

시스템프로그래밍 2021 보고서

보고서 제출서약서

나는 송실대학교 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	어셈블러 구현 (프로젝트#1)
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20162449 김상현 (출석번호 105번)
제출일	2021년 04월 11일

차 례

1장 프로젝트 동기/목적

2장 설계/구현 아이디어

3장 수행결과(구현 화면 포함)

4장 결론 및 보충할 점

5장 소스코드(+주석)

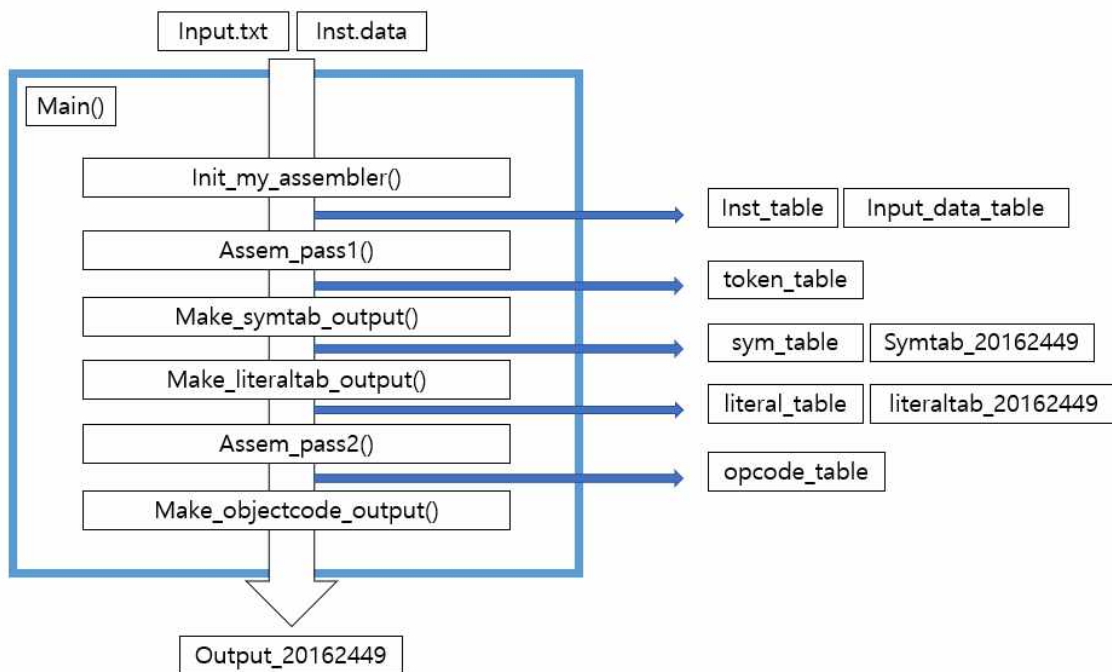
1장 프로젝트 동기/목적

SIC/XE 어셈블러를 개발하여 사용자가 주어진 input이 기계어로 바뀌는 과정을 이해하고 이를 object 파일로 만들어 볼 수 있다. 또 이 결과물을 통해 링커와 로더의 방식을 어렵듯이 이해할 수 있다.

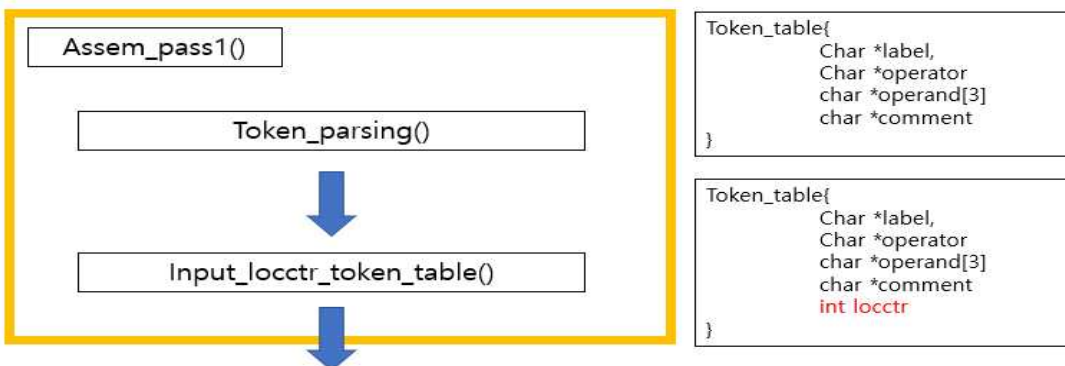
C언어를 이용함으로써 다른 언어에 비해 조금 복잡하지만 기본적인 뼈대와 흐름을 이해할 수 있다.

