에서 한 발 더 나아가, 중단점 조건을 걸어 본인이 의도하지 않은 결과가 나오는 위치를 집중적으로 살펴보는 과정도 수행하였다.



이 사진은 op_tab에 기계어 코드가 본인이 의도한 대로 들어가지 않아 중단점을 설정하고 조사식 설정을 통해 op_tab의 각각의 원소에 어떤 내용이 저장되는지 살펴보는 과정이다.



마지막 외부 참조 부분에 대해 값이 이상하게 저장되어 디버거를 연 모습.
결국 이 버그는 잡지 못했으나, 어느 부분에서 오류가 발생하는지는 알 수 있었다.

6장. 소스코드(+주석)

```
#define _CRT_SECURE_NO_WARNINGS // 일부 함수의 보안 에러 무시용
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include "my_assembler_20163340.h"


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

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

 //make_opcode_output("output_20163340");

 make_symtab_output("symtab_20163340");
 make_literaltab_output("literaltab_107");

 if (assem_pass2() < 0)
 {
  printf(" assem_pass2: 패스2 과정에서 실패하였습니다.  ₩n");
  return -1;
 }
 make_objectcode_output("output_20163340");

 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
  위해 파일 단위로 관리하여 프로그램 초기화를 통해 정보를 읽어 올 수
```

```
있도록
  구현하였다.
 *
-------------------------------------------------------
 */
int init_my_assembler(void) {
 int result = 0;
 if ((result = init_inst_file("inst.data")) < 0)
  return -1;
 if ((result = init_input_file("input.txt")) < 0)
  return -1;
 return result;
}


/*
-------------------------------------------------------
 * 설명 : 명령어 셋이 저장된 inst.data 파일을 한줄씩 inst_table에
저장한다.
 * 매계 : 기계어 목록 파일 (여기서는 inst.data)
 * 반환 : 정상종료 = 0 , 에러 = -1
 * 주의 : inst.data 파일은 이름, 형식, op코드, 오퍼랜드 갯수의 순으로
  띄어쓰기로 구분하여 작성되어있다고 가정한다.
 *
-------------------------------------------------------
 */
int init_inst_file(char* inst_file) {
 FILE* file = fopen(inst_file, "r"); // 읽기 모드로 연 기계어 목록 파일의
파일 포인터
 int errno; // 코드를 수행하면서 정상종료 될경우 0, 아닐 경우 -1이
저장되며, 이 값은 함수 종료시 리턴된다.

 if (file == NULL)
  errno = -1; // 파일을 읽지 못했을 경우 -1이 저장된다.
 else {
  inst* temp; // 한 줄 입력받은 내용을 임시로 저장할 구조체.
  while (feof(file) == 0) {
   temp = (inst*)malloc(sizeof(inst));
   fflush(stdin);
   char c[2]; //format이 이상하게 읽히는 버그 해결용 임시 변수
   fscanf(file, "%s₩t %c₩t %x₩t %d₩n", &temp->str, &c,
&temp->op, &temp->ops);
   temp->format = atoi(c);
   inst_table[inst_index++] = temp; // 한 줄의 내용을 테이블에
저장한다.
  }
  errno = 0; // 여기까지 반복 수행했다면 정상적인 파일임으로 리턴값
0을 저장.
 }
 fclose(file);
 return errno;
}

/*
-------------------------------------------------------
 * 설명 : 어셈블리 할 소스코드를 읽어 소스코드 테이블(input_data)를
```

```
생성하는 함수.
 * 매계 : 어셈블리할 소스파일명 (여기서는 input.txt)
 * 반환 : 정상종료 = 0 , 에러 = -1
 * 주의 : 라인단위로 저장한다.
 *
----------------------------------------------------------
 */
int init_input_file(char* input_file) {
 FILE* file = fopen(input_file, "rt"); // 읽기 모드로 연 소스코드 파일의 파일 포인터
 int errno;

 if (file == NULL) {
  printf("inst_file를 찾을 수 없습니다.₩n");
  errno = -1;
 }
 else {
  line_num = 0;
  char* buffer; // 한 줄 입력받은 내용을 임시로 저장할 char 배열.
  while (1) {
   buffer = (char*)malloc(sizeof(char) * 100);
   memset(buffer, 0, sizeof(buffer)); // 쓰레기값 저장 방지를 위해 0으로 initialize
   fgets(buffer, 100, file); // 라인 단위로 코드를 읽어온다.
   if (feof(file)) break;
   input_data[line_num] = buffer; // 읽어온 한 줄을 소스코드 테이블에 저장한다.
   line_num++; // 다음줄을 가리키기 위해 index값 1 증가.
  }
  errno = 0; // 여기까지 반복 수행했다면 정상적인 파일임으로 리턴값 0을 저장.
 }
 fclose(file);
 return errno; // 리턴 결과값으로 수행결과를 호출함수에 전달.
}

/*
----------------------------------------------------------
 * 설명 : 소스 코드를 한줄씩 읽어와 토큰단위로 분석하고 토큰 테이블을 작성하는 함수이다.
    패스 1 (assem_pass1)로 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[0] == '.')
  return 0;
 char* line_tmp = (char*)malloc(strlen(str) + 2);
 token* tok_temp = (token*)malloc(sizeof(token)); // 한 줄의 소스코드를 분석한 토큰들을 임시로 저장할 변수.
 char* tok[4] = { '₩0', };
 for (int i = 0; i < 4; i++) {
```

```
  tok[i] = strsep(&str, "₩t₩n");
  if (tok[i] == NULL)
   break;
 }

 /* 유효한 기계어인지 확인하기 위해 search_opcode 호출 */
 if (search_opcode(tok[1]) < 0) {
  printf("유효하지 않은 opcode입니다.₩n");
  return -1;
 }

 tok_temp->label = tok[0];
 tok_temp->operator = tok[1];
 if (tok[3] != NULL)
  strcpy(tok_temp->comment, tok[3]); // 코멘트를 테이블에 저장.

