

설계(프로젝트) 보고서

나는 송실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.
나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다.
3. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

교과목	시스템프로그래밍 2020
프로젝트 명	C언어로 파서 구현하기
교과목 교수	최 재 영
제출인	전자정보공학부 학번: 20160433 성명: 김민정 (출석번호: 108)
제출일	2020년 4월 29 일

차 례

1장 프로젝트 개요

1.1 개발 배경 및 목적

2장 배경 지식

2.1 주제에 관한 배경지식

2.2 기술적 배경지식

3장 시스템 설계 내용

3.1 전체 시스템 설계 내용

3.2 모듈별 설계 내용

4장 시스템 구현 내용 (구현 화면 포함)

4.1 전체 시스템 구현 내용

4.2 모듈별 구현 내용

4.3 기존에 생성해놓은 inst.data 파일 내용

4.4 디버깅 과정

4.5 수행결과

5장 기대효과 및 결론

첨부 프로그램 소스파일

1장 프로젝트 개요

1.1 개발 배경 및 목적

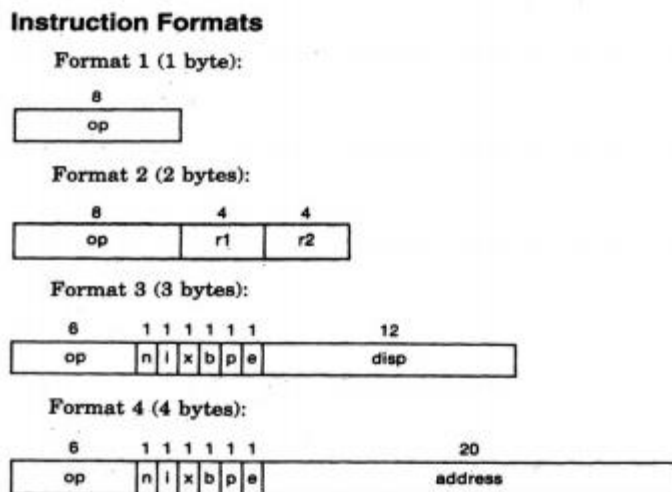
- 주어진 input파일을 이용하여 기본적인 SIC/XE 어셈블러를 구현해보는 프로젝트를 진행해보았다. 이번 프로젝트는 이전에 진행했었던 과제 5에 원하는 기능들을 추가하는 등의 확장을 통하여 input 파일을 원하는 object program으로 작성해보는 프로젝트이다. input파일을 라인별로 읽어들이고 후 각각의 명령어들을 원하는 대로 파싱한 후 파싱한 토큰들을 각각 분석하여 어셈블러가 어떠한 규칙으로 기계어들을 변환하는지를 이해할 수 있다. 또한 object program을 작성하기 위해 input파일의 프로그램을 변환해보면서 그 과정들에 대해 좀 더 명확한 이해가 가능하다. 이를 통해 시스템프로그래밍의 2장 Assemblers를 더 확실하게 공부할 수 있다.

2장 배경 지식

2.1 주제에 관한 배경지식

1) opcode와 명령어 형식

- SIC/XE의 머신은 4종류의 형식을 지원한다.



이 때 비트 e의 값으로 3형식과 4형식을 구분한다. e=0이면 3형식을 따르고 e=1이면 4형식을 따른다.

2) 주소지정방식

- 주소지정방식에는 Absolute addressing, Indirect addressing, Immediate addressing, simple addressing, relative addressing이 존재한다.

3) assembler program

- pass1과 pass2의 방식으로 이루어지는데 pass1에서는 각 라인별로 loc를 지정하고 symbol_table과 literal_table을 작성한다. 그 후 pass2에서는 pass1의 결과물을 이용하여 각 라인별로 object code를 작성하고 이 object code를 이용하여 추후 object program을 만들어낸다.

4) object program format

- object program은 Header record, Define record, Refer record, Text record, Modification record, End record의 형식으로 되어 있는데 각각의 record들의 형식은 다음과 같다.

□ Header record

- ◆ Col. 1 H
- ◆ Col. 2~7 Program name
- ◆ Col. 8~13 Starting address of object program (Hex)
- ◆ Col. 14~19 Length of object program in bytes (Hex)

◆ Define record

- Col. 1 D
- Col. 2-7 Defined external symbol name
- Col. 8-13 Relative address of symbol within this control section
- Col. 14-73 Repeat information in Col. 2-13 for other external symbols

◆ Refer record

- Col. 1 R
- Col. 2-7 Referred external symbol name
- Col. 8-73 Names of other external reference symbols

□ Text record

- ◆ Col. 1 T
- ◆ Col. 2~7 Starting address for object code in this record (Hex)
- ◆ Col. 8~9 Length of object code in this record in bytes (Hex)
- ◆ Col. 10~69 Object code in Hex (2 column per byte)

◆ Modification record (revised)

- Col. 1 M
- Col. 2-7 Starting address of the field to be modified
- Col. 8-9 Length of the field to be modified
- Col. 10 Modification flag (+ or -)
- Col. 11-16 External symbol whose value is to be added to or subtracted from the indicated field

□ End record

- ◆ Col. 1 E
- ◆ Col. 2~7 Address of first executable instruction in object (hex)

2.2 기술적 배경지식

1) 파일입출력

– 파일포인터를 이용하여 파일에 데이터를 읽고 썼다. `fopen()`, `fclose()`를 이용하여 파일을 열고 닫았는데 `fopen()`의 리턴값을 파일포인터로 지정하면 파일을 이용한 연산이 가능하다. 또한 파일포인터를 `stdout`, `stdin`, `stderr`로 지정하면 표준출력, 표준입력, 표준에러로 데이터를 읽고 쓸 수 있다. 데이터의 입출력은 `fprintf()`, `fscanf()`를 이용하였다. 이 함수들은 `format`을 지정할 수가 있어서 어떠한 종류의 데이터이든 편하게 입출력을 할 수 있는 함수들이다. 또한, 첫 번째 파라미터로 파일포인터를 지정해주는데 이때의 값이 어떤 값이냐에 따라 데이터를 파일에서 읽거나 파일에 저장할지, 혹은 표준입력으로 데이터를 입력받거나 표준출력으로 데이터를 출력할지가 결정된다.

2) 토큰분리

– 문자열들을 가공을 하기 위해서는 원하는 문자열만 잘라 보관한 후에 사용하는 경우가 많다. 이번 프로젝트에서도 필수적으로 사용해야 했던 기술이 토큰분리인데 이를 위해 사용한 함수는 `strtok()`, `strtok_s()`이다. 이 두 함수들은 포인터를 이용하여 문자열을 분리하는데 두 함수의 차이는 세 번째 파라미터의 유무이다. `strtok_s()`는 구분자로 토큰분리를 하고 난 후의 문자열을 저장하기 위한 세 번째 파라미터가 존재한다. VS2005 이상에서는 원활한 작동을 위해 `strtok()`보다는 `strtok_s()`의 사용을 권장하고 있다. 이 함수들의 첫 번째 파라미터에는 토큰분리를 하기 위해 탐색하고자 하는 문자열을 넣어주어야 한다. 두 번째 파라미터에는 구분자를 넣어주면 되는데 토큰분리가 끝난 함수의 리턴값인 토큰에는 구분자는 포함되지 않고 구분자를 만나기 전까지의 문자열이 들어가 있다.

또한 구분자는 여러개를 사용할 수도 있는데 만일 ‘,’와 ‘?’를 모두 구분자로 사용하고 싶다면 두 번째 파라미터에 “,?”를 넣어주는 방식으로 이 함수들을 사용할 수 있다. 또한 한 번 토큰분리를 진행한 후에 같은 문자열에서 마저 토큰분리를 진행하고 싶다면 첫 번째 파라미터에 NULL을 넣어주면 된다.

3) 비트연산

- 비트연산자에는 다음과 같은 연산자들이 있다.

&	비트단위 AND 연산
	비트단위 OR 연산
^	비트단위 XOR 연산
~	비트단위 NOT 연산
>>	피연산자의 비트열을 오른쪽으로 이동
<<	피연산자의 비트열을 왼쪽으로 이동

연산자 ‘<<’와 ‘>>’의 경우 시프트연산자라고도 하는데 이 연산자들의 사용방법은 다음과 같다. ‘a << 2’와 같이 사용한다면 이 연산의 의미는 다음과 같다. a의 비트열을 왼쪽으로 2칸 이동하라는 의미이다. 만일 ‘b >> 5’와 같이 사용했다면 이 연산의 의미는 b의 비트열을 오른쪽으로 5칸 이동하라는 의미이다.

4) 서식지정자를 활용한 입출력

- 다음과 같은 서식지정자들을 활용해서 입출력을 좀 더 편하게 할 수 있다.

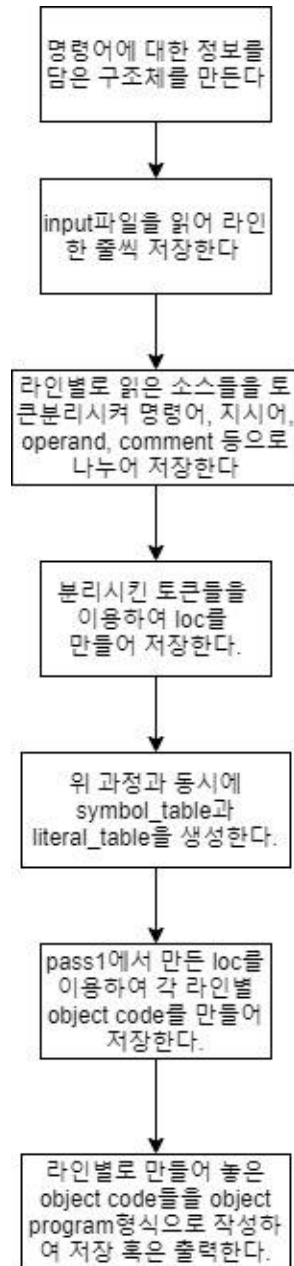
모든 서식지정자들을 넣지는 않고 이번 프로젝트에서 사용한 것들만 추가하였다.

서식 지정자	설명
d,i	부호 있는 10진 정수
u	부호 없는 10진 정수
o	부호 없는 8진 정수
x	부호 없는 16진 정수(소문자)
X	부호 없는 16진 정수(대문자)
f	실수를 소수점으로 표기(소문자)
e	실수 지수표기법 사용(소문자)
a	실수를 16진법으로 표기(소문자)
c	문자
s	문자열
p	포인터의 메모리 주소

플래그	설명
-	왼쪽 정렬
+	양수일 때는 +, 음수일 때는 -
공백	양수일 때는 공백, 음수일 때는 -
#	진법에 맞게 숫자 앞에 0, 0x, 0X 붙임
0	출력하는 폭의 남은 공간에 0으로 채움
폭	설명
숫자	지정된 숫자 만큼 폭을 지정하여 출력, 실수는 .(점), e+ 까지 폭에 포함됨

3장 시스템 설계 내용

3.1 전체 시스템 설계 내용

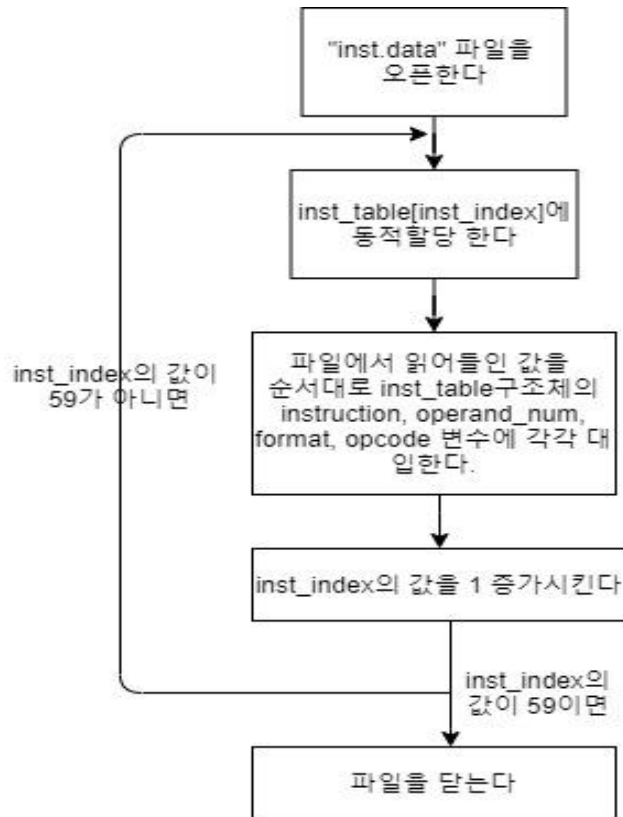


3.2 모듈별 설계 내용

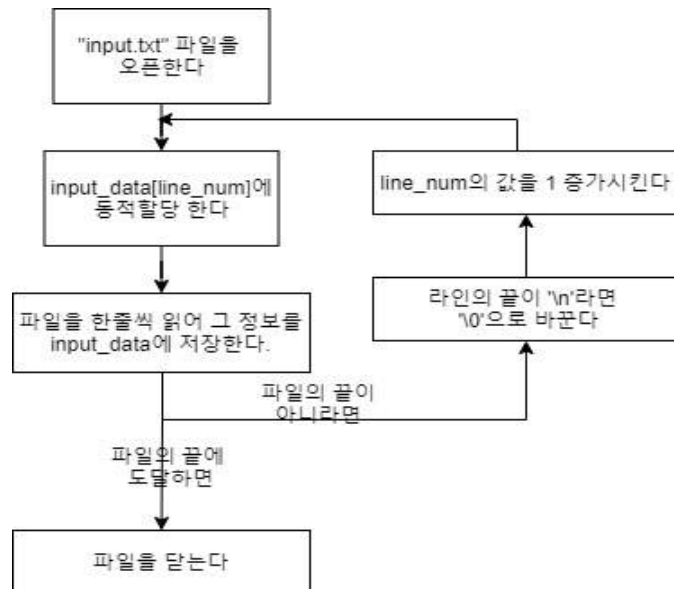
-기본적으로 주어진 함수들을 이용하여 프로그램을 구현하였으나 필요에 의해 여러 함수와 구조체들을 추가하였다.

1) init_my_assembler()

①int init_inst_file(char *inst_file);



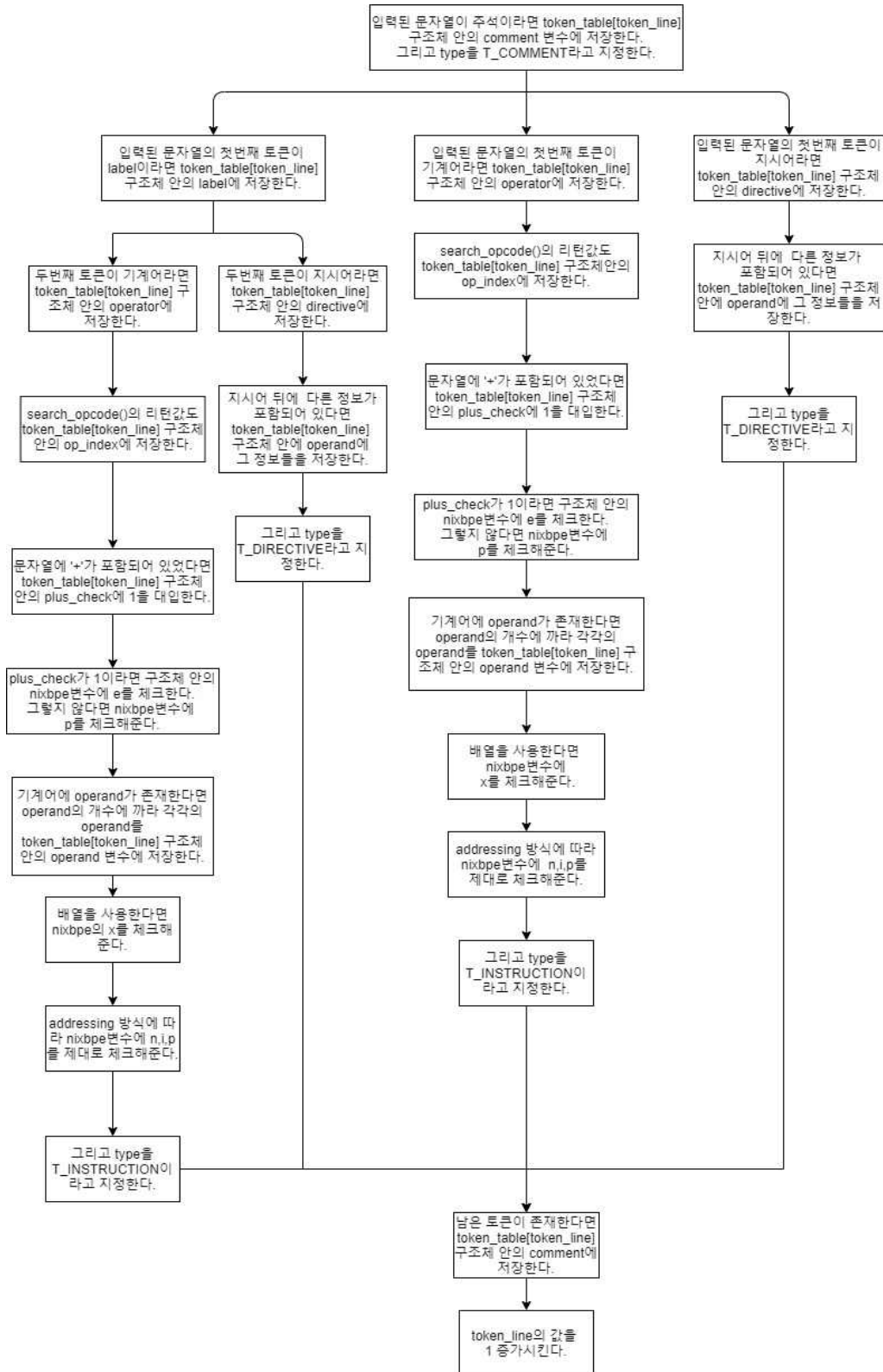
② int init_input_file(char *input_file);



2) assem_pass1();

-input_data 배열을 파라미터로 주어 token_parsing()를 호출하여 입력된 파일들을 한 줄씩 토큰별로 분리한 후 각 라인별로 분리된 토큰에 대한 정보를 활용하여 loc를 할당해주는 함수이다. 또한 loc를 할당하면서 symbol_table과 literal_table을 생성해준다.