2장 설계/구현 아이디어

//아래의 모든 csec, csect, csec_num 은 모두 subroutine 번호를 가르키는 변수 이다. 구현 중 급하게 하는 바람에 변수 이름이 섞여서 가독성이 떨어지기에 본문 전에 언급한다.



이전의 과제에서 추가된 부분부터 설명 하겠다. 먼저 `assem_pass1()` 내부의 함수 구성이다.



`assem_pass1()`은 기존의 `token_parsing` 과 새로운 함수 `input_locctr_token_table()`로 구성되어 진다.

input_locctr_token_table()은 매계와 반환이 모두 없는 함수이다.

object코드를 만들기 위해 token_table에 loccation counter를 추가해 주는 함수이다. token_table을 모두 확인하며 전역 변수인 locctr을 조건에 맞추어 누적해서 더하고 이 값을 token_table의 locctr 변수로 각 토큰에 저장하였다. (어떤 식으로 locctr을 구성하였는지는 뒤에 구현방법에 설명하도록 하겠다.)

Make_symtab_output()

```
sym_table{
    Char *symbol
    int addr
    int csec
}
```

Make_symtab_output() 은 반환 값이 없고 생성할 파일 명을 매계로 받는 함수이다.

tokn_table을 뒤지며 만약 operator가 CSECT라면 0 으로 초기화 된 cset_num을 하나 씩 증가한다.
또한 LABEL이 NULL이 아닌 index를 찾고 그 table_token[index]의 Label과 locctr을 sym_table의 symbol ,
addr 에 각각 복사한다. 또 그때의 cset_num을 sym_table의 csec에 저장한다.
그리고 만들어진 sym_table을 매계로 주어진 파일명을 만들고 그 안에 저장한다.

Make_literaltab_output()

Check_literal_table()

```
literal_table{
    Char *literal
    int addr
    int csec_num
}
```

Make_literaltab_output() 은 반환 값이 없고 생성할 파일 명을 매계로 받는 함수이다.

먼저 Literal 은 쉽게 말에 '='으로 시작하는 operand 값이다. 리터럴이 나오게 되면 그 위치에서부터 밑에 LTORG라는 operator가 나오는지 찾고 있다면 그 위치에 '='으로 시작하는 operand를 선언해 준다.

ex) LDA =C'EOF'
 LTORG

----- 위 아래가 같다고 생각 할 수 있다.

 LDA EOF
EOF BYTE C'EOF'

* 만약 LTORG가 없다면 파일에 마지막에 선언하게 된다.

token_table을 확인하며 if (token_table[i]->operand[0][0] == '=')를 통해 Literal를 찾는다.

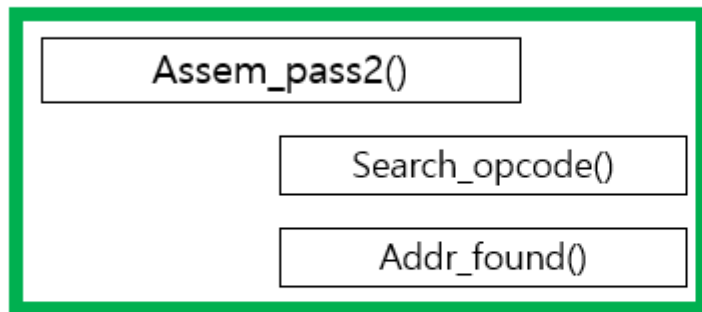
찾은 리터럴들을 lliteral_table에 저장하게 되는데 한가지 주의해야 할 부분이 생긴다.

input.txt 파일을 보면 =X'F1'이라는 리터럴이 2번 나온다. 그렇다면 literal_table에도 두 번 들어갈 수 있다.