 char* optmp = strtok(tok[2], ","); // ,별로 구분한 operand를 임시로
저장하는 변수.
 for (int i = 0; i < MAX_OPERAND; i++) {
  if (optmp == NULL) { // 분리된 내용이 없을경우 i번째 operand
배열에 NULL문자열 저장.
   strcpy(tok_temp->operand[i], "₩0");
  }
  else { // 분리된 내용이 있을 경우 i번째 operand 배열에 분리한
operand 저장.
   strcpy(tok_temp->operand[i], optmp);
  }
  optmp = strtok(NULL, ",");
 }

 token_table[token_line] = tok_temp; // 한 줄의 소스코드를 토큰화해
저장한 구조체를 토큰테이블에 저장.
 token_line++; // 다음줄의 소스코드를 다음칸의 배열에 저장하기 위해
인덱스 증가.
 return 0;
}

/*
------------------------------------------------------------
 * 설명 : 입력 문자열이 기계어 코드인지를 검사하는 함수이다.
 * 매개 : 토큰 단위로 구분된 문자열
 * 반환 : 정상종료 = 기계어 테이블 인덱스 번호, 에러 = -1, 지시어 =
MAX_INST+1
 * 주의 : 지시어는 기계어 테이블 인덱스에 저장되어있지 않기 때문에
함수 안에 문자열 배열 형태로 저장됨.
 *
------------------------------------------------------------
 */
int search_opcode(char* str) {
 /* 몇번째 기계어 테이블 인덱스인지 저장하는 변수.
 기본적으로 오류값에 해당하는 -1이 저장되어있다. */
 int ans = -1;
 char directives[][7] = {
"START","END","BYTE","WORD","RESB","RESW","EXTDEF","EXTREF",
"LTORG", "EQU", "CSECT" }; // 지시어를 저장한 문자열 배열이다.
 for (int i = 0; i < sizeof(directives) / 7; i++) {
```

```
  if (strcmp(str, directives[i]) == 0) {
   ans = MAX_INST + 1; // 지시어 배열에 있다면 MAX_INST+1
넘겨줌.
   break;
  }
 }
 // 지시어 배열에 없다면 ans값은 -1 이므로 기계어 테이블에 있는지
검사한다.
 if (ans == -1) {
  int len = strlen(str);
  char* temp_inst = (char*)malloc(len + 1);
  if (str[0] == '+') { // 4형식 명령어 앞의 + 무시하기 위한 코드
   strncpy(temp_inst, str + 1, sizeof(char) * len);
  }
  else strcpy(temp_inst, str);


  for (int i = 0; i < inst_index; i++) {
   if (strcmp(temp_inst, inst_table[i]->str) == 0) {
    ans = i; //들어온 토큰 string이 inst 테이블에 있으면 인덱스 반환
변수에 저장 및 탈출.
    break;
   }
  }
 }
 /*
 기계어 테이블에 있거나 지시어 배열에 있다면 해당하는 값이
저장되어있을 것이고,
  아닐 경우 초기값이자 오류값에 해당하는 -1이 그대로 저장되어있을
것이다.
  */
 return ans;
}

/*
-------------------------------------------------------------
* 설명 : 어셈블리 코드를 위한 패스1과정을 수행하는 함수이다.
*  패스1에서는..
*  1. 프로그램 소스를 스캔하여 해당하는 토큰단위로 분리하여 프로그램
라인별 토큰
*  테이블을 생성한다.
*  2. locctr을 통해 해당 줄의 코드가 어느 위치의 주소를 갖게 되는지
계산한다.
*  3. 각 줄의 locctr 값을 locctr_index 배열에 저장한다.
*
*
* 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) {
```

```
token_line = 0;

/* input_data의 문자열을 한줄씩 입력 받아서 인자로 취급하고
 * token_parsing()에게 넘겨주어 코드를 실행함으로써 token_unit에 저장
 */
for (int i = 0; i < line_num; i++) {
 if (token_parsing(input_data[i]) < 0)
  return -1;
}
symbol_count = 0; //symtab의 줄 초기화
locctr = 0;
int sect_num = 0;
for (int i = 0; i < token_line; i++) {
 token* temp = token_table[i];
 if (strcmp(temp->label, "₩0") != 0) { //심볼 테이블 저장
  if (search_symbol(temp->label, sect_num) != 0) {
   strcpy(sym_table[symbol_count].symbol, temp->label);
   sym_table[symbol_count].addr = locctr;
   sym_table[symbol_count].sector = sect_num;
   symbol_count++;
  }
 }
 if (temp->operand[0][0] == '=') { //literal table에 저장
  int lit_len = strlen(temp->operand[0]);
  char* lit_temp = (char*)calloc(lit_len + 1, sizeof(char));
  strncpy(lit_temp, temp->operand[0] + 3, lit_len - 4);
  if (search_literal(lit_temp) != 0) {
   literal_table[literal_count].literal = (char*)calloc(lit_len + 1,
sizeof(char));
   strcpy(literal_table[literal_count].literal, lit_temp);
   literal_table[literal_count].addr = -1;
   if (temp->operand[0][1] == 'C')
    literal_table[literal_count].kind = 1;
   else literal_table[literal_count].kind = 0;
   literal_table[literal_count].sector = sect_num;
   literal_count++;
  }
 }
 if (temp->operator[0] == '+') { //4형식 operator인지 확인
  locctr_index[i] = locctr;
  locctr += 4;
 }
 else {
  int indx = search_opcode(temp->operator);
  if (indx < MAX_INST) {
   locctr_index[i] = locctr;
   locctr += inst_table[indx]->format;
  }
  else {
   if (strcmp(temp->operator,"RESW") == 0) {
    locctr_index[i] = locctr;
    int nums = atoi(token_table[i]->operand[0]);
    locctr += nums * 3;
   }
   else if (strcmp(temp->operator,"RESB") == 0) {
    locctr_index[i] = locctr;
```

```
 int nums = atoi(token_table[i]->operand[0]);
 locctr += nums;
}
else if (strcmp(temp->operator, "EXTDEF") == 0
 || strcmp(temp->operator, "EXTREF") == 0) {
 locctr_index[i] = -1; //location counter 필요없음
}
else if (strcmp(temp->operator, "LTORG") == 0 ||
 strcmp(temp->operator,"END") == 0) {
 for (int j = 0; j < literal_count; j++) {
  if (literal_table[j].addr == -1) {
   int len;
   if (literal_table[j].kind == 1)
    len = strlen(literal_table[j].literal);
   else len = strlen(literal_table[j].literal) / 2;
   literal_table[j].addr = locctr;
   locctr_index[i] = locctr;
   locctr += len;
  }
 }
}


else if (strcmp(temp->operator,"CSECT") == 0) {
 locctr_index[i] = locctr;
}
else if (strcmp(temp->operator,"EQU") == 0) {
 if (token_table[i]->operand[0][0] != '*') {
  char* tmp;
  char* tmp_token;
  char operation[2] = { 0, };//변수의 연산자를 저장하는데, strtok
사용 위해 문자배열로 생성.