① int token_parsing(char *str);



② int search_opcode(char *str);

-inst_table 배열과 입력된 문자열을 비교하여 입력된 문자열이 기계어인지를 검사하는 함수이다.

③ int search_directive(char *str);

-함수안에 저장해놓은 directive_table배열과 입력된 문자열을 비교하여 입력된 문자열이 지시어인지를 검사하는 함수이다.

④ int make_symtab(char*str);

-주어진 문자열이 symbol_table에 존재하지 않는다면 symbol_table에 추가하는 방식으로 symbol_table을 생성해주는 함수이다.

⑤ int search_symtab(char*str);

-symbol_table을 탐색하여 주어진 문자열과 일치하는 symbol의 주소값을 반환하는 함수이다.

⑥ int make_literaltab(char*str);

-주어진 문자열이 literal_table에 존재하지 않는다면 literal_table에 추가하는 방식으로 literal_table을 생성하는 함수로 이 함수에서는 literal의 이름만 넣어준다.

⑦ void addr_literal_tab();

-make_literaltab()에서 넣어준 literal들의 주소값을 넣어주는 함수이다.

⑧ void make_symtab_output(char *file_name);

-인자로 들어오는 값이 NULL이라면 파일포인터를 stdout으로 지정해 주고 인자로 들어오는 값이 파일이름이라면 파일포인터를 쓰기권한으로 오픈해준 파일의 포인터로 지정해준다. 그 후 fprintf를 이용하여 symbol_table의 symbol과 주소값을 출력/저장해주는 함수이다.

⑨ void make_literaltab_output(char *file_name);

-인자로 들어오는 값이 NULL이라면 파일포인터를 stdout으로 지정해 주고 인자로 들어오는 값이 파일이름이라면 파일포인터를 쓰기권한으로 오픈해준 파일의 포인터로 지정해준다. 그 후 fprintf를 이용하여 literal_table의 literal과 주소값을 출력/저장해주는 함수이다.

3) assem_pass2()

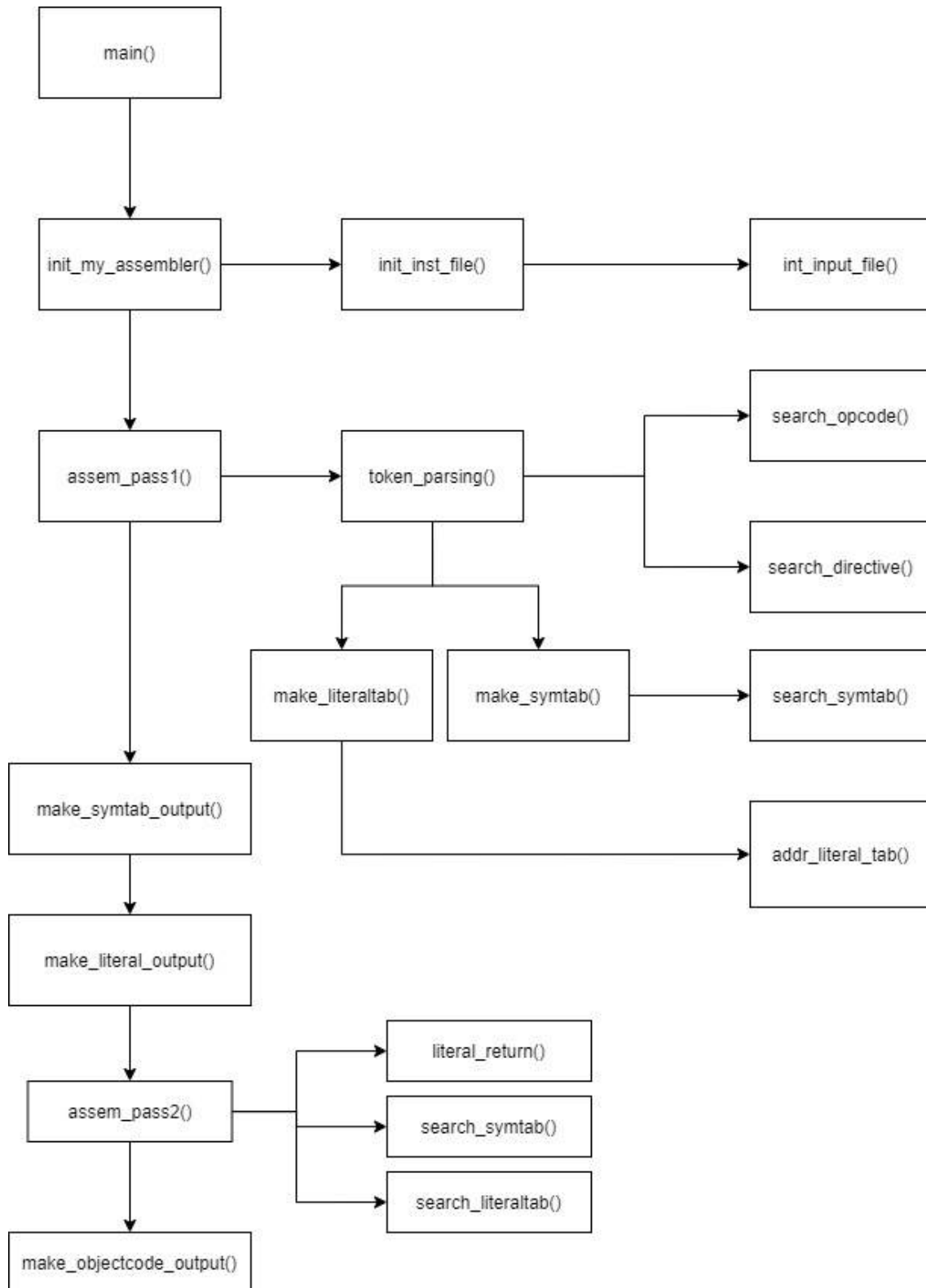
-토큰분리로 분리한 토큰들과 assem_pass1() 수행 후 만들어진 loc를 이용하여 라인별로 object code를 만들어 저장하는 함수이다.

4) make_objectcode_output()

-assem_pass2에서 만든 object code들을 Header record, Define record, Refer record, Text record, Modification record, End record의 형식에 맞추어 작성하여 출력/저장해주는 함수이다.

4장 시스템 구현 내용 (구현 화면 포함)

4.1 전체 시스템 구현 내용



```
int init_my_assembler(void);
```

- 어셈블리로 변환할 준비를 하는 함수로 `init_inst_file()`와 `int_input_file()`를 호출한다.

```
int init_inst_file(char *inst_file);
```

- 기존에 존재하던 명령어파일을 읽어들이어 구조체에 입력하는 함수

```
int init_input_file(char *input_file);
```

- 입력파일을 한줄씩 읽어 저장하는 함수

```
int token_parsing(char *str);
```

- 명령어라인을 토큰별로 분리하는 함수

```
int search_opcode(char *str);
```

- 명령어인지 아닌지를 판별해주는 함수

```
int search_directive(char *str);
```

- 지시어인지 아닌지를 판별해주는 함수

```
static int assem_pass1(void);
```

- 패스1 과정을 수행하는 함수이다. 이 함수를 수행하면 각 라인별로 loc가 할당되고 symbol_table과 literal_table이 생성된다.

```
int make_syntab(char*str);
```

- symbol_table을 생성해주기 위한 함수

```
int search_syntab(char*str);
```

- symbol_table을 탐색하여 해당하는 symbol의 주소값을 반환하는 함수

```
int make_litaltab(char*str);
```

- literal_table을 생성하는 함수. 이 함수에서는 literal의 이름만 넣어준다.

```
void addr_literal_tab();
```

- make_litaltab()에서 넣어준 literal들의 주소를 넣어주는 함수이다.

```
int search_litaltab(char*str);
```

- literal_table을 탐색하여 해당하는 literal의 주소값을 반환하는 함수

```
void literal_return();
```

- literal table에 있는 literal을 메모리에 할당해주기 위해 literal 값을 반환해주는 함수

```
void make_syntab_output(char *file_name);
```

- 만들어져있는 symbol_table을 읽어 symbol과 그 주소값을 저장/출력해주는 함수

```
void make_litaltab_output(char *file_name);
```

- 만들어져있는 literal_table을 읽어 literal과 그 주소값을 저장/출력해주는 함수

```
static int assem_pass2(void);
```

- 패스2 과정을 수행하는 함수이다. 이 함수를 수행하면 각 라인별로 object code가 할당된다.

```
void make_objectcode_output(char *file_name);
```

- 패스2 과정에서 작성된 object code들을 가지고 양식에 맞게 object program을 작성하여 저장/출력해주는 함수

4.2 모듈별 구현 내용

1) instruction 목록 파일로부터 정보를 받아와서 생성하는 구조체 inst_unit는 본인이 만든 파일의 형식(3.3참고)을 참조하여 다음과 같은 변수들을 생성해주었다.

```
struct inst_unit
{
    char instruction[10]; //명령어
    int opcode; //각 명령어에 해당하는 opcode
    int format; //명령어의 형식
    int operand_num; //각 명령의 피연산자 개수
```

```
};
```

2) 어셈블리 할 소스코드를 토큰단위로 관리하기 위한 구조체 token_unit에 다음과 같이 추가로 몇 개의 변수를 추가해주었다.

```
struct token_unit
{
    char *label;           //명령어 라인 중 label
    char *operator;        //명령어 라인 중 operator
    char *operand[MAX_OPERAND]; //명령어 라인 중 operand
    char *comment;         //명령어 라인 중 comment
    char nixbpe;           // 하위 6bit 사용: _ _ n i x b p e
    int op_index;          //새로 추가한 변수로 소스코드에 존재하는 명령어
    //에 대한 정보를 포함하고 있는 inst_table의 index
    int plus_check;        //새로 추가한 변수로 operand에 +가 포함되어 있
    //는지에 대한 정보를 저장
    char *directive;       //새로 추가한 변수로 명령어 라인 중 지시어
    int address;           //새로 추가한 변수로 loc 저장
    char section[6];       //새로 추가한 변수로 해당 라인이 어떤 프로그램에 속해있
    //는지 저장
    TYPE type;            //새로 추가한 변수로 타입을 명시
    int opnum;            //object code에 들어갈 상위 2개의 16진수 값을 저장
};
```

3) symbol을 따로 관리하기 위한 symbol 구조체로 관리를 용이하게 하기 위해 변수를 추가하였다.

```
struct symbol_unit
{
    char symbol[6];
    int addr;
    char type; //새로 추가한 변수로 symbol이 'R'(relative expression)인지 'A'(absolute
    //expression)인지 구분
    char* area; //새로 추가한 변수로 해당 심볼이 어떤 프로그램에 포함되어 있는지 저장
};
```

4) literal을 따로 관리하기 위한 literal 구조체로 관리를 용이하게 하기 위해 변수를 추가하였다.

```
struct literal_unit
{
    char literal[6];
    int addr;
    char type; //새로 추가한 변수로 literal이 'C'로 시작하는지 'X'로 시작하는지 구분
    int alloc; //새로 추가한 변수로 LORG를 만나서 메모리가 할당되었는지의 여부 체크
};
```

5) 프로그램별로 object code를 저장하기 위해 만든 구조체이다.

```
struct CtlSection { //프로그램별로 opcode를 저장하기 위해 추가한 구조체
    char obj_code[MAX_LINES][9]; //해당 라인이 변환되는 address 저장
    int obj_line; //라인의 개수
    char section[10]; //어떤 프로그램을 관리하는지 확인하기 위해 추가한 변수
    int extdef; //지시어 EXTDEF가 나왔을 때 symbol의 개수를 저장
    int extref; //지시어 EXTREF가 나왔을 때 symbol의 개수를 저장
    symbol def[MAX_LINES]; //지시어 EXTDEF가 나왔을 때 symbol을 저장
    symbol ref[MAX_LINES]; //지시어 EXTREF가 나왔을 때 symbol을 저장
    symbol mod[MAX_LINES]; //라인별로 읽으면서 operand에 ref인 변수가 나타났을 때 저장
    int mod_num; //라인별로 읽으면서 operand에 ref인 변수가 나타나면 그 개수를 저장
    int ref_size[MAX_LINES]; //라인별로 읽으면서 operand에 ref인 변수가 나타나면 그 크기를
    저장
    int lastAddr; //프로그램의 마지막 주소값을 저장
    int startAddr; //프로그램의 시작 주소값을 저장
};
```

6) init_my_assembler()

① init_inst_file() 수행 결과 생성되는 instruction 정보를 담은 배열

The screenshot shows a code editor with the following C code for `init_inst_file()`:

```
96 FILE *file;
97 int errno;
98 /* add your code here */
99
100 file = fopen(inst_file, "r"); //읽기모드로 파일을 연다
101
102 if (file == NULL) { //파일 열기에 실패하면 에러
103     errno = -1;
104 }
105 else {
106     while (1) { //동작할당 후 기계어 목록파일의 형식에 따라 해당
107         inst_table[inst_index] = (inst*)malloc(sizeof(inst));
108         memset(inst_table[inst_index], 0, sizeof(inst));
109
110         fscanf(file, "%s %d %d %x", &inst_table[inst_index]-
111             inst_index++;
112
113         if (inst_index == 59)
114             break;
115     }
116     fclose(file);
117
118     for (int i = 0; i < inst_index; i++) {
119         printf("%d\n", i);
120         printf("instruction : %s\n", inst_table[i]->instruction);
121         printf("operand_num : %d\n", inst_table[i]->operand_num);
122         printf("format : %d\n", inst_table[i]->format);
123         printf("opcode : %02X\n\n", inst_table[i]->opcode);
124     }
125     return errno;

```

On the right, a window titled "D:\MinJeong\W... -" displays the generated instruction table data:

```
0 )
instruction : ADD
operand_num : 1
format : 3
opcode : 18

1 )
instruction : ADDF
operand_num : 1
format : 3
opcode : 58

2 )
instruction : ADDR
operand_num : 2
format : 2
opcode : 90

3 )
instruction : AND
operand_num : 1
format : 3
opcode : 40

4 )
instruction : CLEAR
operand_num : 1
format : 2
opcode : B4

```

그림 10 init_inst_file() 수행 결과 생성되는 inst_table배열 중간출력

② init_input_file() 수행 결과 생성되는 line 정보를 담은 배열

my_assembler_20160433.h
my_assembler_20160433.c
project1
(전역 범위)

```

769  *      화면에 출력해준다.
770  *
771  *
772  */
773  void make_symtab_output(char *file_name)
774  {
775      /* add your code here */
776      FILE* fp;
777      if (file_name != NULL) { //인자로 NULL값이 들어오지
778          fp = fopen(file_name, "w+t");
779      }
780      else { //인자로 NULL값이 들어오는 경우 표준출력으로
781          fp = stdout;
782      }
783
784      char *tmp_area = (char*)malloc(sizeof(char));
785      strcpy(tmp_area, sym_table[0].area);
786      //SYMBOL별 주소값이 저장된 TABLE 내용을 출력
787      for (int i = 0; i < sym_num; i++) {
788          if (strcmp(tmp_area, sym_table[i].area)) {
789              fprintf(fp, "%n");
790              strcpy(tmp_area, sym_table[i].area);
791          }
792          fprintf(fp, "%s\t\t\t\t\t%X\n", sym_table[i].symbol,
793              );
794      }
795      if (file_name != NULL) { //파일을 열은 경우 닫아줌.
796          fclose(fp);
797      }

```

D:\MinJeong\실험실대\4-1\시스템 프로그래밍\프로젝트

COPY	0
FIRST	0
CLOOP	3
ENDFIL	17
RETADR	2A
LENGTH	2D
BUFFER	33
BUFEND	1033
MAXLEN	1000
RDREC	0
RLOOP	9
EXIT	20
INPUT	27
MAXLEN	28
WRREC	0
WLOOP	6

그림 14 make_symtab_output()의 인자가 NULL인 경우 화면에 출력

symtab_20160433.txt - 메모장

파일(F)	편집(E)	서식(O)	보기(V)	도움말(H)
COPY			0	
FIRST			0	
CLOOP			3	
ENDFIL			17	
RETADR			2A	
LENGTH			2D	
BUFFER			33	
BUFEND			1033	
MAXLEN			1000	
RDREC			0	
RLOOP			9	
EXIT			20	
INPUT			27	
MAXLEN			28	
WRREC			0	
WLOOP			6	

그림 15 make_symtab_output()의 인자가 symtab_20160433.txt인 경우 파일로 저장된 결과

④ assem_pass1() 수행결과 생성되는 literal_table

- make_literaltab_output()를 수행하면 literal_table이 출력된다. 이때 make_literaltab_output()의 인자가 어떤 값이냐에 따라 output이 화면에 출력되거나 파일로 저장된다.