이렇게 중복된 literal을 확인하기 위해서 check_literal_table() 함수를 제작하였다.

check_literal_table() :literal_table을 확인하여 새로 들어올 literal이 기존에 있는 값인지를 확인 하는 함수

완성된 literal_table을 주어진 파일명을 새로 만들어나 오픈하여 파일에 저장한다.



```

Token_table{
    Char *label,
    Char *operator
    char *operand[3]
    char *comment
    char nixbpe
}
  
```

```

opcode_table{
    int csec_num
    int addr
    int opcode
    int format
}
  
```

Assem_pass2() 은 어셈블리 코드를 기계어 코드로 바꾸어 주는 함수이다.

● opcode_table의 이름에 오류가 있다. objectcode로 만들어 저장하는 테이블인데 opcode라고 구현하였다.

먼저 token_table의 nixbpe라는 비트 연산을 위한 char 변수를 추가하였다. token_table을 확인하면서 ‘#’(immediate), ‘@’(indirect) 인지 operand[1]값이 X가 나오는지(루프문인지) , 4형식인지 조건을 확인하며 nixbpe를 각각 위치에 넣는다.

- 4형식이 아닌 operator에 대해서는 모두 pc relatives 라고 가정한다.

ex) FIRST STL RETADR

#:0 @:0 X:0 B:0 P:1 E :0

=> n : 1, l : 1, x : 0, b : 0, p : 1, e :0

- 4형식일 때는 (operator에 ‘+’가 들어감을 확인함으로써 알 수 있다) e : 1 로 바꾸어 준다.

ex) CLOOP +JSUB RDREC

#:0 @:0 X:0 B:0 P:0 E :1

=> n : 1, l : 1, x : 0, b : 0, p : 0, e :1

그리고 temp라는 int형 변수를 선언한다 (objectcode는 총 24비트가 필요하고 4형식의 경우는 32비트가 필요하기 때문에 int형을 사용하였다.) 이 temp 에 search_opcode()를 통해 해당 operator의 inst_table의 index를 찾고 inst_table[index]->op를 temp에 더해준다 그리고 그 값을 왼쪽으로 4번 시프트 해준다.

마지막으로 temp에 nixbpe 값을 더해주면 앞 3자리가 완성된다.

ex) STL : 14(16)

temp = 0001 0100 , temp = temp << 4

=> temp = 0001 0100 0000

temp += nixbpe

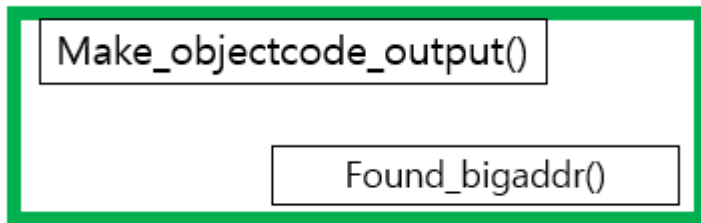
temp = 0001 0100 0000

+ nixbpe = 0000 0011 0010

= temp = 0001 0111 0010 =>172(16)

뒤에 주소부분을 구현 한 방식에는 뒤에서 기술하겠다.

완성된 opcode_table 예시 ----->
(opcode.opcode opcode.addr, opcode.csec_num)



```
172027 0 0
4b100000 3 0
032023 7 0
290000 a 0
332007 d 0
4b100000 10 0
3f2fec 14 0
032016 17 0
0f2016 1a 0
010003 1d 0
0f200a 20 0
4b100000 23 0
3e2000 27 0
454f46 30 0
```

Make_objectcode_output() 은 opcode_table을 가지고 object file을 출력하고
파일을 저장하는 함수이다.

● 구현을 완성하지 못했고 구현을 한부분에 대해서만 설명하려고 한다.

object file에서 맨앞에는 H 라는 헤더가 들어간다. header에는 symbol_table에서 csec_num 별로 제일 앞에있
는 symbol 일 것이다. (COPY, RDREC, WDREC) 그 이후는 해당 symbol의 주소값을 넣는다.