  int v1, v2;
  tmp = (char*)malloc(strlen(temp->operand[0]) + 1);
  strcpy(tmp, temp->operand[0]);
  for (int j = 0; temp->operand[0][j] != '₩0'; j++) {
   if (temp->operand[0][j] == '+' ||
    temp->operand[0][j] == '-') {
    operation[0] = token_table[i]->operand[0][j];
    break;
   }
  }
  tmp_token = strtok(tmp, operation);
  //symtab에서 주소값 가져오기 위한 과정
  for (int k = 0; k < symbol_count; k++) {
   if (strcmp(tmp_token, sym_table[k].symbol) == 0)
    v1 = sym_table[k].addr;
  }
  tmp_token = strtok(NULL, "₩0");
  for (int k = 0; k < symbol_count; k++) {
   if (strcmp(tmp_token, sym_table[k].symbol) == 0)
    v2 = sym_table[k].addr;
  }
  if (operation[0] == '+') locctr = v1 + v2;
  else locctr = v1 - v2;
 }
 locctr_index[i] = locctr;
```

```
 for (int j = 0; j < symbol_count; j++) {
  if (strcmp(temp->label, sym_table[j].symbol) == 0) {
   sym_table[j].addr = locctr;
  }
 }
}
else if (strcmp(temp->operator,"START") == 0)
 locctr_index[i] = locctr;
else if (strcmp(temp->operator,"*") == 0) {
 locctr_index[i] = locctr;
 if (temp->operand[0][1] == 'C') {
  int ch_num = 0;
  for (int j = 3; temp->operand[0][j] != '₩'; j++)
   ch_num++;
  locctr += ch_num;
 }
 else if (temp->operand[0][1] == 'X') {
  int x_num = 0;
  for (int j = 3; temp->operand[0][j] != '₩'; j++)
   x_num++;
  locctr += (x_num / 2);
 }
 locctr_index[i + 1] = locctr;
}
else if (strcmp(temp->operator,"BYTE") == 0) {
 locctr_index[i] = locctr;
 if (temp->operand[0][0] == 'C') {
  int ch_num = 0;
  for (int j = 2; temp->operand[0][j] != '₩'; j++)
   ch_num++;
  locctr += ch_num;
 }
 else if (temp->operand[0][0] == 'X') {
  int x_num = 0;
  for (int j = 2; temp->operand[0][j] != '₩'; j++)
   x_num++;
  locctr += (x_num) / 2;
 }
}
else if (strcmp(temp->operator, "WORD") == 0) {
 locctr_index[i] = locctr;
 locctr += 3;
}
}
}

 if (i + 1 < token_line && strcmp(token_table[i +
1]->operator,"CSECT") == 0) {
  sect_num++;
  locctr = 0;
 }
}

return 0;
}
```

```
/*
----------------------------------------------------------
* 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 3번에서만 쓰이는 함수이므로 이후의 프로젝트에서는 사용되지 않는다.
*
----------------------------------------------------------
*/
void make_opcode_output(char* file_name) {
 if (file_name == NULL) {
  for (int i = 0; i < token_line; i++) {
   token* tmp = token_table[i]; // 한 라인의 토큰들을 불러와 임시로 저장한 함수.
   printf("%s₩t%s₩t", tmp->label, tmp->operator); // label과 operator 출력.
   for (int j = 0; j < MAX_OPERAND; j++) {
    printf("%s", tmp->operand[j]);
    /* operand가 더 있다면 ','를 출력하고 아닐 경우 tab만큼 띈 후 for문 탈출. */
    if (j + 1 < MAX_OPERAND && strcmp(tmp->operand[j + 1], "₩0") != 0) {
     printf(",");
    }
    else {
     printf("₩t");
     break;
    }
   }
   int index = search_opcode(token_table[i]->operator); // 기계어 테이블 인덱스값이 저장되는 변수.
   if (index == MAX_INST + 1) // 지시어일 경우 inst_table을 찾을 필요가 없으므로 tab 출력 후 다음줄로.
    printf("₩t₩n");
   else { // 아닐경우 op코드 출력.
    if (inst_table[index]->op < 16) // 16보다 작을 경우 한자리밖에 안나오므로 0 padding 필요한지 여부 확인
     printf("0%X₩n", inst_table[index]->op); // 16 미만이면 0 패딩 필요함.
    else printf("%X₩n", inst_table[index]->op); // 16 이상이면 0 패딩 필요 없음.