[illegible]

그림 16 make_literal_tab_output()의 인자가 NULL인 경우 화면에 출력



그림 17 make_literaltab_output()의 인자가 literaltab_20160433.txt인 경우 파일로 저장

8) `assem_pass2()`

-assem_pass2() 수행결과 생성되는 object code 중간출력

```

1022 token_table[i]->operand[0]++;
1023 sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%s", token_table[i]->operand[0]);
1024 printf("%s\n", control_section[cs_num].obj_code[control_section[cs_num].obj_line]);
1025 control_section[cs_num].obj_line++;
1026 continue;
1027 }
1028 if (strstr(token_table[i]->operand[0], "@") != NULL) {
1029     token_table[i]->operand[0]++;
1030 }
1031 if (strstr(token_table[i]->operand[0], "=") != NULL) {
1032     if ((target = search_literals(token_table[i]->operand[0])) != NULL) {
1033         fprintf(stderr, "no literal");
1034     }
1035 }
1036 else {
1037     target = search_syms(token_table[i]->operand[0]);
1038 }
1039 pc = token_table[i + 1]->address;
1040 addr = target - pc;
1041 sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%s", token_table[i]->operand[0]);
1042 printf("%s\n", control_section[cs_num].obj_code[control_section[cs_num].obj_line]);
1043 }
1044 }
1045 control_section[cs_num].obj_line++;
1046 }
1047 }
1048 return 0;
1049 }

```

Generated Object Code (Hex):

```

172027
4B100000
032023
290000
932007
4B100000
3F2FEC
032016
0F2016
010003
0F200A
4B100000
3E2000
454F46
B410
B400
B440
77201F
E3201B
332FFA
0B2015
A004
332009
57900000
B850
3B2FE9
13100000
4F0000
F1

```

그림 18 assem_pass2() 수행결과 생성되는 object code 중간출력

9) make_objectcode_output()

-make_opcode_output()의 인자가 어떤 값이냐에 따라 output이 화면에 출력되거나 파일로 저장된다(4.5 참고).

4.3 기존에 생성해놓은 inst.data 파일 내용

각 명령어들마다 명령어이름, operand의 개수, format, opcode의 순서로 appendix를 참고하여 inst.data파일을 생성하였다. 파일의 내용은 다음과 같다.

inst.data	input.txt	my_assembler_20160433.c
1	ADD 1 3	18
2	ADDF 1 3	58
3	ADDR 2 2	90
4	AND 1 3	40
5	CLEAR 1 2	B4
6	COMP 1 3	28
7	COMPF 1 3	88
8	COMPR 2 2	A0
9	DIV 1 3	24
10	DIVF 1 3	64
11	DIVR 2 2	9C
12	FIX 0 1	C4
13	FLOAT 0 1	C0
14	HIO 0 1	F4
15	J 1 3	3C
16	JEQ 1 3	30
17	JGT 1 3	34
18	JLT 1 3	38
19	JSUB 1 3	48
20	LDA 1 3	00
21	LDB 1 3	68
22	LDCH 1 3	50
23	LDF 1 3	70
24	LDL 1 3	08
25	LDS 1 3	6C
26	LDT 1 3	74
27	LDX 1 3	04
28	LPS 1 3	D0
29	MUL 1 3	20
30	MULF 1 3	60
31	MULR 2 2	98
32	NORM 0 1	C8
33	OR 1 3	44

그림 19 inst.data의 내용(1)

inst.data	input.txt	my_assembler_20160433.c
33	OR 1 3	44
34	RD 1 3	D8
35	RMO 2 2	AC
36	RSUB 0 3	4C
37	SHIFTL 2 2	A4
38	SHIFTR 2 2	A8
39	SIO 0 1	F0
40	SSK 1 3	EC
41	STA 1 3	0C
42	STB 1 3	78
43	STCH 1 3	54
44	STF 1 3	80
45	STI 1 3	D4
46	STL 1 3	14
47	STS 1 3	7C
48	STSW 1 3	E8
49	STT 1 3	84
50	STX 1 3	10
51	SUB 1 3	1C
52	SUBF 1 3	5C
53	SUBR 2 2	94
54	SVC 1 2	B0
55	TD 1 3	E0
56	TIO 0 1	F8
57	TIX 1 3	2C
58	TIXR 1 2	B8
59	WD 1 3	DC
60		

그림 20 inst.data의 내용(2)

4.4 디버깅 과정

-프로그램을 구현하면서 디버깅했던 내용들 중 일부화면들을 첨부하였다.