(COPY : 000000 RDREC : 000000 WDREC : 000000)

그 뒤는 총 크기 가 나오게 되는데 이는 token_table에서 같은 csec_num 중에 제일 큰 addr를 찾으면 되는데
found_bigaddr() 함수를 제작하여 사용하였다.

found_bigaddr() : token_table에서 입력받은 input 과 같은 csect를 가지는 token 중에 제일 큰 addr를
가진 값을 저장하고 그 addr를 리턴한다.

ex) H COPY000000001033

다음은 'D'가 나온다. 같은 subrutne 번호를 가지는 token_table에서 operator 가 EXTDEF 일 때 operand를
모두 출력해주고 그뒤는 위에서 언급한 **addr_found()**함수를 사용하여 해당 operand를 가지고 symbol_table나
literal_table 에 존재하는지 확인하고 그 때의 주소를 리턴하여 적었다.

ex) D BUFFER000033 BUFEND001033 LENGTH00002d

'R'은 D때와 마찬가지로 같은 subrutne 번호를 가지는 token_table에서 operator 가 EXTREF 일 때 operand
값을 모두 출력해 주었다.

ex) R RDREC WRREC

본문 차례이다. 'T'로 시작하게 되고 시작주소, 총 길이, object cod 순으로 출력되어야한다. 임시
버퍼(temp_char[])를 만들고 opcode_table.opcode(object 코드들)을 16진수 값을 그대로 문자열로 바꾸어
버퍼에 넣는다. (여기서 버퍼에 넣는 이유는 총 길이를 세서 최대 1E만큼만 한 줄에 들어갈 수 있다는
조건을 만들기 위해서이다.) 2형식일때는 %.4x 3형식은 %.6x 4형식은 %.8x로 문자열로 변환되어 들어간다.

이제 임시 버퍼의 총 길이를 세고 1E가 넘지 않도록 하고 넘는다면 다시 루프로 T부터 다시 넣는다.

T000000 1d 1720274b1000000320232900003320074b1000003f2fec0320160f2016
T00001d 10 0100030f200a4b1000003e2000454f46

여기까지가 구현에 성공한 내용이다.

<과제 수행 중 사용한 구현 방법>

input_locctr_token_table()

1. input_locctr_token_table()에서 locctr을 넣은 방법

조건문으로 operator이 format에 따라 2형식과 3형식과 4형식을 각각 구분하고 locctr을 각각 계속 누적해서 더해간다.

여기서 추가된 예외처리는 token_table의 operator가 RESW, RESB, CSECT, LTORG, BYTE일 때 이다.

1. RESW 일 때

- 기존의 locctr에 operand[0]을 정수화 한 값을 더해줘야한다. 그런데 word는 3byte이기 때문에 locctr에 3*atoi(token_table[i]->operand[0])을 누적해서 더한다.

2. RESB 일 때

- BYTE는 그대로 정수화 한값을 추가해 주면 될 거라고 생각했다.
locctr에 atoi(token_table[i]->operand[0])를 누적해서 더한다.

3. CSECT

- locctr을 초기화하여 다시 0부터 저장될 수 있게한다.

4. LTORG

- locctr += (strlen(token_table[Search_LTORG(i)]->operand[0]) - 4) 이렇게 구현 하였는데 **search_LTORG(i)**는 LTORG가 나왔을 때 그 위의 '='으로 시작하는 operand를 찾고 그 index를 리턴하는 함수이다.
리턴값을 token_table의 index로 넣고 나온 값은 ex) =C'EOF' 일 것이다. 이때 = C ' ' 이 네가지를 빼서 나온 길이를 locctr에 누적하여 더해준다.

5. BYTE

- 뒤에나오는 operand 값만큼 더해주면 된다.