   }
  }
 }
 else {
  FILE* fp = fopen(file_name, "wt"); // 분석결과를 저장할 파일을 Write Text 모드로 연다.
  for (int i = 0; i < token_line; i++) {
```

```
   token* tmp = token_table[i]; // 한 라인의 토큰들을 불러와 임시로 저장한 함수.
   fprintf(fp, "%s₩t%s₩t", tmp->label, tmp->operator); // label과 operator를 파일에 씀.
   for (int j = 0; j < MAX_OPERAND; j++) {
    fprintf(fp, "%s", tmp->operand[j]);
    /* operand가 더 있다면 ','를 저장하고 아닐 경우 tab만큼 띄어 쓰고 for문 탈출. */
    if (j + 1 < MAX_OPERAND && strcmp(tmp->operand[j + 1], "₩0") != 0) {
     fprintf(fp, ",");
    }
    else {
     fprintf(fp, "₩t");
     break;
    }
   }
   int index = search_opcode(token_table[i]->operator); // 기계어 테이블 인덱스값이 저장되는 변수.
   if (index == MAX_INST + 1) {// 지시어일 경우 inst_table을 찾을 필요가 없으므로 tab 출력 후 다음줄로.
    fprintf(fp, "₩t₩n");
   }
   else {
    if (inst_table[index]->op < 16) // 16보다 작을 경우 한자리밖에 안나오므로 0 padding 필요한지 여부 확인
     fprintf(fp, "0%X₩n", inst_table[index]->op); // 16 미만이면 0 패딩 필요함.
    else fprintf(fp, "%X₩n", inst_table[index]->op); // 16 이상이면 0 패딩 필요 없음.
   }
  }
  fclose(fp);
 }
}
/*
-------------------------------------------------------------
* 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 프로그램의 결과를 표준출력으로 보내어
*        화면에 출력해준다.
*
*
-------------------------------------------------------------
*/
void make_symtab_output(char* file_name)
{
 FILE* fp = fopen(file_name, "wt");
 for (int i = 0; i < symbol_count; i++) {
  fprintf(fp, "%s₩t₩t%X₩n", sym_table[i].symbol, sym_table[i].addr);
  printf("%s₩t₩t%X₩n", sym_table[i].symbol, sym_table[i].addr);
```

```
 }
 fclose(fp);

 printf("₩n");
 return;
}

/*
------------------------------------------------------------
* 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 프로그램의 결과를
표준출력으로 보내어
*        화면에 출력해준다.
*
*
------------------------------------------------------------
*/
void make_literaltab_output(char* file_name)
{
 FILE* fp = fopen(file_name, "wt");
 for (int i = 0; i < literal_count; i++) {
  fprintf(fp, "%s₩t₩t%X₩n", literal_table[i].literal,
literal_table[i].addr);
  printf("%s₩t₩t%X₩n", literal_table[i].literal, literal_table[i].addr);
 }
 fclose(fp);
 return;
}

/*
------------------------------------------------------------
* 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 sect_num = 0; //현재 몇번째 control section인지저장.
 for (int i = 0; i < token_line; i++) {
  if (strcmp(token_table[i]->operator,"₩0") == 0) continue;
  int idx = search_opcode(token_table[i]->operator); //해당 operator가
inst_table의 몇번째 줄에 있는지 반환.
  /*operator가 유효한 명령어라면*/
```

```
  if (idx >= 0 && idx < inst_index) {
   op_tab[i] |= inst_table[idx]->op;
   switch (inst_table[idx]->format) {
   case 1:
    op_tab[i] = inst_table[idx]->op;
    form_tab[i] = 1;
    break;
   case 2:
    op_tab[i] = inst_table[idx]->op;
    for (int j = 0; j < 2; j++) {
     op_tab[i] <<= 4; //1바이트(4bit)만큼 shift left
     if (j == 1 && strcmp(token_table[i]->operand[j], "₩0") == 0)
op_tab[i] |= 0;
     else if (strcmp(token_table[i]->operand[j], "SW") == 0) op_tab[i]
|= 9;
     else if (strcmp(token_table[i]->operand[j], "PC") == 0) op_tab[i]
|= 8;
     else if (strcmp(token_table[i]->operand[j], "F") == 0) op_tab[i]
|= 6;
     else if (strcmp(token_table[i]->operand[j], "T") == 0) op_tab[i]
|= 5;
     else if (strcmp(token_table[i]->operand[j], "S") == 0) op_tab[i]
|= 4;
     else if (strcmp(token_table[i]->operand[j], "B") == 0) op_tab[i]
|= 3;
     else if (strcmp(token_table[i]->operand[j], "L") == 0) op_tab[i]
|= 2;
     else if (strcmp(token_table[i]->operand[j], "X") == 0) op_tab[i]
|= 1;
     else if (strcmp(token_table[i]->operand[j], "A") == 0) op_tab[i]
|= 0;
    }
    form_tab[i] = 2;
    break;
   case 3:
    op_tab[i] = inst_table[idx]->op;
    form_tab[i] = 3;
    op_tab[i] <<= 4; //nixbpe 들어가기 위해 4비트만큼 옆으로 shift
    if (strcmp(token_table[i]->operand[0], "₩0") != 0) {
     /*direct, indirect, 지정*/
     if (token_table[i]->operand[0][0] == '@')
      token_table[i]->nixbpe = 2; //간접주소일때 이진수 10
     else if (token_table[i]->operand[0][0] == '#')
      token_table[i]->nixbpe = 1; //immediate일때 이진수 01
     else token_table[i]->nixbpe = 3; //아닐경우 이진수 11
    }
    else token_table[i]->nixbpe = 3;
    token_table[i]->nixbpe <<= 1; //X 비트 들어갈 공간 생성.