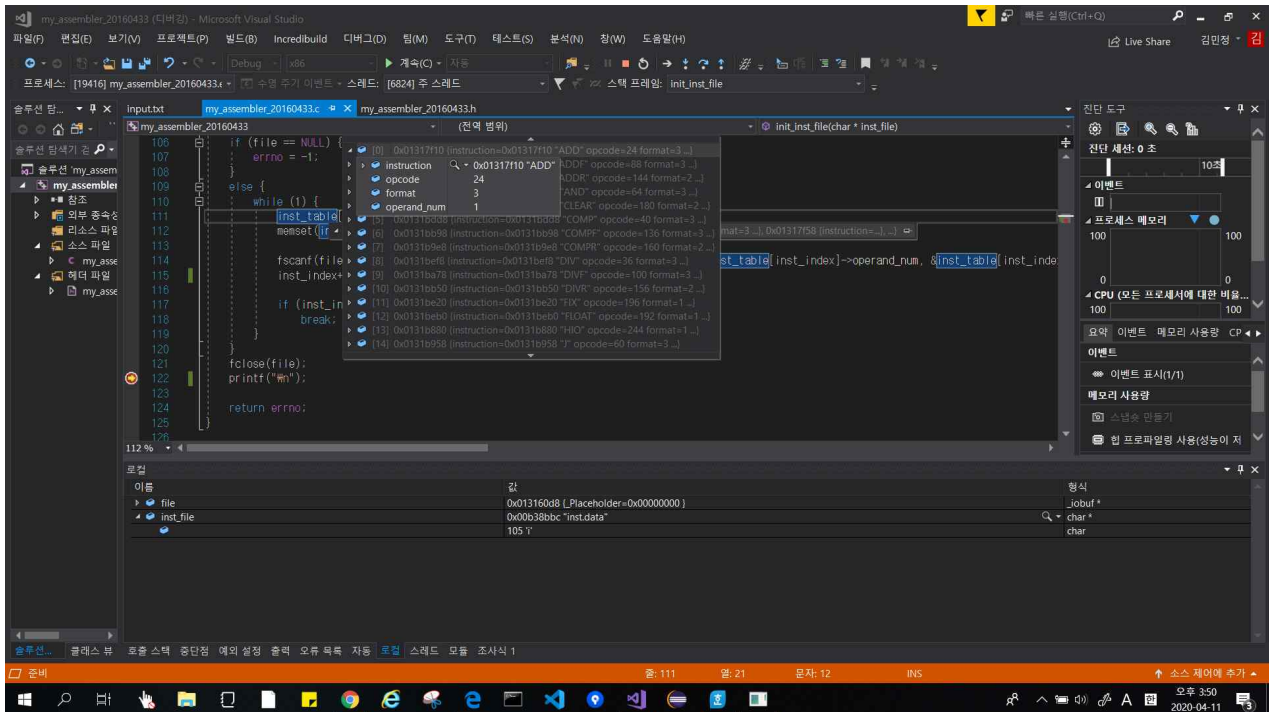


그림 21 기존에 만들어 놓은 inst.data 파일의 내용을 읽어와 inst_table에 저장하는 과정 디버깅

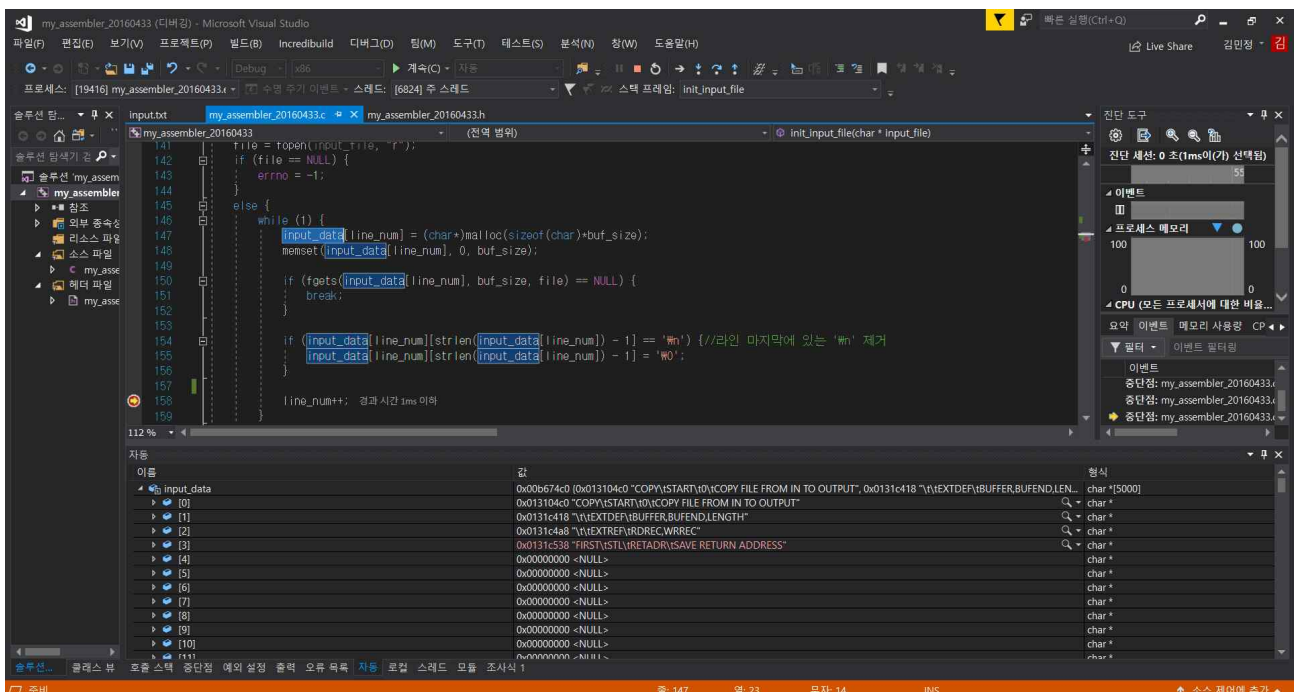


그림 22 input file의 내용을 라인단위로 저장하는 과정 디버깅

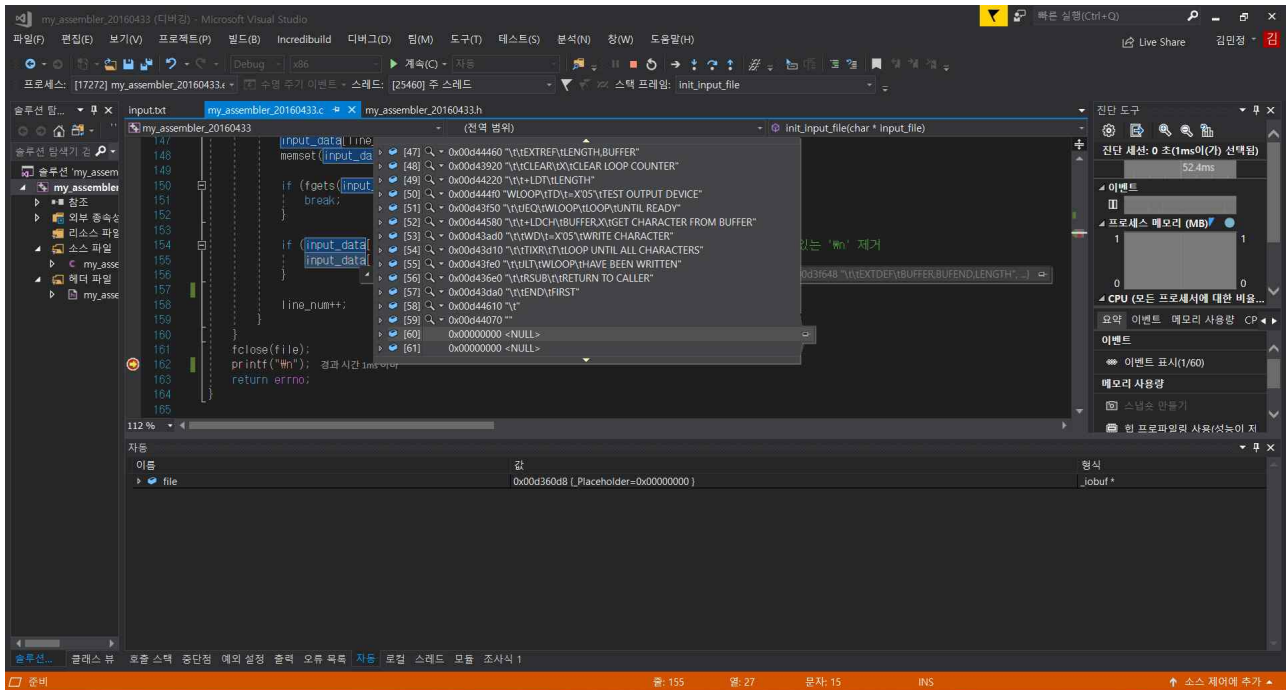


그림 23 input file의 내용을 라인단위로 저장하는 과정 디버깅

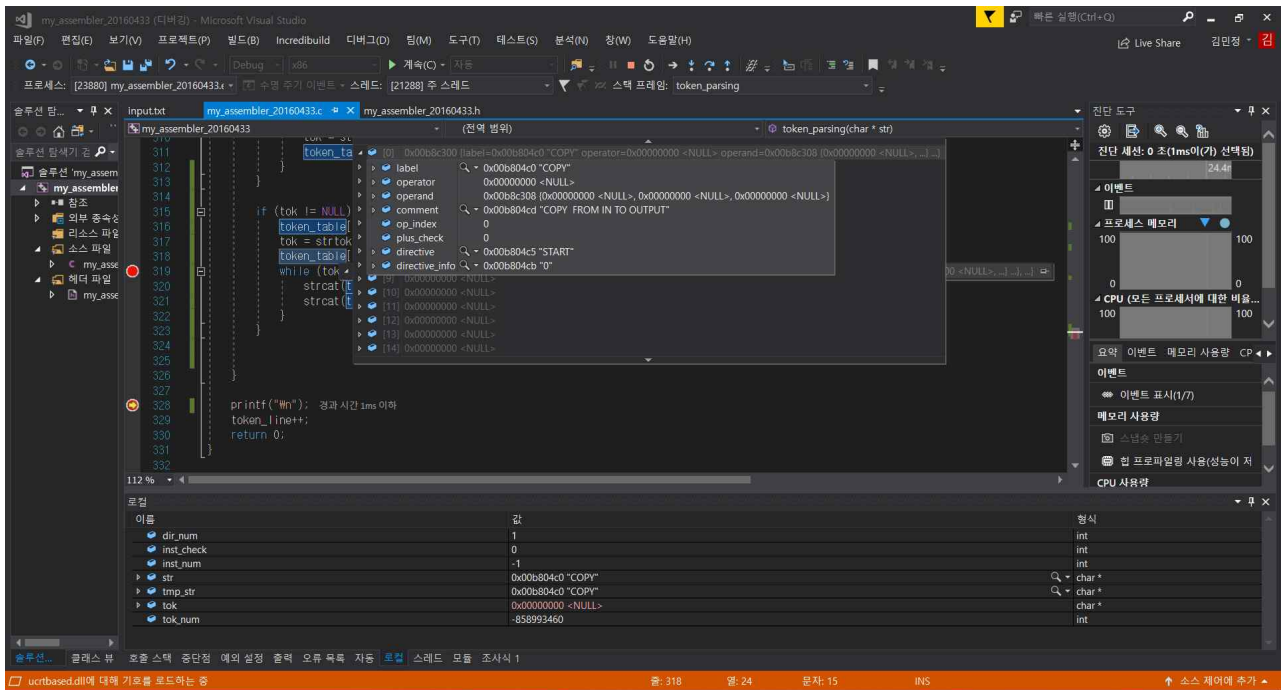


그림 24 소스 코드를 읽어와 토큰단위로 분리하여 토큰 테이블을 작성하는 과정 디버깅

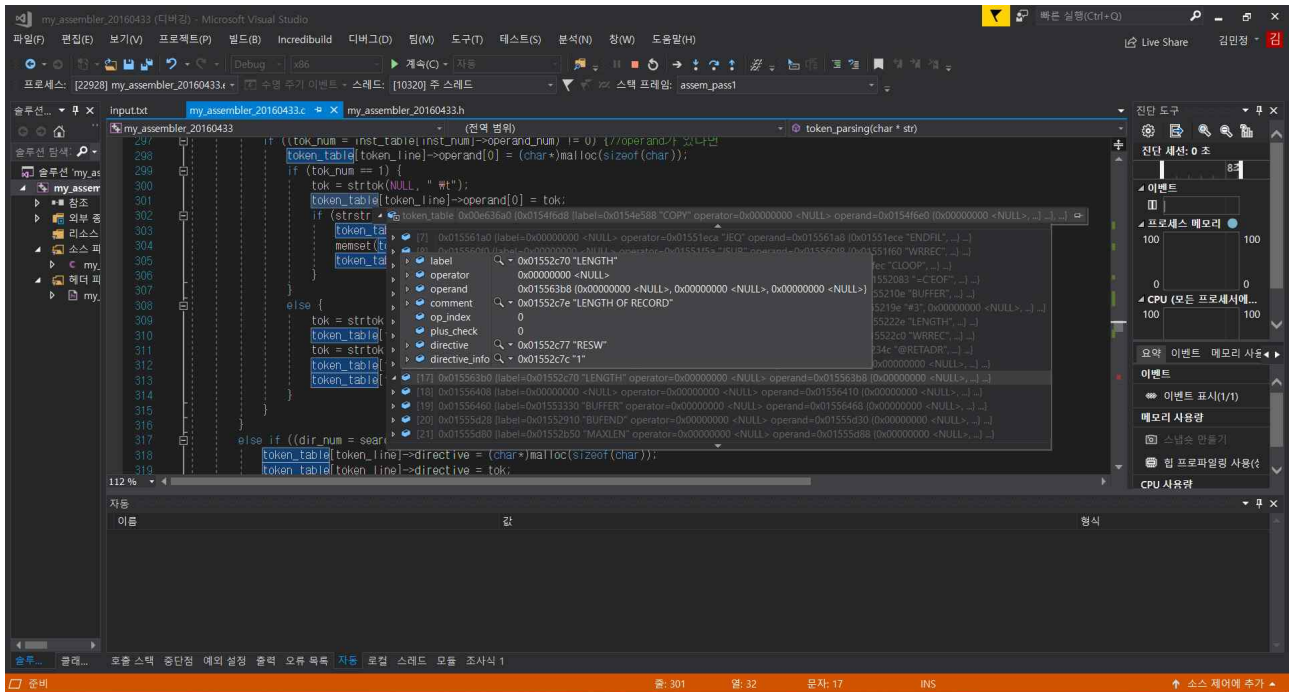


그림 25 소스 코드를 읽어와 토큰단위로 분리하여 토큰 테이블을 작성하는 과정 디버깅

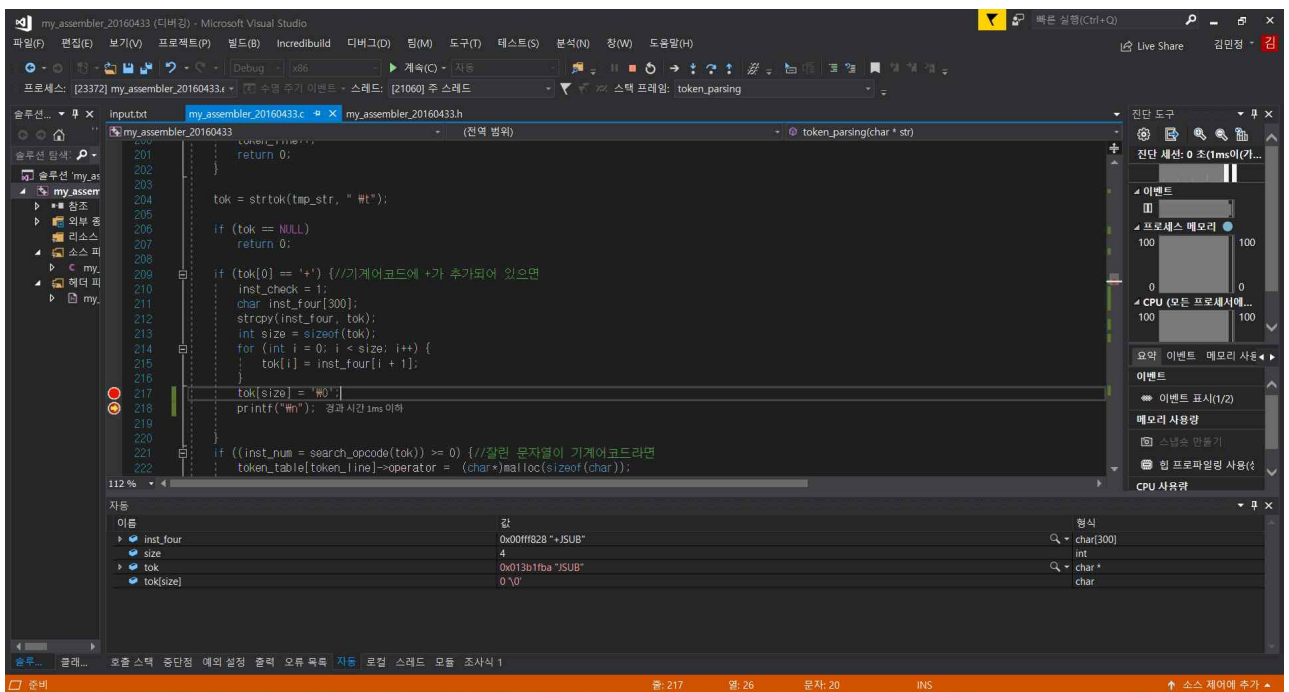


그림 26 소스 코드를 읽어와 토큰단위로 분리하여 토큰 테이블을 작성하는 과정 디버깅

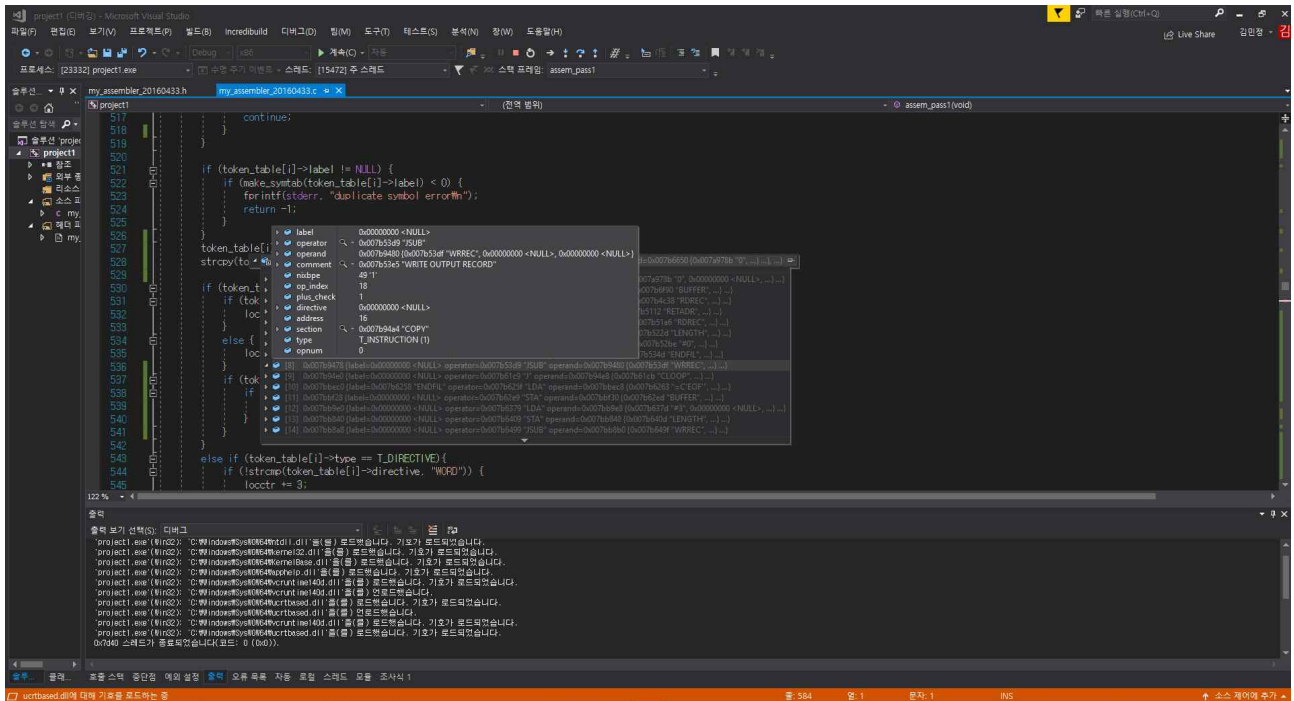


그림 27 loc가 제대로 저장되었는지 디버깅

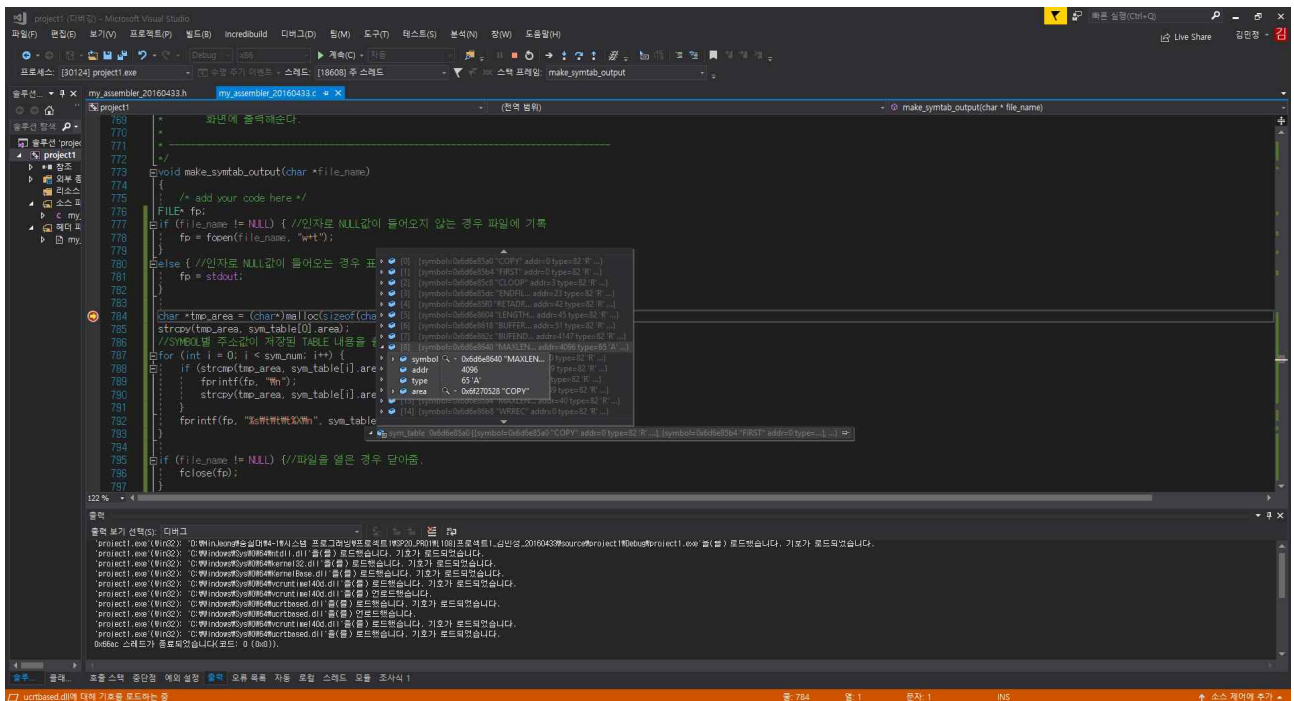


그림 28 symbol_table이 제대로 작성되었는지 디버깅

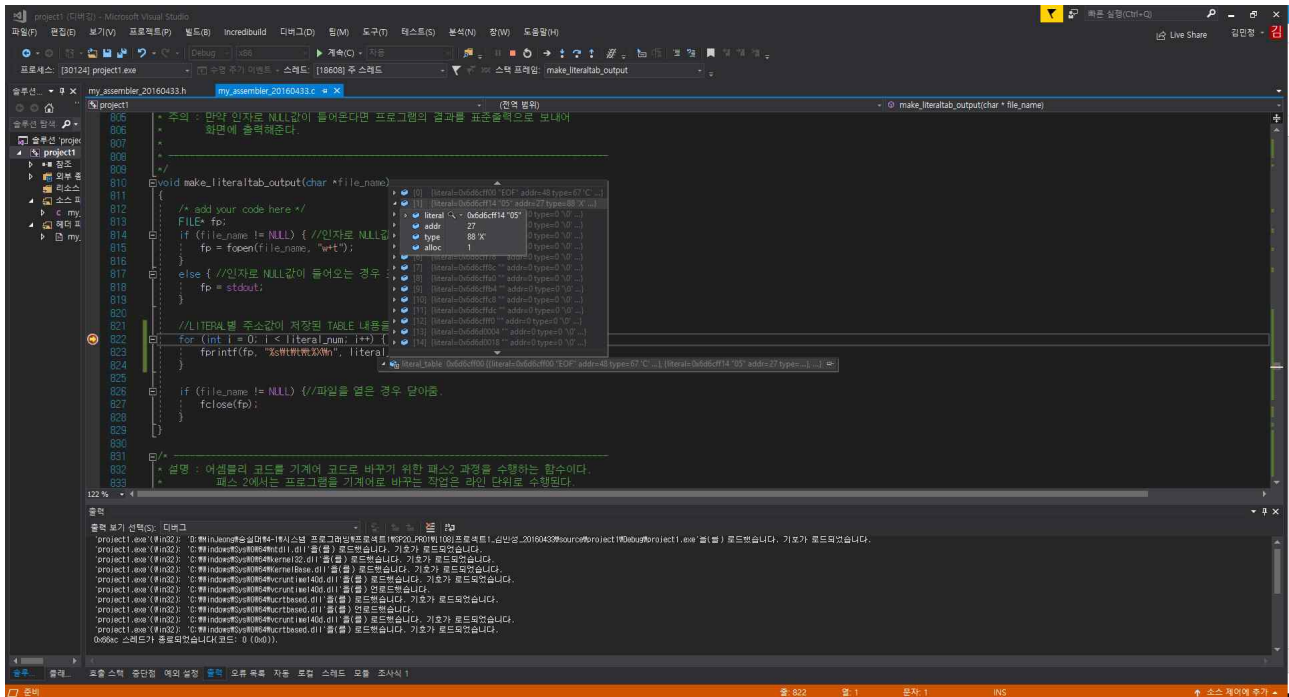


그림 29 literal_table이 제대로 작성되었는지 디버깅

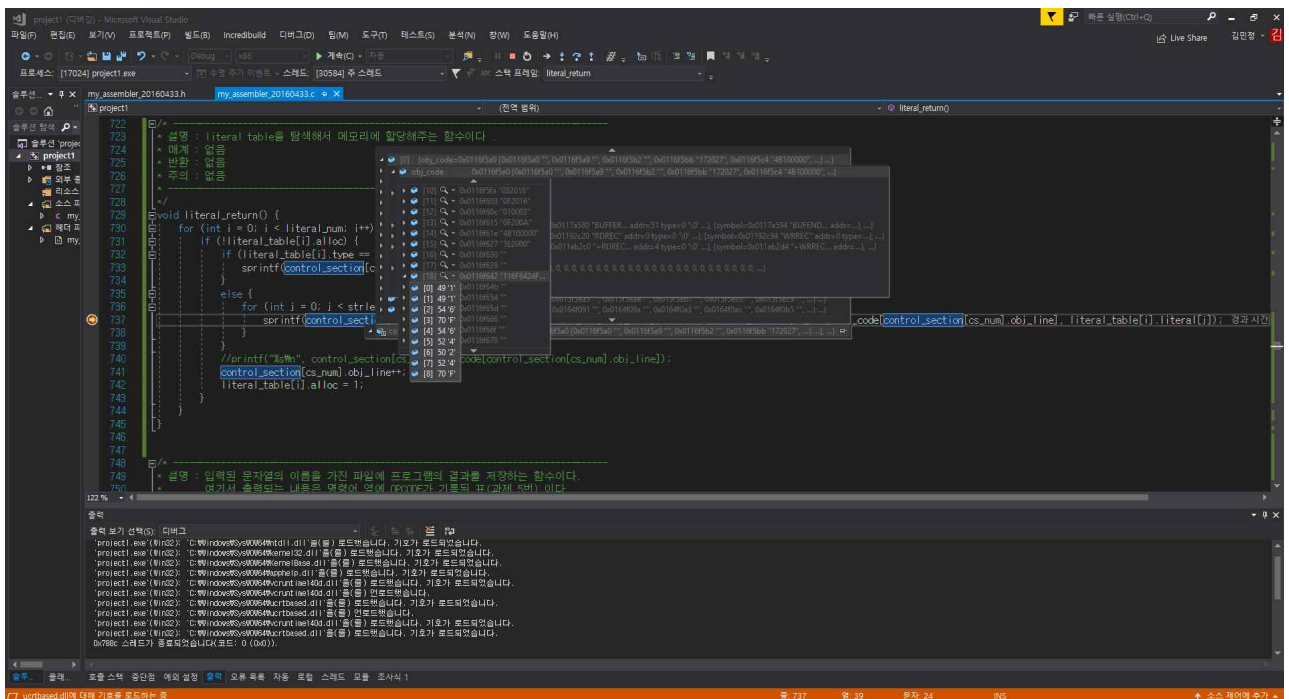


그림 30 object code 만드는 과정 디버깅

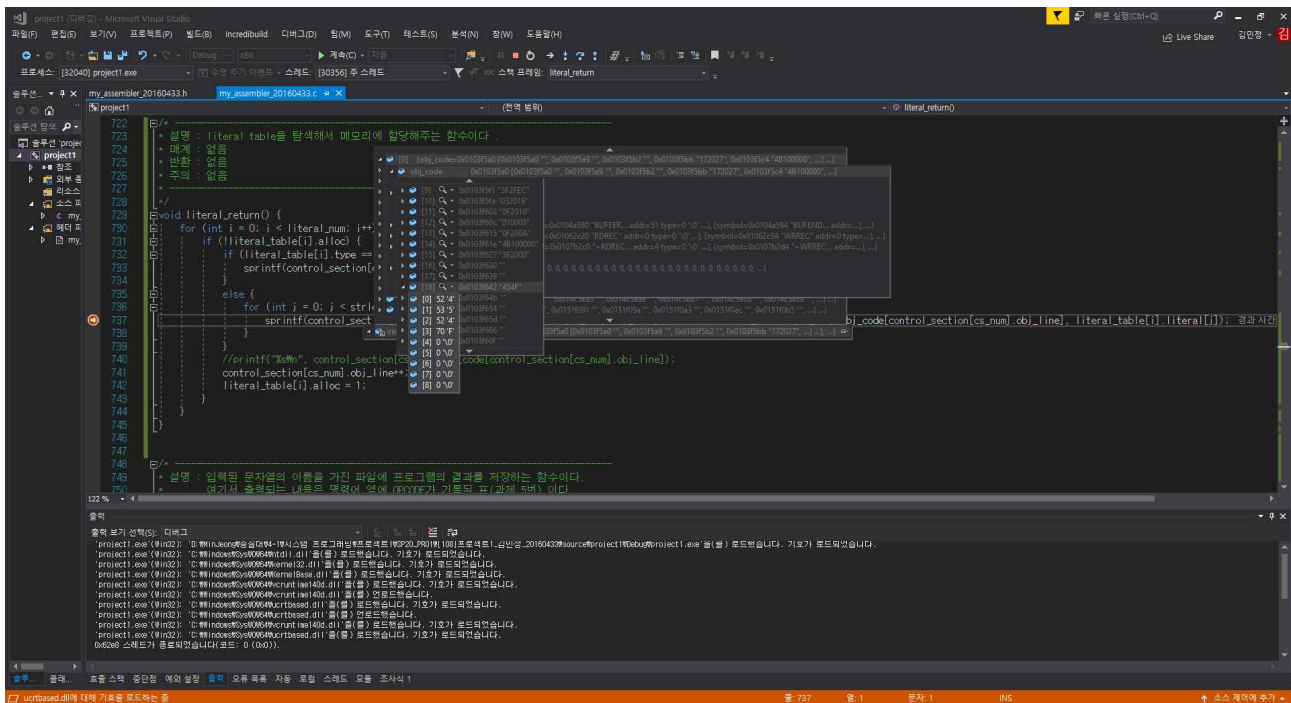