6. BUFFER-BUFFEND ?

input에서 주어지는 LENGTH는 위의 값이 operand로 들어있다. 나머지 locctr은 모두 상대 주소인데 반해 이 값은 절대 주소이다. BUFFER의 주소와 BUFFEND의 주소의 차이를 저장하게 되는데

먼저 " BUFFER - BUFFEND " 라는 문자열을 '-' 문자를 기준으로 파싱하고 각각의 Buffer, Buffend를 **Search_Label()** 이라는 함수의 인자로 넣어 그 label을 가지는 token_table의 index를 각각 저장한다.

그리고 token_table[index1], token_table[index2]의 locctr의 차이를 구하여 locctr에 저장한다.

- Search_Label()은 문자열을 매개로 하여 해당하는 문자열이 label과 같은 token_table의 index를 반환한다.

2. `assem_pass2()`에서 주소부분 구현한 방법

이전의 `assem_pass2()` 과정을 통해 opcode 와 nixbpe가 결합된 temp 버퍼를 만들고 이를 왼쪽으로 각 형식 별로 8 , 12 , 20 만큼 시프트 하였다. (ex. 172000 (16))

일단 형식을 기준으로 조건을 잡을 수 있다.

i) 2형식 일 때

2형식 뒤에는 레지스터 값이 나오게 되는데 각각의 레지스터는 지칭하는 주소가 있다.

해당 token_table에서 `operand[0]` 값이 레지스터(X, S, A ..) 일 때

`temp += 16 * (레지스터의 주소)`

주소에 16을 곱하는 이유는 2번째 자리에 위치시키기 위함이다.

ex) B400 (`temp`) + 10 (16 * 1(X 레지스터 주소)) = B410

`operand[1]`값이 존재한다면 `temp += (레지스터의 주소)`

ex) A000 (`temp`) + 01 (X 레지스터 주소) = A004

ii) 3형식 일 때

SIC/XE 머신은 PC relative를 기준으로 주소를 찾는다. 주소값을 찾는 가장 간단한 공식은 `Target address - program counter` 가 된다.

여기서 Target 주소는 `operand`값을 label로 가지는 token_table의 locctr 이 되는 것이고

program counter은 다음 명령어의 locctr 이 될 것이다.

아래는 구현 방식이다.

`addr_found(token_table[i]->operand[0], csec_num)` : Target address

`token_table[i + 1]->locctr` : program counter

ex) FIRST STL RETADR

RETADR의 주소 : 2A

현재의 pc 값 : 03

$2A - 03 = 27(16)$

$172000 + 027 = 172027$

여기서 이 차가 양 수 일 때는 정확한 값이 나오지만 음수가 되면 전혀 다른 값이 나올 수 있다.

그래서 조건문을 사용하여 음수일 때를 따로 지정한다.

`4096 - token_table[i + 1]->locctr + addr_found(token_table[i]->operand[0], csec_num)`

4096은 16진수 FFF+1 값이다. 보수계산을 위해 큰 값을 FFF+1에서 빼고 나온 결과에

작은 값을 넣게 되면 보수의 계산을 할 수 있다.

위 결과를 temp 에 더해주면 우리가 구하는 object code를 만들 수 있다.

ex) J CLOOP

CLOOP의 주소 : $03(16) = 3(10)$

현재의 pc 값 : $17(16) = 23(10)$

$3 - 17 < 0$, $(1111\ 1111\ 1111 + 1) - 0001\ 0111 + 0011 = 1111\ 1110\ 1100 \Rightarrow FEC(16)$

`temp = 3F2000`

$3F2000 + FEC = 3F2FEC$

iii) 4형식일 때

operand[0]을 라벨로 가지는 token_table 의 locctr을 그대로 넣으면 된다.