    /*X 레지스터 사용 여부 검사*/
    for (int j = 0; j < 3 && strcmp(token_table[i]->operand[j], "₩0")
!= 0; j++) {
     if (strcmp(token_table[i]->operand[j], "X") == 0) {
      token_table[i]->nixbpe |= 1;
      break;
     }
    }
```

```
    token_table[i]->nixbpe <<= 3; // b,p,e 비트 들어갈 공간 세팅.
    /*명령어가 3형식이고 오퍼랜드가 있는 경우*/
    if (strcmp(token_table[i]->operand[0], "₩0") != 0 &&
token_table[i]->operator[0] != '+') {
     /*오퍼랜드가 간접모드일 때*/
     if (token_table[i]->operand[0][0] == '@') {
      int target_addr = 0;
      int disp = 0; //displacement
      int pc = 0; //프로그램 카운터
      token_table[i]->nixbpe |= 2; //nixbpe에 p=1, e=0 넣음.
      op_tab[i] |= token_table[i]->nixbpe;
      op_tab[i] <<= 12; // 뒤의 12비트 들어갈 공간 만들어줌.
      pc = locctr_index[i + 1]; //프로그램 카운터에 다음 코드의 주소
넣음.
      for (int j = 0; j < symbol_count; j++) {
       if (token_table[i]->operand[0] != NULL &&
strcmp(sym_table[j].symbol, token_table[i]->operand[0] + 1) == 0)
         target_addr = sym_table[j].addr;
      }
      disp = target_addr - pc;
      disp &= 0xfff;
      op_tab[i] |= disp;
     }
     /*오퍼랜드가 immediate 방식일 때*/
     else if (token_table[i]->operand[0][0] == '#') {
      char* tmp = NULL;
      int immediate;
      int idx = 0;
      op_tab[i] |= token_table[i]->nixbpe;
      op_tab[i] <<= 12;
      tmp = (char*)calloc(strlen(token_table[i]->operand[0]), 1);
      for (int j = 0; j < strlen(token_table[i]->operand[0]) - 1; j++)
{
       tmp[idx++] = token_table[i]->operand[0][j + 1];
      }
      tmp[idx] = '₩0';
      immediate = strtol(tmp, NULL, 10);
      op_tab[i] |= immediate;
      free(tmp);
     }
     /*리터럴일 때*/
     else if (token_table[i]->operand[0][0] == '=') {
      int disp = 0; //displacement
      int pc = 0; //Program Counter
      int target_addr = 0; //Target 주소
      token_table[i]->nixbpe |= 2;
      op_tab[i] |= token_table[i]->nixbpe;
      op_tab[i] <<= 12; //12비트 공간 할당
      pc = locctr_index[i + 1];
      char* temp;
      int index = 0;
      temp = calloc(strlen(token_table[i]->operand[0]) - 3, 1);
      for (int j = 3; token_table[i]->operand[0][j] != '₩'; j++) {
       temp[index++] = token_table[i]->operand[0][j];
      }
      temp[index] = '₩0';
```

```
      for (int j = 0; j < literal_count; j++) {
       if (strcmp(temp, literal_table[j].literal) == 0) {
        target_addr = literal_table[j].addr;
        break;
       }
      }
      disp = target_addr - pc;
      disp &= 0xfff;
      op_tab[i] |= disp;
     }
     /*Simple Addressing일 때*/
     else {
      int disp = 0;
      int pc = 0;
      int target_addr = 0;

      token_table[i]->nixbpe |= 2;
      op_tab[i] |= token_table[i]->nixbpe;
      op_tab[i] <<= 12;
      pc = locctr_index[i + 1]; //PC에 다음줄 명령어 주소 넣음
      for (int j = 0; j < symbol_count; j++) {
       if (strcmp(token_table[i]->operand[0], "₩0") != 0
        && strcmp(token_table[i]->operand[0], sym_table[j].symbol)
== 0 && sect_num == sym_table[j].sector) {
        target_addr = sym_table[j].addr;
        break;
       }
      }
      disp = target_addr - pc;
      disp &= 0xfff;
      op_tab[i] |= disp;
     }
    }
    /* 오퍼랜드가 없는 경우*/
    else if (strcmp(token_table[i]->operand[0], "₩0") == 0) {
     // 오퍼랜드가 없는경우 p,e = 0,0 이므로 nixbpe에 추가적인
연산필요 없음.
     op_tab[i] |= token_table[i]->nixbpe;
     op_tab[i] <<= 12;
    }
    /*4형식일 때*/
    else if (token_table[i]->operator[0] == '+') {
     int target_addr = 0;
     token_table[i]->nixbpe |= 1; //p,e비트를 0,1로 지정.