그림 31 object code 만드는 과정 디버깅

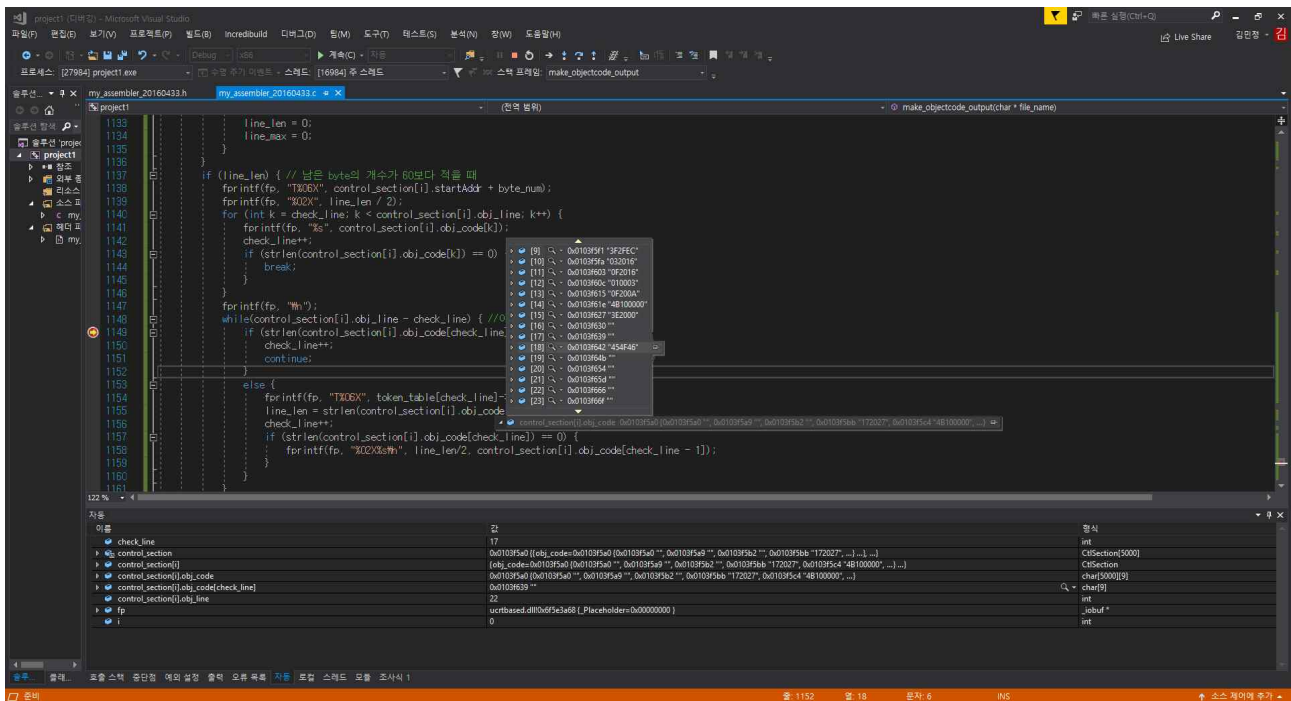
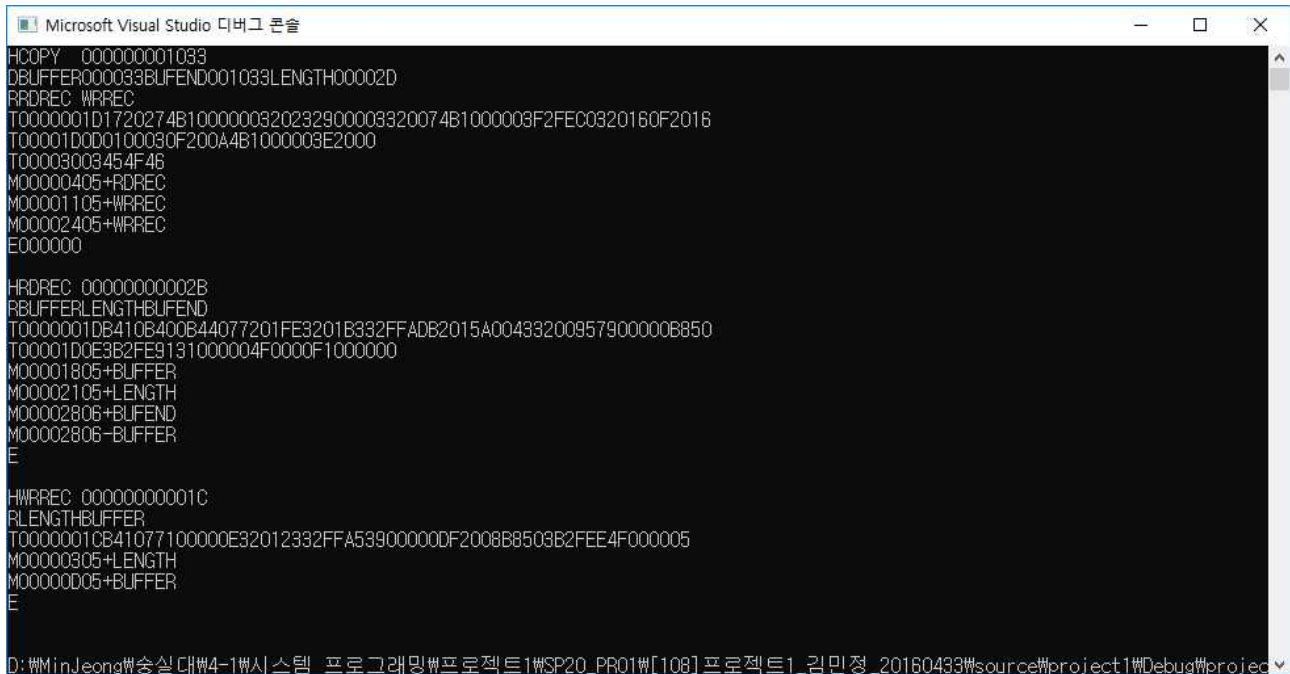


그림 32 object program 작성하는 과정 디버깅

4.5 수행 결과

1) make_objectcode_output()의 인자가 NULL인 경우



```
Microsoft Visual Studio 디버깅 콘솔
HCPY 000000001033
DBUFFER0000033BUFEND001033LENGTH00002D
RRREC WRREC
T0000001D1720274B1000000320232900003320074B1000003F2FEC0320160F2016
T00001D0D0100030F200A4B1000003E2000
T00003003454F46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E0000000

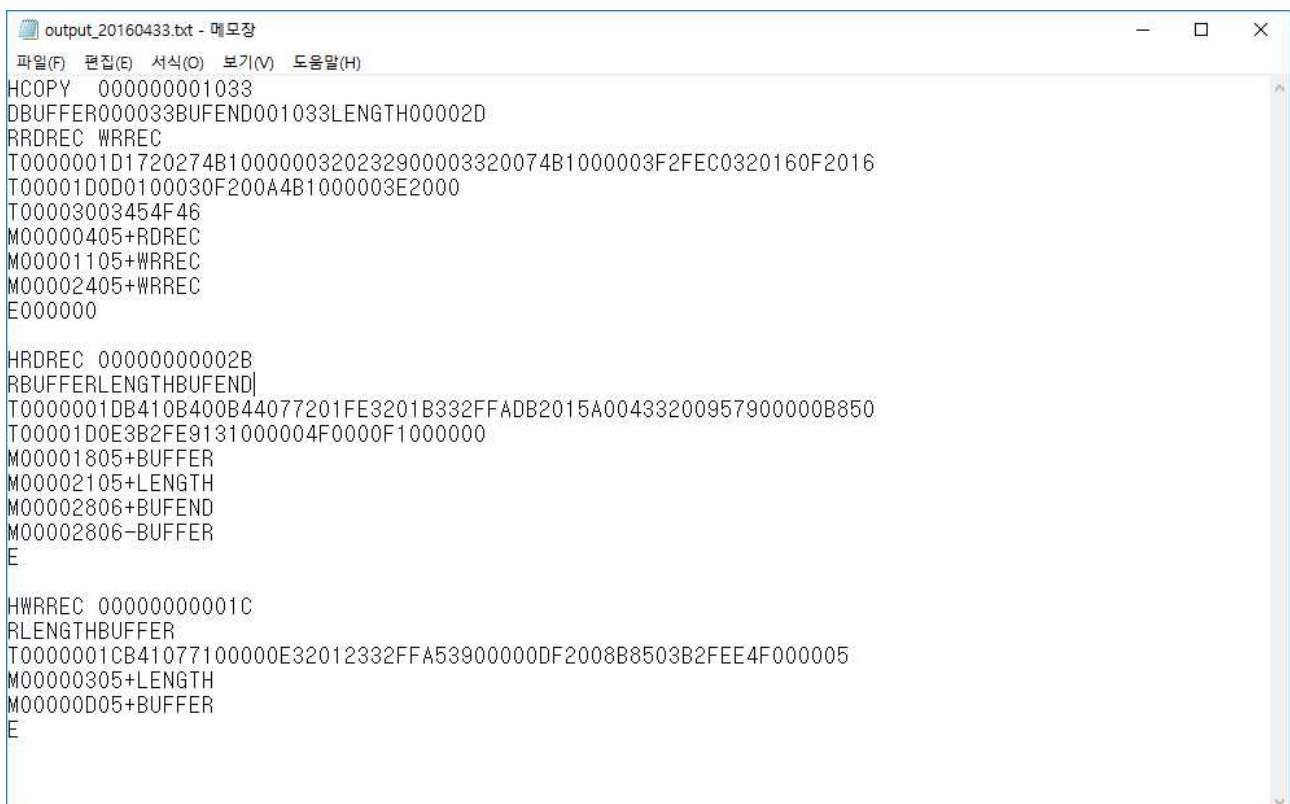
HRDREC 00000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A00433200957900000B850
T00001D0E3B2FE9131000004F0000F1000000
M00001805+BUFFER
M00002105+LENGTH
M00002806+BUFEND
M00002806-BUFFER
E

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

D:\MinJeong\송실대\4-1\시스템 프로그래밍\프로젝트1\SP20_PR01\108]프로젝트1_김민정_20160433\source\project1\Debug\projec
```