     op_tab[i] |= token_table[i]->nixbpe;
     op_tab[i] <<= 20; //20비트의 공간 생성
     for (int j = 0; j < symbol_count; j++) {
      if (strcmp(token_table[i]->operand[0], "₩0") == 0
       && strcmp(token_table[i]->operand, sym_table[j].symbol) ==
0) {
       target_addr = sym_table[j].addr;
       break;
      }
     }
     target_addr &= 0xfffff;
     op_tab[i] |= target_addr;
```

```
    form_tab[i] = 4;
   }
   break;
  }
}
/*operator가 directives 일 때*/
/*LTORG나 END 일 때*/
else if (strcmp(token_table[i]->operator,"LTORG") == 0 ||
 strcmp(token_table[i]->operator,"END") == 0) {
 for (int j = 0; j < literal_count; j++) {
  if (literal_table[j].sector == sect_num) {
   int count = strlen(literal_table[j].literal);
   if (literal_table[j].kind == 1) { //리터럴이 문자형인 경우
    for (int k = 0; k < count; k++) {
     op_tab[i] |= literal_table[j].literal[k];
     if (k < strlen(literal_table[j].literal) - 1)
      op_tab[i] <<= 8;
    }
    form_tab[i] = count;
   }
   else {
    char* tmp;
    int index = 0;
    tmp = (char*)calloc(count + 1, sizeof(char));
    for (int k = 0; k < count; k++)
     tmp[index++] = literal_table[j].literal[k];
    tmp[index] = '₩0';
    op_tab[i] = strtol(tmp, NULL, 16); //16진수로 바꿔줌
    form_tab[i] = count / 2;
    free(tmp);
   }
  }
 }

}
/*operator가 BYTE일 때*/
else if (strcmp(token_table[i]->operator,"BYTE") == 0) {
 int cnt = 0;
 int index = 0;
 char* tmp = NULL;

 for (int j = 2; token_table[i]->operand[0][j] != '₩'; j++)
  cnt++;
 tmp = (char*)calloc(cnt + 1, sizeof(char));
 for (int j = 2; token_table[i]->operand[0][j] != '₩'; j++)
  tmp[index++] = token_table[i]->operand[0][j];
 tmp[index] = '₩0';
 op_tab[i] = strtol(tmp, NULL, 16);
 form_tab[i] = cnt / 2;
 free(tmp);
}
/*operator가 WORD일 때*/
else if (strcmp(token_table[i]->operator,"WORD") == 0) {
 form_tab[i] = 3;
 if (token_table[i]->operand[0][0] >= '0' &&
token_table[i]->operand[0][0] <= '9') //숫자라면
```

```
    op_tab[i] = atoi(token_table[i]->operand[0]);
   else {
    for (int j = 0; j < token_line; j++) {
     if (strcmp(token_table[i]->operator,"₩0") != 0 &&
strcmp(token_table[i]->operator, "EXTREF") == 0) {
      for (int k = 0; k < 3; k++)
       if (strcmp(token_table[i]->operand[k], "₩0") != 0 &&
strstr(token_table[i]->operand[0], token_table[i]->operand[k]) !=
NULL)
        continue;
     }
    }
   }
  }
  else if (strcmp(token_table[i]->operator,"CSECT") == 0)
   sect_num++;
 }
}


/*
-------------------------------------------------------------
* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는
함수이다.
*        여기서 출력되는 내용은 object code (프로젝트 1번) 이다.
* 매계 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프로그램의 결과를
표준출력으로 보내어
*        화면에 출력해준다.
*
-------------------------------------------------------------
*/
void make_objectcode_output(char* file_name)
{
 int start_line = 0; // 섹터 시작의 라인 번호
 int end_line = 0; // 섹터 끝의 라인 번호
 int start_addr = 0; //섹터 시작 주소
 int end_addr = 0; //섹터 끝에서의 프로그램 카운터
 int sector = 0; //현재 control section 번호
 int is_newline = 1; //새로운 줄인지 판단
 int is_endline = 0; // 전체 프로그램이 완전히 끝났는지에 대한 검사.
 int line_char_cnt = 0; //현재 줄에 몇개의 문자가 쓰였는지 기록.
 int temp_data = 0; //주소 계산시 임시 라인
 int now = 0; //현위치

 FILE* fp = fopen(file_name, "w");
 char* tmp;
 char* buffer = (char*)calloc(70, sizeof(char)); // 데이터를 담음
 tmp = (char*)calloc(7, sizeof(char));
 while (is_endline == 0) {
  while (strcmp(token_table[now]->label, "/0") != 0 &&
token_table[now]->label[0] == '.')
   now++;
  /*한 프로그램(섹션)을 시작할 때 프로그램의 이름, 시작주소 기록*/
  fprintf(fp, "H%s₩t", token_table[now]->label); //프로그램 이름 기록
```

```
  if (strcmp(token_table[now]->operand[0], "₩0") != 0)
   start_addr = atoi(token_table[now]->operand[0]);
  fprintf(fp, "%06X", start_addr); // 프로그램 시작주소
  /*프로그램(섹션) 하나당 몇줄의 코드가 있는지, literal은 몇개나 있는지
확인.*/
  for (int i = now + 1; i < token_line; i++) {
   if (strcmp(token_table[i]->operator,"₩0") == 0)
    continue;
   if (strcmp(token_table[i]->operator,"CSECT") == 0)
    break;
   else if (strcmp(token_table[i]->operator,"END") == 0) {
    end_line = i + 1;
    break;
   }
   end_line = i;
  }

  if (end_line == token_line)
   is_endline = 1;
  temp_data = end_line;
  if (end_line != token_line) {
   while (strcmp(token_table[temp_data]->operator,"EQU") == 0)
    temp_data--;

   end_addr = locctr_index[temp_data + 1];//마지막 줄의 PC 주소값.
  }
  else end_addr = locctr_index[temp_data];

  if (sector != 0) {
   if (sector == 1)
    end_addr = 0x2B;
   else end_addr = 0x1C;
  }
  fprintf(fp, "%06X₩n", end_addr - start_addr);
  for (int i = now; i <= end_line; i++) {
   if (token_table[i] == NULL)
    continue;
   else {
    /*EXTDEF(외부정의) 처리*/
    if (strcmp(token_table[i]->operator,"EXTDEF") == 0) {
     now = i;
     fputc('D', fp);
     line_char_cnt = 1;
     for (int j = 0; j < 3; j++) {
      if (strcmp(token_table[i]->operand[j], "₩0") != 0) {
       int sym_addr = 0;
       /*operand가 심볼 테이블에 정의되어있는지 확인.*/
       for (int k = 0; k < symbol_count; k++) {
        if (strcmp(sym_table[k].symbol, token_table[i]->operand[j])
== 0
          && sym_table[k].sector == sector)
          sym_addr = sym_table[k].addr;
       }
       line_char_cnt += (strlen(token_table[i]->operand[j]) + 6);
       if (line_char_cnt > 72) { //한 줄에 들어갈 수 있는 최대의 길이를
넘으면
```

```
        fputs("₩nD", fp);
        line_char_cnt = strlen(token_table[i]->operand[j]) + 6;
       }
       fprintf(fp, "%s%06X", token_table[i]->operand[j], sym_addr);
      }
     }
     fputc('₩n', fp);
     line_char_cnt = 0;
    }
    /*EXTREF(외부참조) 처리*/
    else if (strcmp(token_table[i]->operator,"EXTREF") == 0) {
     now = i;
     fputc('R', fp);
     line_char_cnt = 1;
     for (int j = 0; j < 3; j++) {
      if (strcmp(token_table[i]->operand[j], "₩0") != 0) {
       line_char_cnt += strlen(token_table[i]->operand[j]) + 1;
       if (line_char_cnt > 72) { //한 줄에 들어갈 수 있는 최대의 길이를
넘으면
        fputs("₩nR", fp);
        line_char_cnt = strlen(token_table[i]->operand[j]) + 1;
       }
       fprintf(fp, "%s", token_table[i]->operand[j]);
      }
     }
     fputc('₩n', fp);
     line_char_cnt = 0;
    }
     }
    }
    now++;
    start_line = now;
    int tmp_end = 0;
    while (now <= end_line) {
     if (is_newline) {
      sprintf(buffer, "T%06X",
(locctr_index[now]>locctr_index[now-1]?locctr_index[now]:locctr_inde
x[now-1]));
      line_char_cnt = 7;
      strcat(buffer, "00"); //글자의 개수 나타내는 비트를 00으로 둔다
      line_char_cnt += 2;
      is_newline = 0;
     }
     switch (form_tab[now]) {
     case 1:
      line_char_cnt += 2;
      /*현재 위치한 줄의 글자 수가 69 넘어가는지 여부 확인*/
      if (line_char_cnt > 69) {
       line_char_cnt -= 2;
       sprintf(tmp, "%02X", (strlen(buffer) - 9) / 2);
       //그 줄의 길이를 buffer에 저장
       buffer[7] = tmp[0];
       buffer[8] = tmp[1];
       fputs(buffer, fp); //buffer의 내용을 파일에 저장
       fputc('₩n', fp);
       line_char_cnt = 0;
```

```
      sprintf(buffer, "T%06X", locctr_index[now]); //새로운 줄의 내용
저장 위해 buffer를 초기화
      line_char_cnt += 7;
      strcat(buffer, "00");
      strcat(buffer, tmp);
      line_char_cnt += 2;
     }
     /*그 control section(=sector)가 끝나는 경우*/
     else if (form_tab[now + 1] <= 0 && token_table[now + 1] !=
NULL
      && strcmp(token_table[now + 1]->operator,"END") != 0) {
      sprintf(tmp, "%02X", op_tab[now++]);
      strcat(buffer, tmp);
      sprintf(tmp, "%02X", (strlen(buffer) - 9) / 2);
      buffer[7] = tmp[0];
      buffer[8] = tmp[1];
      fputs(buffer, fp);
      fputc('₩n', fp);
      line_char_cnt = 0;
      is_newline = 1;
     }
     /*넘어가지 않으면 buffer에 저장*/
     else {
      sprintf(tmp, "%02X", op_tab[now++]);
      strcat(buffer, tmp);
     }
     break;
    case 2:
      line_char_cnt += 4;
      if (line_char_cnt > 69) {
       line_char_cnt -= 4;
       sprintf(tmp, "%02X", (strlen(buffer) - 9) / 2) + 1;
       buffer[7] = tmp[0];
       buffer[8] = tmp[1];
       fputs(buffer, fp);
       fputc('₩n', fp);
       line_char_cnt = 0;
       sprintf(buffer, "T%06X", locctr_index[now]);
       line_char_cnt += 7;
       strcat(buffer, "00");
       line_char_cnt += 2;
       sprintf(tmp, "%04X", op_tab[now++]);
       strcat(buffer, tmp);
       line_char_cnt += 4;
      }
      else if (form_tab[now + 1] <= 0 && token_table[now + 1] !=
NULL
       && strcmp(token_table[now + 1]->operator,"END") != 0) {
       sprintf(tmp, "%02X", op_tab[now++]);
       strcat(buffer, tmp);
       sprintf(tmp, "%02X", (strlen(buffer) - 9) / 2);
       buffer[7] = tmp[0];
       buffer[8] = tmp[1];
       fputs(buffer, fp);
       fputc('₩n', fp);
       line_char_cnt = 0;
```

```
      is_newline = 1;
     }
     else {
      sprintf(tmp, "%04X", op_tab[now++]);
      strcat(buffer, tmp);
     }
     break;
    case 3:
     line_char_cnt += 6;
     if (line_char_cnt > 69) {
      line_char_cnt -= 6;
      sprintf(tmp, "%02X", (strlen(buffer) - 8) / 2);
      buffer[7] = tmp[0];