그림 33 make_objectcode_output()의 인자가 NULL인 경우 화면에 결과 출력

2) make_objectcode_output()의 인자가 “output_20160433.txt”인 경우



```
output_20160433.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
HCPY 000000001033
DBUFFER0000033BUFEND001033LENGTH00002D
RRREC WRREC
T0000001D1720274B1000000320232900003320074B1000003F2FEC0320160F2016
T00001D0D0100030F200A4B1000003E2000
T00003003454F46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E0000000

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

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

그림 34 make_objectcode_output()의 인자가 “output_20160433.txt”인 경우 파일로 저장

5장 기대효과 및 결론

-이번 프로젝트는 주어진 input파일을 파싱하여 토큰화하고 기본적인 SIC/XE 머신의 어셈블러를 구현해보는 프로젝트였다. 이 프로젝트를 진행하기 전에 과제 5에서 input파일을 토큰화했던 프로그램이 큰 도움이 되었었다. 그 과제를 했었기 때문에 이번 프로젝트를 좀 더 수월하게 할 수 있었다고 생각한다. 이번 프로젝트에서는 input파일을 토큰화하는 것도 굉장히 중요했지만 토큰분리한 토큰들을 어떻게 활용할 것인지가 더 중요한 이슈였다. 이번 프로젝트에서는 COPY프로그램을 변환하는 것이 목표였기 때문에 모든 문법들이나 모든 기능들이 포함되어 있는 어셈블러를 구현하지는 못하였다. 그것이 좀 아쉽기는 하지만 구현하면서 느꼈던 미구현 부분들을 추후에 보완하면 더 나은 어셈블러를 구현할 수 있을 것이다. 또한 이 부분에 대해 더 공부하고 부족한 부분을 채우면서 assembler문법에 대해 더 이해할 수 있는 시간을 갖을 수 있을 것이라고 생각한다. 또한 object program은 수업시간에 배웠던 pass1, pass2를 이해하지 못했다면 만들 수 없었다. 즉, 이번 프로젝트를 구현했다는 것은 수업내용에 대한 체득이 어느정도 되었다고 판단할 수 있는 기준이 될 수 있는 것이다. 물론 구현하면서 아직 잘 모르는 부분이 있다는 것도 알게 되었지만 이 부분을 다시 공부하면 좀 더 완벽한 이해를 할 수 있으리라 믿는다.

첨부 프로그램 소스파일

1)my_assembler_20160433.h

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

#define N (1<<5)
#define I (1<<4)
#define X (1<<3)
#define B (1<<2)
#define P (1<<1)
#define E 1

/*
 * 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
{
    /* add your code here */
    char instruction[10]; //명령어
    int opcode; //각 명령어에 해당하는 opcode
    int format; //명령어의 형식
    int operand_num; //각 명령의 피연산자 개수
};

// 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;

/*TYPE을 열거형으로 선언*/
```

```

typedef enum {
    T_DIRECTIVE,
    T_INSTRUCTION,
    T_LITERAL,
    T_COMMENT
} TYPE;

/*
 * 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;                //명령어 라인 중 label
    char *operator;            //명령어 라인 중 operator
    char *operand[MAX_OPERAND]; //명령어 라인 중 operand
    char *comment;             //명령어 라인 중 comment
    char nixbpe;               //하위 6bit 사용: _ _ n i x b p e
    int op_index;              //새로 추가한 변수로 소스코드에 존재하는 명령어
    //새로 추가한 변수로 operand에 +가 포함되어 있는
    int plus_check;
    char *directive;           //새로 추가한 변수로 명령어 라인 중 지시어
    int address;               //새로 추가한 변수로 loc 저장
    char section[6];           //새로 추가한 변수로 해당 라인이 어떤 프로그램에 속해있
    //새로 추가한 변수로 타입을 명시
    TYPE type;
    int opnum;                 //object code에 들어갈 상위 2개의 16진
    //새로 추가한 변수로 타입을 명시
    int opnum;
    //object code에 들어갈 상위 2개의 16진
    //수 값을 저장
};

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

/*
 * 심볼을 관리하는 구조체이다.
 * 심볼 테이블은 심볼 이름, 심볼의 위치로 구성된다.
 */
struct symbol_unit
{
    char symbol[6];

```

```

        int addr;
        char type; //새로 추가한 변수로 symbol이 'R'(relative expression)인지 'A'(absolute
expression)인지 구분
        char* area; //새로 추가한 변수로 해당 심볼이 어떤 프로그램에 포함되어 있는지 저장
};

typedef struct symbol_unit symbol;
symbol sym_table[MAX_LINES];
static int sym_num;

/*
* 리터럴을 관리하는 구조체이다.
* 리터럴 테이블은 리터럴의 이름, 리터럴의 위치로 구성된다.
*/
struct literal_unit
{
    char literal[6];
    int addr;
    char type; //새로 추가한 변수로 literal이 'C'로 시작하는지 'X'로 시작하는지 구분
    int alloc; //새로 추가한 변수로 LORG를 만나서 메모리가 할당되었는지의 여부 체크
};

typedef struct literal_unit literal;
literal literal_table[MAX_LINES];
static int literal_num;

static int locctr;

struct CtlSection { //프로그램별로 opcode를 저장하기 위해 추가한 구조체
    char obj_code[MAX_LINES][9]; //해당 라인이 변환되는 address 저장
    int obj_line; //라인의 개수
    char section[10]; //어떤 프로그램을 관리하는지 확인하기 위해 추가한 변수
    int extdef; //지시어 EXTDEF가 나왔을 때 symbol의 개수를 저장
    int extref; //지시어 EXTREF가 나왔을 때 symbol의 개수를 저장
    symbol def[MAX_LINES]; //지시어 EXTDEF가 나왔을 때 symbol을 저장
    symbol ref[MAX_LINES]; //지시어 EXTREF가 나왔을 때 symbol을 저장
    symbol mod[MAX_LINES]; //라인별로 읽으면서 operand에 ref인 변수가 나타났을 때 저장
    int mod_num; //라인별로 읽으면서 operand에 ref인 변수가 나타나면 그 개수를 저장
    int ref_size[MAX_LINES]; //라인별로 읽으면서 operand에 ref인 변수가 나타나면 그 크기를
저장
    int lastAddr; //프로그램의 마지막 주소값을 저장
    int startAddr; //프로그램의 시작 주소값을 저장
};

```

```

typedef struct CtlSection cs;
cs control_section[MAX_LINES];
int cs_num;

//-----
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);
int search_directive(char *str); //새로 추가한 함수로 지시어에 대한 정보를 함수 안에 포함시켜 파라미터가 지시어인지 아닌지를 판별해주는 함수
static int assem_pass1(void);
//void make_opcode_output(char *file_name); //이번 프로젝트에서는 사용하지 않는 함수
int make_symtab(char*str); //새로 추가한 함수로 symbol_table을 생성해주기 위한 함수
int search_symtab(char*str); //새로 추가한 함수로 symbol_table을 탐색하여 해당하는 symbol의 주소값을 반환하는 함수
int make_literaltab(char*str); //새로 추가한 함수로 literal_table을 생성하는 함수
void addr_literal_tab(); //새로 추가한 함수로 make_literaltab()에서 넣어준 literal들의 주소를 넣어주는 함수이다.
int search_literaltab(char*str); //새로 추가한 함수로 literal_table을 탐색하여 해당하는 literal의 주소값을 반환하는 함수
void literal_return(); //새로 추가한 함수로 literal table에 있는 literal을 메모리에 할당해주기 위해 literal값을 반환해주는 함수
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);

```

2)my_assembler_20160433.c

```

/*
 * 파일명 : my_assembler_20160433.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 <ctype.h>
```

```
// 파일명의 "00000000"은 자신의 학번으로 변경할 것.
```

```
#include "my_assembler_20160433.h"
```

```
#pragma warning(disable:4996)
```

```
#pragma warning(disable:4018)
```

```
#pragma warning(disable:4047)
```

```
#pragma warning(disable:4024)
```

```
/
```

★

- * 설명 : 사용자로부터 어셈블리 파일을 받아서 명령어의 OPCODE를 찾아 출력한다.

- * 매개 : 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_syntab_output(NULL); //symbol table을 화면에 출력
```

```
    make_syntab_output("syntab_20160433.txt"); // symbol table을 파일로 출력
```

```
    //make_literaltab_output(NULL); //literal table을 화면에 출력
```

```
    make_literaltab_output("literaltab_20160433.txt"); // literal table을 파일로 출력
```

```

if (assem_pass2() < 0) //object code 생성
{
    printf("assem_pass2: 패스2 과정에서 실패하였습니다. \n");
    return -1;
}

//make_objectcode_output(NULL); //결과를 화면에 출력
make_objectcode_output("output_20160433.txt"); // 결과를 파일로 출력

return 0;
}

```

/ *

* 설명 : 프로그램 초기화를 위한 자료구조 생성 및 파일을 읽는 함수이다.
 * 매개 : 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.
 * 매개 : 기계어 목록 파일
 * 반환 : 정상종료 = 0 , 에러 < 0
 * 주의 : 기계어 목록파일 형식은 자유롭게 구현한다. 예시는 다음과 같다.
 *
 *

```

*           | 이름 | 오퍼랜드의 갯수 | 형식 | 기계어 코드 | NULL |
*
=====

*
*
-----

*/
int init_inst_file(char *inst_file)
{
    FILE *file;
    int errno;

    /* add your code here */
    file = fopen(inst_file, "r"); //읽기모드로 파일을 연다

    if (file == NULL) { //파일 열기에 실패하면 에러
        errno = -1;
    }
    else {
        while (1) { //동적할당 후 기계어 목록파일의 형식에 따라 해당하는 값들을 기계어 목록 테이블에 저장.

            inst_table[inst_index] = (inst*)malloc(sizeof(inst));
            memset(inst_table[inst_index], 0, sizeof(inst));

            fscanf(file, "%s %d %d %x", &inst_table[inst_index]->instruction,
            &inst_table[inst_index]->operand_num, &inst_table[inst_index]->format, &inst_table[inst_index]->opcode);
            inst_index++;

            if (inst_index == 59)
                break;
        }
        fclose(file);
    }

    return errno;
}

/

```

- * 설명 : 어셈블리 할 소스코드를 읽어 소스코드 테이블(input_data)를 생성하는 함수이다.
- * 매개 : 어셈블리할 소스파일명
- * 반환 : 정상종료 = 0 , 에러 < 0
- * 주의 : 라인단위로 저장한다.

```

*
*
-----

*/
int init_input_file(char *input_file)
{
    FILE *file;
    int errno;

    /* add your code here */
    int buf_size = 100;

    file = fopen(input_file, "r"); //읽기모드로 파일을 연다
    if (file == NULL) { //파일 열기에 실패하면 에러
        errno = -1;
    }
    else {
        while (1) { //동적할당 후 파일을 라인단위로 읽어 소스코드 테이블에 저장
            input_data[line_num] = (char*)malloc(sizeof(char)*buf_size);
            memset(input_data[line_num], 0, buf_size);

            if (fgets(input_data[line_num], buf_size, file) == NULL) { //파일의 끝에 도달하면
while문 탈출
                break;
            }

            if (input_data[line_num][strlen(input_data[line_num]) - 1] == '\n') { //라인의 끝
에 존재하는 '\n'를 '\0'로 대체
                input_data[line_num][strlen(input_data[line_num]) - 1] = '\0';
            }

            line_num++;
        }
        fclose(file);
    }
    return errno;
}

/

```

*

- * 설명 : 소스 코드를 읽어와 토큰단위로 분석하고 토큰 테이블을 작성하는 함수이다.
- * 패스 1로 부터 호출된다.
- * 매개 : 파싱을 원하는 문자열
- * 반환 : 정상종료 = 0 , 에러 < 0

* 주의 : my_assembler 프로그램에서는 라인단위로 토큰 및 오브젝트 관리를 하고 있다.

*

*/

int token_parsing(char *str)

{

/* add your code here */

char *tmp_str;

char *tok;

int inst_num;

int tok_num;

int dir_num;

int inst_check = 0;

tmp_str = (char*)malloc(sizeof(char));

tmp_str = str;

token_table[token_line] = (token*)malloc(sizeof(token));

memset(token_table[token_line], 0, sizeof(token));

token_table[token_line]->label = NULL;

token_table[token_line]->operator = NULL;

token_table[token_line]->operand[0] = NULL;

token_table[token_line]->operand[1] = NULL;

token_table[token_line]->comment = NULL;

token_table[token_line]->directive = NULL;

if (tmp_str[0] == '.') { //입력문자열이 주석이라면 토큰테이블의 comment변수에 저장

token_table[token_line]->comment = (char*)malloc(sizeof(char));

token_table[token_line]->comment = tmp_str;

token_table[token_line]->type = T_COMMENT;

token_line++;

return 0;

}

tok = strtok(tmp_str, " Wt");// " Wt"라는 구분자로 토큰분리

if (tok == NULL) //분리할 문자열이 없으면 리턴

return 0;

if (tok[0] == '+') { //기계어에 +가 추가되어 있으면

inst_check = 1;

char inst_four[100];

strcpy(inst_four, tok);

```

int size = sizeof(tok);
for (int i = 0; i < size; i++) {
    tok[i] = inst_four[i + 1];
}
tok[size] = 'W0';
}
if ((inst_num = search_opcode(tok)) >= 0) { //잘린 문자열이 기계어라면
    token_table[token_line] -> operator = (char*)malloc(sizeof(char));
    token_table[token_line] -> operator = tok;
    token_table[token_line] -> op_index = inst_num;
    if (inst_table[token_table[token_line] -> op_index] -> format == 3) {
        if (inst_check) {
            //기계어에 +가 추가되어 있으면 토큰테이블에 따로 저장
            token_table[token_line] -> plus_check = 1;
            //4형식명령어이므로 e를 표시
            token_table[token_line] -> nixbpe |= E;
        }
        else {
            //기계어에 +가 추가되어 있지 않다면 pc relative를 사용할 것이므로 p를
            token_table[token_line] -> nixbpe |= P;
        }
    }
    if ((tok_num = inst_table[inst_num] -> operand_num) != 0) { //operand가 있다면
        token_table[token_line] -> operand[0] = (char*)malloc(sizeof(char));
        if (tok_num == 1) { //operand 개수가 1개일 때
            tok = strtok(NULL, " Wt");
            token_table[token_line] -> operand[0] = tok;
            if (strstr(tok, ",") != NULL) { //BUFFER,x 같이 배열을 쓰는 경우에 토큰
                token_table[token_line] -> operand[1] =
                    (char*)malloc(sizeof(char));
                memset(token_table[token_line] -> operand[1], 0, 10);
                token_table[token_line] -> operand[0] = strtok_s(tok, ",",
                    &token_table[token_line] -> operand[1]);
                if (!strcmp(token_table[token_line] -> operand[1], "X")) {
                    token_table[token_line] -> nixbpe |= X;
                }
            }
        }
        else { //operand 개수가 2개일 때
            tok = strtok(NULL, ",");
            token_table[token_line] -> operand[0] = tok;

```

표시

분리

```

        tok = strtok(NULL, " Wt");
        token_table[token_line]->operand[1] = (char*)malloc(sizeof(char));
        token_table[token_line]->operand[1] = tok;
    }

    //addressing 방식이
    if (strstr(token_table[token_line]->operand[0], "@") != NULL) { //indirection
addressing일 때
        token_table[token_line]->nixbpe |= N;
    }
    else if (strstr(token_table[token_line]->operand[0], "#") != NULL) { //immediate
addressing일 때
        token_table[token_line]->nixbpe |= I;
        token_table[token_line]->nixbpe ^= P;
    }
    else { // 둘다 아니면
        if (inst_table[inst_num]->format == 3) { //그런데 기계어의 형식이 3형
식/4형식이라면
            token_table[token_line]->nixbpe |= N;
            token_table[token_line]->nixbpe |= I;
        }
    }

}
else { //operand가 없는 3형식 명령어라면
    if (inst_table[inst_num]->format == 3) {
        token_table[token_line]->nixbpe |= N;
        token_table[token_line]->nixbpe |= I;
    }
}

if (tok != NULL) { //comment 분리
    token_table[token_line]->comment = (char*)malloc(sizeof(char));
    tok = strtok(NULL, "W0");
    token_table[token_line]->comment = tok;
}
token_table[token_line]->type = T_INSTRUCTION;
}

else if ((dir_num = search_directive(tok)) >= 0) { //잘린 문자열이 지시어라면
    token_table[token_line]->directive = (char*)malloc(sizeof(char));
    token_table[token_line]->directive = tok;
    if (dir_num == 1) { //지시어 뒤에 다른 정보가 포함되어 있을 때
        token_table[token_line]->operand[0] = (char*)malloc(sizeof(char));
        tok = strtok(NULL, " Wt");
    }
}

```

```

token_table[token_line]->operand[0] = tok;
if (strstr(tok, ",") != NULL) {
    token_table[token_line]->operand[0] = strtok(tok, ",");
    token_table[token_line]->operand[1] = (char*)malloc(sizeof(char));
    token_table[token_line]->operand[1] = strtok(NULL, ",");
    if (tok != NULL) {
        token_table[token_line]->operand[2] =
(char*)malloc(sizeof(char));
        token_table[token_line]->operand[2] = strtok(NULL, ",");
    }
}