      buffer[8] = tmp[1];
      fputs(buffer, fp);
      fputc('₩n', fp);
      line_char_cnt = 0;
      sprintf(buffer, "T%06X", locctr_index[now]);
      line_char_cnt += 7;
      strcat(buffer, "00");
      line_char_cnt += 2;
      sprintf(tmp, "%06X", op_tab[now++]);
      strcat(buffer, tmp);
      line_char_cnt += 6;
     }
     else if (form_tab[now + 1] <= 0 && token_table[now + 1] !=
NULL
      && strcmp(token_table[now + 1]->operator,"END") != 0) {
      sprintf(tmp, "%06X", op_tab[now++]);
      strcat(buffer, tmp);
      sprintf(tmp, "%02X", (strlen(buffer) - 9) / 2);
      buffer[7] = tmp[0];
      buffer[8] = tmp[1];
      fputs(buffer, fp);
      fputc('₩n', fp);
      line_char_cnt = 0;
      is_newline = 1;
     }
     else {
      sprintf(tmp, "%06X", op_tab[now++]);
      strcat(buffer, tmp);
     }
     break;
    case 4:
     line_char_cnt += 8;
     if (line_char_cnt > 69) {
      line_char_cnt -= 8;
      sprintf(tmp, "%02X", (strlen(buffer) - 9) / 2);
      buffer[7] = tmp[0];
      buffer[8] = tmp[1];
      fputs(buffer, fp);
      fputc('₩n', fp);
      line_char_cnt = 0;
      sprintf(buffer, "T%06X", locctr_index[now]);
      line_char_cnt += 7;
      strcat(buffer, "00");
```

```
      line_char_cnt += 2;
      sprintf(tmp, "%08X", op_tab[now++]);
      strcat(buffer, tmp);
      line_char_cnt += 8;
     }
     else if (form_tab[now + 1] <= 0 && token_table[now + 1] !=
NULL
      && strcmp(token_table[now + 1]->operator,"END") != 0) {
      sprintf(tmp, "%08X", op_tab[now++]);
      strcat(buffer, tmp);
      sprintf(tmp, "%02X", (strlen(buffer) - 9) / 2);
      buffer[7] = tmp[0];
      buffer[8] = tmp[1];
      fputs(buffer, fp);
      fputc('₩n', fp);
      line_char_cnt = 0;
      is_newline = 1;
     }
     else {
      sprintf(tmp, "%08X", op_tab[now++]);
      strcat(buffer, tmp);
     }
     break;
    default:
     if (token_table[now] != NULL &&
strcmp(token_table[now]->operator,"END") == 0)
      tmp_end = now;
     else if (strcmp(token_table[now - 2]->operator,"₩0") != 0 &&
strcmp(token_table[tmp_end]->operator,"END") == 0
      && token_table[now] != NULL) {
      sprintf(tmp, "%02X", (strlen(buffer) - 9) / 2);
      buffer[7] = tmp[0];
      buffer[8] = tmp[1];
      fputs(buffer, fp);
      fputc('₩n', fp);
     }
     else if (token_table[now] == NULL) {
      sprintf(tmp, "%02X", (strlen(buffer) - 9) / 2);
      buffer[7] = tmp[0];
      buffer[8] = tmp[1];
      fputs(buffer, fp);
      fputc('₩n', fp);
     }
     else {
      is_newline = 1;
      line_char_cnt = 0;
     }
     now++;
    }
   }
   now--;
   for (int i = start_line - 1; i <= end_line; i++) {
    /*외부참조(EXTREF) 처리*/
    if (strcmp(token_table[i]->operator,"EXTREF") == 0) {
     char* index = NULL;
     for (int j = 0; j < 2; j++) {
```

```
      if (strcmp(token_table[i]->operand[j], "₩0") != 0) {
       for (int k = start_line; k <= end_line; k++) {
        if (token_table[k] != NULL) {
         if ((index = strstr(token_table[k]->operand[0],
token_table[i]->operand[j])) != NULL) {
          if (search_opcode(token_table[k]->operator) <= MAX_INST)
           fprintf(fp, "M%06X05+%s₩n", locctr_index[k] + 1,
token_table[i]->operand[j]);
          else {
           if ((index - 1)[0] == '-')
            fprintf(fp, "M%06X06-%s₩n", locctr_index[k] + 1,
token_table[i]->operand[j]);
           else
            fprintf(fp, "M%06X06+%s₩n", locctr_index[k] + 1,
token_table[i]->operand[j]);
          }
         }
        }
       }
      }
     }
     break;
    }
   }
   if (sector == 0)
    fprintf(fp, "E%06X₩n₩n", locctr_index[start_line]);
   else fputs("E₩n₩n", fp);
   sector++;
    now++;
   }
   fclose(fp);
}



/*
-------------------------------------------------------------
* 설명 : strcmp의 연속 구분자 무시 문제를 해결하기 위해 추가적으로
구현한 함수이다.
* 매계 : 구분할 string, 구분자.
* 반환 : 구분된 토큰 배열
* 주의 :
*
-------------------------------------------------------------
*/
char* strsep(char** stringp, const char* delim) {
 char* start = *stringp;
 char* p;

 p = (start != NULL) ? strpbrk(start, delim) : NULL;

 if (p == NULL) {
  *stringp = NULL;
 }
 else {
  *p = '₩0';
  *stringp = p + 1;
```

```
 }

 return start;
}

int search_literal(char* str) {
 for (int i = 0; i < literal_count; i++) {
  if (strcmp(literal_table[i].literal, str) == 0)
   return 0;
 }
 return -1;
}

int search_symbol(char* str, int sector_num) {
 for (int i = 0; i < symbol_count; i++) {
  if (strcmp(sym_table[i].symbol, str) == 0) {
   if (sector_num == sym_table[i].sector)
    return 0;
  }
 }
 return -1;
}
```