}

if (tok != NULL) { //comment 분리
    token_table[token_line]->comment = (char*)malloc(sizeof(char));
    tok = strtok(NULL, "W0");
    token_table[token_line]->comment = tok;
}
token_table[token_line]->type = T_DIRECTIVE;
}

else { //잘린 문자열이 label이라면
    token_table[token_line]->label = (char*)malloc(sizeof(char));
    token_table[token_line]->label = tok;
    tok = strtok(NULL, "Wt");

    if (tok[0] == '+') { //기계어코드에 +가 추가되어 있으면
        inst_check = 1;
        char inst_four[100];
        strcpy(inst_four, tok);
        int size = sizeof(tok);
        for (int i = 0; i < size; i++) {
            tok[i] = inst_four[i + 1];
        }
        tok[size] = 'W0';
    }

    if ((inst_num = search_opcode(tok)) >= 0) { //잘린 문자열이 기계어코드라면
        token_table[token_line]->operator = (char*)malloc(sizeof(char));
        token_table[token_line]->operator = tok;
        token_table[token_line]->op_index = inst_num;

        if (inst_table[token_table[token_line]->op_index]->format == 3) {
            if (inst_check) {

```



```

//+가 붙은 명령어라면 토큰테이블에 따로 저장
token_table[token_line]->plus_check = 1;
//4형식명령어이므로 e를 표시
token_table[token_line]->nixbpe |= E;
}
else {
//+가 붙은 명령어가 아니라면 pc relative를 사용할 것이므로 p
를 표시

token_table[token_line]->nixbpe |= P;
}
}

if ((tok_num = inst_table[inst_num]->operand_num) != 0) { //operand가 있다면
token_table[token_line]->operand[0] = (char*)malloc(sizeof(char));
if (tok_num == 1) { //operand의 개수가 1개일 때
tok = strtok(NULL, " Wt");
token_table[token_line]->operand[0] = tok;
if (strstr(tok, ",") != NULL) { //BUFFER,x 같이 배열을 쓰는 경
우에 토큰분리

token_table[token_line]->operand[1] =
(char*)malloc(sizeof(char));

memset(token_table[token_line]->operand[1], 0, 10);
token_table[token_line]->operand[0] = strtok_s(tok,
",", &token_table[token_line]->operand[1]);

if (!strcmp(token_table[token_line]->operand[1], "X"))
{
token_table[token_line]->nixbpe |= X;
}
}
}
else { //operand 개수가 2개일 때
tok = strtok(NULL, ",");
token_table[token_line]->operand[0] = tok;
tok = strtok(NULL, " Wt");
token_table[token_line]->operand[1] =
(char*)malloc(sizeof(char));

token_table[token_line]->operand[1] = tok;
}
//addressing 방식이
if (strstr(token_table[token_line]->operand[0], "@" ) != NULL)
{ //indirection addressing일 때

token_table[token_line]->nixbpe |= N;
}
}
}

```

```

else if (strstr(token_table[token_line]->operand[0], "#") != NULL)
{
    token_table[token_line]->nixbpe |= I;
    token_table[token_line]->nixbpe ^= P;
}
else { // 둘다 아니라면
    if (inst_table[inst_num]->format == 3) { //그런데 기계어의 형
        token_table[token_line]->nixbpe |= N;
        token_table[token_line]->nixbpe |= I;
    }
}
}
else { //operand가 없는 3형식 명령어라면
    if (inst_table[inst_num]->format == 3) {
        token_table[token_line]->nixbpe |= N;
        token_table[token_line]->nixbpe |= I;
    }
}
token_table[token_line]->type = T_INSTRUCTION;
}
else if ((dir_num = search_directive(tok)) >= 0) { //잘린 문자열이 지시어라면
    token_table[token_line]->directive = (char*)malloc(sizeof(char));
    token_table[token_line]->directive = tok;
    if (dir_num == 1) { //지시어 뒤에 다른 정보가 포함되어 있을 때
        token_table[token_line]->operand[0] = (char*)malloc(sizeof(char));
        tok = strtok(NULL, " Wt");
        token_table[token_line]->operand[0] = tok;
        if (strstr(tok, ",") != NULL) {
            token_table[token_line]->operand[0] = strtok(tok, ",");
            token_table[token_line]->operand[1] =
(char*)malloc(sizeof(char));
            token_table[token_line]->operand[1] = strtok(NULL, ",");
            if (tok != NULL) {
                token_table[token_line]->operand[2] =
(char*)malloc(sizeof(char));
                token_table[token_line]->operand[2] = strtok(NULL,
",");
            }
        }
    }
}
token_table[token_line]->type = T_DIRECTIVE;
}
if (tok != NULL) { //comment 분리

```

```

        token_table[token_line]->comment = (char*)malloc(sizeof(char));
        tok = strtok(NULL, "W0");
        token_table[token_line]->comment = tok;
    }
}

token_line++;
return 0;
}

/
-----
* 설명 : 입력 문자열이 기계어 코드인지를 검사하는 함수이다.
* 매개 : 토큰 단위로 구분된 문자열
* 반환 : 정상종료 = 기계어 테이블 인덱스, 에러 < 0
* 주의 :
*
*
-----
*/
int search_opcode(char *str)
{
    /* add your code here */
    for (int i = 0; i < inst_index; i++) {
        if (!strcmp(inst_table[i]->instruction, str)) { //입력된 문자열이 inst_table의 명령어와 일치
            한다면 inst_table의 index를 반환
                return i;
        }
    }
    return -1;
}

//추가한 함수
/
-----
* 설명 : 입력 문자열이 지시어인지를 검사하는 함수이다.
* 매개 : 토큰 단위로 구분된 문자열
* 반환 : 정상종료 >= 0, 에러 < 0
* 주의 :
*
*
-----
*/
int search_directive(char *str)

```

```

{
    char                *directive_table[] = {
"START","END","BYTE","WORD","RESB","RESW","CSECT","EXTDEF","EXTREF","EQU","ORG","LTORG" };
    int directive_num[] = { 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0 };

    for (int i = 0; i < sizeof(directive_num) / sizeof(int); i++) {
        if (!strcmp(directive_table[i], str)) { //입력된 문자열이 directive_table의 지시어와 일치한
다면 뒤에 정보가 포함되어 있는 지시어라면 1, 아니라면 0 리턴
            return directive_num[i];
        }
    }
    return -1;
}

```

/

* 설명 : 어셈블리 코드를 위한 패스1과정을 수행하는 함수이다.

* 패스1에서는..

* 1. 프로그램 소스를 스캔하여 해당하는 토큰단위로 분리하여 프로그램 라인별 토큰
테이블을 생성한다.

* 매개 : 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)
```

```
{
```

```
    /* add your code here */
```

```
    for (int i = 0; i < line_num; i++) { //입력파일에서 라인별로 읽어들이 소스들을 각각 토큰분리시킨
다.
```

```
        if (token_parsing(input_data[i]) < 0) {
```

```
            return -1;
```

```
        }
```

```
    }
```

```
    now_area = (char*)malloc(sizeof(char));
```

```
    //토큰 분리한 것들을 가지고 loc를 포함한 기타 필요한 정보들 저장
```

```
    for (int i = 0; i < token_line; i++) {
```

```

if (token_table[i]->type == T_DIRECTIVE) {
    if (!strcmp(token_table[i]->directive, "START")) { //프로그램이 시작한다면
        locctr = atoi(token_table[i]->operand[0]);
        token_table[i]->address = locctr;
        now_area = token_table[i]->label;
        strcpy(token_table[i]->section, now_area);
        if (make_symtab(token_table[i]->label) < 0) {
            fprintf(stderr, "duplicate symbol error\n");
            return -1;
        }
        continue;
    }
    else if (!strcmp(token_table[i]->directive, "CSECT")) { //새로운 sub program이
        control_section[cs_num].lastAddr = locctr; // 이전 프로그램의 마지막
        cs_num++;
        locctr = 0;
        token_table[i]->address = locctr;
        now_area = token_table[i]->label;
        strcpy(token_table[i]->section, now_area);
        if (make_symtab(token_table[i]->label) < 0) {
            fprintf(stderr, "duplicate symbol error\n");
            return -1;
        }
        continue;
    }
}

if (token_table[i]->label != NULL) {
    if (make_symtab(token_table[i]->label) < 0) {
        fprintf(stderr, "duplicate symbol error\n");
        return -1;
    }
}

token_table[i]->address = locctr;
strcpy(token_table[i]->section, now_area);

if (token_table[i]->type == T_INSTRUCTION) {
    if (token_table[i]->plus_check) { //명령어의 형식에 맞춰 locctr값 증가
        locctr += 4;
    }
    else {
        locctr += inst_table[token_table[i]->op_index]->format;
    }
}

```

시작된다면

주소값을 저장

```

    }
    if (token_table[i]->operand[0] != NULL) {
        if (strstr(token_table[i]->operand[0], "=") != NULL) { //indirection
            make_literaltab(token_table[i]->operand[0]);
        }
    }
}
else if (token_table[i]->type == T_DIRECTIVE){
    if (!strcmp(token_table[i]->directive, "WORD")) { //locctr을 3 증가시킴
        locctr += 3;
    }
    else if (!strcmp(token_table[i]->directive, "RESW")) { //locctr을 (3 * 피연산자값)
        locctr += 3* atoi(token_table[i]->operand[0]);
    }
    else if (!strcmp(token_table[i]->directive, "RESB")) { //locctr을 피연산자값만큼 증
        locctr += atoi(token_table[i]->operand[0]);
    }
    else if (!strcmp(token_table[i]->directive, "BYTE")) { //locctr을 1 증가시킴
        locctr += 1;
    }
    else if (!strcmp(token_table[i]->directive, "EQU")){
        if (!strcmp(token_table[i]->operand[0], "*")) { // 피연산자가 *인 경우
            sym_table[sym_num-1].addr = locctr; //현재 locctr의 값을 주
        }
        else { // 그렇지 않은 경우 피연산자를 통해 주소값을 계산하여 저장
            if (strstr(token_table[i]->operand[0], "-")) {
                char op_t[100];
                strcpy(op_t, token_table[i]->operand[0]);
                char* op1 = (char*)malloc(sizeof(char));
                char* op2 = (char*)malloc(sizeof(char));
                memset(op1, 0, 10);
                memset(op2, 0, 10);
                op1 = strtok_s(op_t, "-", &op2);
                sym_table[sym_num-1].addr = search_symtab(op1) -
                search_symtab(op2);
                token_table[i]->address = sym_table[sym_num -
                1].addr;
                sym_table[sym_num - 1].type = 'A';
            }
        }
    }
}

```

addressing을 때

만큼 증가시킴

가시킴

소로 저장

search_symtab(op2);

1].addr;

```

    }
    else if (!strcmp(token_table[i]->directive, "LORG")) { //프로그램에 나왔던 literal
을 저장
        addr_literal_tab();
    }
    else if (!strcmp(token_table[i]->directive, "END")) { //프로그램에 나왔던 literal을
저장
        addr_literal_tab();
    }
}
}
return 0;
}

```

```

/
-----
* 설명 : symbol table을 생성하는 함수이다.
* 매개 : symbol table에 추가할 symbol명
* 반환 : 정상종료 >= 0, 에러 < 0
* 주의 : 없음
*
-----
*/
int make_symtab(char*str) {
    for (int i = 0; i < sym_num; i++) {
        if (!strcmp(sym_table[i].area, now_area)) { //symtab에서 area도 같고 str도 같은 symbol이
존재하면 에러
            if (!strcmp(sym_table[i].symbol, str)) {
                return -1;
            }
        }
    }
    strcpy(sym_table[sym_num].symbol, str); //symtab에 존재하지 않는 symbol이라면 추가
    sym_table[sym_num].area = (char*)malloc(sizeof(char));
    sym_table[sym_num].area = now_area;
    sym_table[sym_num].addr = locctr;
    sym_table[sym_num].type = 'R';
    sym_num++;
    return 0;
}

```

```

/
-----
* 설명 : symbol table을 탐색해서 해당하는 symbol의 주소값을 반환하는 함수이다.

```

* 매개 : 탐색할 symbol명

* 반환 : 정상종료 >= 0, 에러 < 0

* 주의 : 없음

```

* -----
*/
int search_syntab(char*str) {
    for (int i = 0; i < sym_num; i++) {
        if (!strcmp(sym_table[i].area, now_area)) { //syntab에서 area도 같고 str도 같은 symbol이
존재하면 symbol의 주소값 반환
            if (!strcmp(sym_table[i].symbol, str)) {
                return sym_table[i].addr;
            }
        }
    }
    return -1;
}

```

/ *

* 설명 : literal table을 생성하는 함수이다.

* 매개 : literal table에 추가할 literal명

* 반환 : 정상종료 >= 0, 에러 < 0

* 주의 : 없음

```

* -----
*/
int make_literaltab(char*str) {

    char *tmp_str = (char*)malloc(sizeof(char));
    strcpy(tmp_str, str);
    tmp_str++;

    for (int i = 0; i < literal_num; i++) { //이미 literal table에 존재하는 literal인지 확인
        if (!strcmp(literal_table[i].literal, tmp_str)) {
            return -1;
        }
    }
    strcpy(literal_table[literal_num].literal, tmp_str); //literal_table에 존재하지 않는 literal이라면 추가
    literal_num++;
    return 0;
}

```

/ *

* 설명 : literal table의 literal들의 주소를 넣어주는 함수이다.

* 매개 : 없음

* 반환 : 없음

* 주의 : 없음

*

*/

void addr_literal_tab() {

 char* tmp_literal = (char*)malloc(sizeof(char));

 int literalSize;

 for (int i = 0; i < literal_num; i++) {

 if (!literal_table[i].alloc) {

 literal_table[i].addr = locctr;

 if (literal_table[i].literal[0] == 'C' || literal_table[i].literal[0] == 'c') { //literal이
 'C'인지 확인

 literal_table[i].type = 'C';

 strcpy(tmp_literal, literal_table[i].literal);

 tmp_literal += 2;

 literalSize = strlen(tmp_literal);

 tmp_literal[literalSize - 1] = '\0';

 strcpy(literal_table[i].literal, tmp_literal);

 for (int j = literalSize; j < literalSize + 2; j++) {

 literal_table[i].literal[j] = '\0';

 }

 locctr += literalSize - 1; //char개수만큼 locctr 증가

 literal_table[i].alloc = 1; //주소를 넣어줬다고 체크

 }

 else if (literal_table[i].literal[0] == 'X' || literal_table[i].literal[0] == 'x') {
//literal이 'X'인지 확인

 literal_table[i].type = 'X';

 strcpy(tmp_literal, literal_table[i].literal);

 tmp_literal += 2;

 literalSize = strlen(tmp_literal);

 tmp_literal[literalSize - 1] = '\0';

 strcpy(literal_table[i].literal, tmp_literal);

 for (int j = literalSize; j < literalSize + 2; j++) {

 literal_table[i].literal[j] = '\0';

 }

 locctr += (literalSize - 1)/2; //byte 개수의 절반만큼 locctr증가

 literal_table[i].alloc = 1; //주소를 넣어줬다고 체크

 }

}

```

    }
}

/
-----
* 설명 : literal table을 탐색해서 해당하는 literal의 주소값을 반환하는 함수이다.
* 매개 : 탐색할 literal명
* 반환 : 정상종료 >= 0, 에러 < 0
* 주의 : 없음
* -----
*/
int search_literaltab(char*str) {
    char* tmp_literal = (char*)malloc(sizeof(char));
    int literalSize;
    strcpy(tmp_literal, str);
    tmp_literal += 3;
    literalSize = strlen(tmp_literal);
    tmp_literal[literalSize - 1] = '\0';

    for (int i = 0; i < literal_num; i++) { //literal table에 찾고자 하는 literal이 존재하는지 확인
        if (!strcmp(literal_table[i].literal, tmp_literal)) {
            literal_table[i].alloc = 0; //메모리에 할당해주기 위해 체크
            return literal_table[i].addr;
        }
    }

    return -1;
}

/
-----
* 설명 : literal table을 탐색해서 메모리에 할당해주는 함수이다 .
* 매개 : 없음
* 반환 : 없음
* 주의 : 없음
* -----
*/
void literal_return() {
    for (int i = 0; i < literal_num; i++) {
        if (!literal_table[i].alloc) {
            if (literal_table[i].type == 'X') { //literal의 type이 'X'라면 literal값을 그대로 메모
리에 할당

sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%s", literal_table[i].literal);

```

```

        }
        else {
            for (int j = 0; j < strlen(literal_table[i].literal); j++) { //literal의 type이
'C'라면 각 char 값의 ASCII code를 메모리에 할당

sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%s%X",
control_section[cs_num].obj_code[control_section[cs_num].obj_line], literal_table[i].literal[j]);

        }
    }
    control_section[cs_num].obj_line++;
    literal_table[i].alloc = 1;
}
}
}

```

/ ★

* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
* 여기서 출력되는 내용은 명령어 옆에 OPCODE가 기록된 표(과제 5번) 이다.
* 매개 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프로그램의 결과를 표준출력으로 보내어
화면에 출력해준다.
* 또한 과제 5번에서만 쓰이는 함수이므로 이후의 프로젝트에서는 사용되지 않는다.
*

```

*/
// void make_opcode_output(char *file_name)
// {
//     /* add your code here */
// }

```

/ ★

* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
* 여기서 출력되는 내용은 SYMBOL별 주소값이 저장된 TABLE이다.
* 매개 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프로그램의 결과를 표준출력으로 보내어
화면에 출력해준다.
*
*

```

*/
void make_symtab_output(char *file_name)
{
    /* add your code here */
    FILE* fp;
    if (file_name != NULL) { //인자로 NULL값이 들어오지 않는 경우 파일에 기록
        fp = fopen(file_name, "w+t");
    }
    else { //인자로 NULL값이 들어오는 경우 표준출력으로 화면에 출력
        fp = stdout;
    }

    char *tmp_area = (char*)malloc(sizeof(char));
    strcpy(tmp_area, sym_table[0].area);
    //SYMBOL별 주소값이 저장된 TABLE 내용을 출력
    for (int i = 0; i < sym_num; i++) {
        if (strcmp(tmp_area, sym_table[i].area)) {
            fprintf(fp, "Wn");
            strcpy(tmp_area, sym_table[i].area);
        }
        fprintf(fp, "%sWtWtWt%XWn", sym_table[i].symbol, sym_table[i].addr);
    }

    if (file_name != NULL) { //파일을 열은 경우 닫아줌.
        fclose(fp);
    }
}

```

/ ★

* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
 * 여기서 출력되는 내용은 LITERAL별 주소값이 저장된 TABLE이다.
 * 매개 : 생성할 오브젝트 파일명
 * 반환 : 없음
 * 주의 : 만약 인자로 NULL값이 들어온다면 프로그램의 결과를 표준출력으로 보내어
 * 화면에 출력해준다.

*
 *

```

*/
void make_literaltab_output(char *file_name)
{
    /* add your code here */
    FILE* fp;

```

```

if (file_name != NULL) { //인자로 NULL값이 들어오지 않는 경우 파일에 기록
    fp = fopen(file_name, "w+t");
}
else { //인자로 NULL값이 들어오는 경우 표준출력으로 화면에 출력
    fp = stdout;
}

//LITERAL별 주소값이 저장된 TABLE 내용을 아래의 형식으로 출력
for (int i = 0; i < literal_num; i++) {
    fprintf(fp, "%sWtWtWt%XWn", literal_table[i].literal, literal_table[i].addr);
}

if (file_name != NULL) { //파일을 열은 경우 닫아줌.
    fclose(fp);
}
}

```

/ *

```

* 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)
{
    /* add your code here */
    int pc;
    int target;
    int addr;
    char* reg[10] = { "A", "X", "L", "B", "S", "T", "F", "", "PC", "SW" };
    cs_num = 0;

    //토큰 분리한 것들을 가지고 pass2 시작
    for (int i = 0; i < token_line; i++) {
        if (token_table[i]->type == T_DIRECTIVE) {
            if (!strcmp(token_table[i]->directive, "START")) { //프로그램이 시작하면
                control_section[cs_num].startAddr = atoi(token_table[i]->operand[0]);
                now_area = token_table[i]->section;
            }
        }
    }
}

```

면

값들 저장

```
        strcpy(control_section[cs_num].section,now_area);
    }
    else if (!strcmp(token_table[i]->directive, "CSECT")) { //서브프로그램이 시작하
        now_area = token_table[i]->section;
        cs_num++;
        control_section[cs_num].startAddr = 0;
        strcpy(control_section[cs_num].section, now_area);
    }
    else if (!strcmp(token_table[i]->directive, "EXTDEF")) { //EXTDEF 뒤에 나오는
        strcpy(control_section[cs_num].def[control_section[cs_num].extdef].symbol, token_table[i]->operand[0]);
        control_section[cs_num].def[control_section[cs_num].extdef].addr =
        search_symtab(token_table[i]->operand[0]);
        control_section[cs_num].extdef++;
        if (token_table[i]->operand[1] != NULL) {

            strcpy(control_section[cs_num].def[control_section[cs_num].extdef].symbol, token_table[i]->operand[1]);

            control_section[cs_num].def[control_section[cs_num].extdef].addr =
            search_symtab(token_table[i]->operand[1]);
            control_section[cs_num].extdef++;
            if (token_table[i]->operand[2] != NULL) {

                strcpy(control_section[cs_num].def[control_section[cs_num].extdef].symbol, token_table[i]->operand[2]);

                control_section[cs_num].def[control_section[cs_num].extdef].addr =
                search_symtab(token_table[i]->operand[2]);
                control_section[cs_num].extdef++;
            }
        }
    }
    else if (!strcmp(token_table[i]->directive, "EXTREF")) { //EXTREF 뒤에 나오는 값
        strcpy(control_section[cs_num].ref[control_section[cs_num].extref].symbol, token_table[i]->operand[0]);
        control_section[cs_num].extref++;
        if (token_table[i]->operand[1] != NULL) {

            strcpy(control_section[cs_num].ref[control_section[cs_num].extref].symbol, token_table[i]->operand[1]);
            control_section[cs_num].extref++;
            if (token_table[i]->operand[2] != NULL) {
```

들 저장

```

strcpy(control_section[cs_num].ref[control_section[cs_num].extref].symbol, token_table[i]->operand[2]);
        control_section[cs_num].extref++;
    }
}
}
else if (!strcmp(token_table[i]->directive, "LORG")) { //프로그램에 나왔던 literal
값들을 메모리에 저장
        literal_return();
        continue;
}
else if (!strcmp(token_table[i]->directive, "END")) { //프로그램에 나왔던 literal값
들을 메모리에 저장
        literal_return();
        control_section[cs_num].lastAddr = locctr;
        cs_num++;
        continue;
}
else if (!strcmp(token_table[i]->directive, "BYTE")) {
    char * tmp_data = (char*)malloc(sizeof(char));
    char *tmp_token = NULL;
    strcpy(tmp_data, token_table[i]->operand[0]);
    tmp_token = strtok(tmp_data, "");
    tmp_token = strtok(NULL, "");
    if(token_table[i]->operand[0][0] == 'X' ||
token_table[i]->operand[0][0] == 'x') {

        sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%s", tmp_token);
        }
        else if(token_table[i]->operand[0][0] == 'C' ||
token_table[i]->operand[0][0] == 'c'){

            sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%X", tmp_token[0]);
            for (int j = 1; j < strlen(tmp_token); j++) {

                sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%s%X",
control_section[cs_num].obj_code[control_section[cs_num].obj_line], tmp_token[j]);
            }
        }
    }
}
else if (!strcmp(token_table[i]->directive, "WORD")) {

        if (isdigit(token_table[i]->operand[0][0])) { // 숫자라면

            sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%X",

```

```

atoi(token_table[i]->operand[0][0]));
    }
    else { // 문자일때의 처리
        if (strstr(token_table[i]->operand[0], "-")) {
            char op_t[100];
            int ref_check = 0;
            strcpy(op_t, token_table[i]->operand[0]);
            char* op1 = (char*)malloc(sizeof(char));
            char* op2 = (char*)malloc(sizeof(char));
            memset(op1, 0, 10);
            memset(op2, 0, 10);
            op1 = strtok_s(op_t, "-", &op2);
            for (int j = 0; j < control_section[cs_num].extref;
j++) {
                if (!strcmp(op1,
control_section[cs_num].ref[j].symbol)) {
                    ref_check = 1;
                }
                if (!strcmp(op2,
control_section[cs_num].ref[j].symbol)) {
                    ref_check = 2;
                }
            }
            if (ref_check != 0) {

sprintf(control_section[cs_num].mod[control_section[cs_num].mod_num].symbol, "+%s", op1);

control_section[cs_num].mod[control_section[cs_num].mod_num].addr = token_table[i]->address;

control_section[cs_num].ref_size[control_section[cs_num].mod_num] = 6;
                control_section[cs_num].mod_num++;

sprintf(control_section[cs_num].mod[control_section[cs_num].mod_num].symbol, "-%s", op2);

control_section[cs_num].mod[control_section[cs_num].mod_num].addr = token_table[i]->address;

control_section[cs_num].ref_size[control_section[cs_num].mod_num] = 6;
                control_section[cs_num].mod_num++;

sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%06X", 0);
            }
        }
    }
}

```



```

    }
}
else if (token_table[i]->type == T_INSTRUCTION) {
    int op = 252;
    token_table[i]->opnum = op & inst_table[token_table[i]->op_index]->opcode;
    if ((token_table[i]->nixbpe & N) == N) {
        token_table[i]->opnum += 2;
    }
    if ((token_table[i]->nixbpe & I) == I) {
        token_table[i]->opnum += 1;
    }
    if (token_table[i]->plus_check == 1) { //명령어가 4형식일 때

sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line],          "%02X%01X%05X",
token_table[i]->opnum, token_table[i]->nixbpe & 0b001111, 0);
        for (int j = 0; j < control_section[cs_num].extref; j++) {
            if (!strcmp(token_table[i]->operand[0],
control_section[cs_num].ref[j].symbol)) {

sprintf(control_section[cs_num].mod[control_section[cs_num].mod_num].symbol, "+%s",
token_table[i]->operand[0]);

control_section[cs_num].mod[control_section[cs_num].mod_num].addr = token_table[i]->address + 1;

control_section[cs_num].ref_size[control_section[cs_num].mod_num] = 5;
                control_section[cs_num].mod_num++;
            }
        }
        control_section[cs_num].obj_line++;
        continue;
    }
    if (inst_table[token_table[i]->op_index]->format == 1) { //1형식일 때

sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line],          "%02X",
inst_table[token_table[i]->op_index]->opcode);
    }
    else if (inst_table[token_table[i]->op_index]->format == 2) { //2형식일 때
        if(inst_table[token_table[i]->op_index]->operand_num == 1){ //피연산
자의 개수가 1개일 때

            int reg_num;
            for (int j = 0; j < 10; j++) {
                if (!strcmp(token_table[i]->operand[0], reg[j])){
                    reg_num = j;
                    break;

```

```

        }
    }

    sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%02X%01X%01X",
    token_table[i]->opnum, reg_num, 0);
    }
    else if (inst_table[token_table[i]->op_index]->operand_num == 2) { //
피연산자의 개수가 2개일 때

        int reg_num1, reg_num2;
        for (int j = 0; j < 6; j++) {
            if (!strcmp(token_table[i]->operand[0], reg[j])) {
                reg_num1 = j;
            }
            if (!strcmp(token_table[i]->operand[1], reg[j])) {
                reg_num2 = j;
            }
        }

        sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%02X%01X%01X",
        token_table[i]->opnum, reg_num1, reg_num2);
    }
}

else if (inst_table[token_table[i]->op_index]->format == 3) { //3형식일 때
    if (inst_table[token_table[i]->op_index]->operand_num == 0) { //피연
산자가 존재하지 않을 때

        sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%02X%04X",
        token_table[i]->opnum, 0);

        control_section[cs_num].obj_line++;
        continue;
    }
    else if (token_table[i]->operand[0] != NULL) { //피연산자가 존재할 때
        if (strstr(token_table[i]->operand[0], "#") != NULL) {
//immediate addressing일 때

            token_table[i]->operand[0]++;

            sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%02X%04X",
            token_table[i]->opnum, atoi(token_table[i]->operand[0]));

            control_section[cs_num].obj_line++;
            continue;
        }
        if (strstr(token_table[i]->operand[0], "@") != NULL){
//indirection addressing일 때

            token_table[i]->operand[0]++;

```

```

    }
    if (strstr(token_table[i]->operand[0], "=") != NULL) { //피연산
자가 literal일 때
        if ((target =
search_literaltab(token_table[i]->operand[0])) < 0) {
            fprintf(stderr, "no literal");
        }
    }
    else {
        target = search_symtab(token_table[i]->operand[0]);
    }
    pc = token_table[i + 1]->address;
    addr = target - pc;

    sprintf(control_section[cs_num].obj_code[control_section[cs_num].obj_line], "%02X%01X%03X",
token_table[i]->opnum, token_table[i]->nixbpe & 0b001111, addr & 0xFFFF);
    }
}
control_section[cs_num].obj_line++;
}
return 0;
}

```

/ *

* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
* 여기서 출력되는 내용은 object code (프로젝트 1번) 이다.
* 매 계 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프로그램의 결과를 표준출력으로 보내어
* 화면에 출력해준다.

*
*

```

*/
void make_objectcode_output(char *file_name)
{
    /* add your code here */
    FILE* fp;
    if (file_name != NULL) { //인자로 NULL값이 들어오지 않는 경우 파일에 기록
        fp = fopen(file_name, "w+t");
    }
    else { //인자로 NULL값이 들어오는 경우 표준출력으로 화면에 출력

```

```

        fp = stdout;
    }
    int line_len = 0;
    int check_line = 0;
    int byte_num = 0;
    int start = 0;
    int line_max = 0;
    //object code 내용을 아래의 형식으로 출력
    for (int i = 0; i < cs_num; i++) {
        //Header record 작성
        fprintf(fp, "H%-6s%06X%06XWn", control_section[i].section, control_section[i].startAddr,
control_section[i].lastAddr);

        if (control_section[i].extdef != 0) { //Define record 작성
            fprintf(fp, "D");
            for (int j = 0; j < control_section[i].extdef; j++) {
                fprintf(fp, "%-6s%06X", control_section[i].def[j].symbol,
control_section[i].def[j].addr);
            }
            fprintf(fp, "Wn");
        }

        if (control_section[i].extref != 0) { //Refer record 작성
            fprintf(fp, "R");
            for (int j = 0; j < control_section[i].extref; j++) {
                fprintf(fp, "%-6s", control_section[i].ref[j].symbol);
            }
            fprintf(fp, "Wn");
        }

        //Text record 작성
        line_len = 0;
        check_line = 0;
        byte_num = 0;
        start = 0;
        line_max = 0;

        for (int j = 0; j < control_section[i].obj_line; j++) {
            if (strlen(control_section[i].obj_code[j]) == 0) {
                if (!start) {
                    check_line++;
                    continue;
                }
            }
        }
    }

```

면

```
else {
    start = 1;
}

if(strlen(control_section[i].obj_code[j]) == 0){
    break;
}
line_len += strlen(control_section[i].obj_code[j]);
if (line_len > 60) { //현재까지의 byte의 개수가 한줄에 쓸수 있는 분량보다 많다

    line_len -= strlen(control_section[i].obj_code[j]);
    line_max = 1;
}
if(line_max){
    fprintf(fp, "T%06X", control_section[i].startAddr + byte_num);
    byte_num += line_len / 2;
    fprintf(fp, "%02X", line_len / 2);
    for (int k = check_line; k < j; k++) {
        fprintf(fp, "%s", control_section[i].obj_code[k]);
        check_line++;
    }
    fprintf(fp, "\n");
    j--;
    line_len = 0;
    line_max = 0;
}
}

if (line_len) { // 남은 byte의 개수가 60보다 적을 때
    fprintf(fp, "T%06X", control_section[i].startAddr + byte_num);
    fprintf(fp, "%02X", line_len / 2);
    for (int k = check_line; k < control_section[i].obj_line; k++) {
        fprintf(fp, "%s", control_section[i].obj_code[k]);
        check_line++;
        if (strlen(control_section[i].obj_code[k]) == 0) {
            break;
        }
    }
    fprintf(fp, "\n");
    while(control_section[i].obj_line - check_line) { //아직 다 끝나지 않았을 때
        if (strlen(control_section[i].obj_code[check_line]) == 0) {
            check_line++;
            continue;
        }
        else {
```

```

        fprintf(fp, "T%06X", token_table[check_line]->address);
        line_len = strlen(control_section[i].obj_code[check_line]);
        check_line++;
        if (strlen(control_section[i].obj_code[check_line]) == 0) {
            fprintf(fp, "%02X%sWn", line_len/2,
control_section[i].obj_code[check_line - 1]);
        }
    }
}

if (control_section[i].extref != 0) { //Modification record 작성
    for (int j = 0; j < control_section[i].mod_num; j++) {
        fprintf(fp, "M%06X%02X%sWn", control_section[i].mod[j].addr,
control_section[i].ref_size[j], control_section[i].mod[j].symbol);
    }
}

if (i == 0) { //End record 작성
    fprintf(fp, "E%06XWnWn", control_section[i].startAddr);
}
else {
    fprintf(fp, "EWnWn");
}
}

if (file_name != NULL) { //파일을 열은 경우 닫아줌.
    fclose(fp);
}
}

```