CLOOP + JSUB RDREC

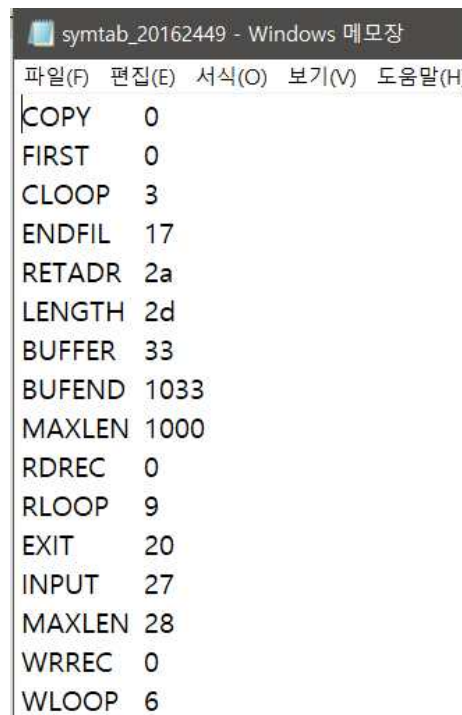
RDREC의 주소 000000

temp = 4b100000

3장 수행결과(구현 화면 포함)

```
COPY      0      0
FIRST     0      0
CLOOP     3      0
ENDFIL    17     0
RETADR    2a     0
LENGTH   2d     0
BUFFER    33     0
BUFEND    1033   0
MAXLEN    1000   0
RDREC     0      1
RLOOP     9      1
EXIT      20     1
INPUT     27     1
MAXLEN    28     1
WRREC     0      2
WLOOP     6      2

fin symtable
```



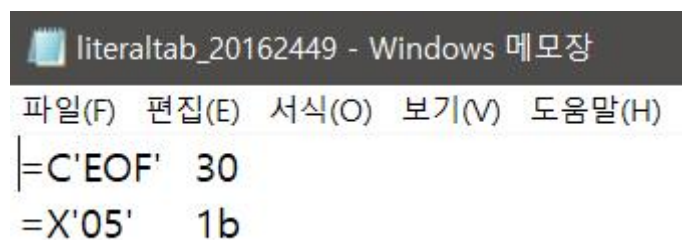
symtab_20162449 - Windows 메모장

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
```

```
=C'EOF'  48
=X'05'   27

fin literalb
```



literalb_20162449 - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
=C'EOF'  30
=X'05'   1b
```

```
output_20162449 - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
HCOPY000000001033
DBUFFER000033BUFEND001033LENGTH00002d
RRDRECWRREC
T000000 1d 1720274b1000000320232900003320074b1000003f2fec0320160f2016
T00001d 10 0100030f200a4b1000003e2000454f46

HRDREC000000000028
RBUFFERLENGTHBUFEND
T000000 1d b410b400b44077201fe3201b332ffadb2015a00433200957900000b850
T00001d 0e 3b2fe9131000004f0000f1000000

HWRREC000000000001b
RLENGTHBUFFER
T000000 1c b41077100000e32012332ffa53900000df2008b8503b2fee4f000005
```

미완성 output

4장 결론 및 보충할 점

literal등 과 같은 특수한 경우가 주어졌을 때 대부분을 예외처리를 통해 해결하였기 때문에 현재 주어진 input.txt의 경우에는 문제 없이 출력되지만 input.txt 와 조금 다른 형식으로 주어지는 소스코드의 경우 나의 코드는 출력되지 않을 수 있다. 여기서 찾은 나의 코드의 문제점은 아래와 같다.

1. literal의 표현?

literal을 마지막 object file을 만들 때 조금 더 쉽게 표현하고자 LTORG 자리에 BYTE =C'EOF'로 치환 하도록 하였고 LTORG 가 없는 literal의 경우에는 마지막에 추가하기 위해 END 위치에 BYTE = X'F1' 로 치환하였다.

```
locctr : 30          LTORG          기존의 token_Table
```

```
locctr : 30          BYTE    =C'EOF' objectcodefmt 만들기 위해 치환한 token_table
```

```
HCOPY000000001033
DBUFFER000033BUFEND001033LENGTH00002d
RRDRECWRREC
T000000 1d 1720274b1000000320232900003320074b1000003f2fec0320160f2016
T00001d 10 0100030f200a4b1000003e2000454f46
```

치환한 table을 가지고 object code를 만들었더니 3e2000 뒤에 바로 BYTE 에 해당하는 objectcode가 붙게 되었다. 이 오류를 해결하지 못했다.

2. 예외처리

```
if (token_table[i]->operand[0][1] == 'X') { // =X'F1'에 대한 예외처리
```

input.txt 에는 =X'F1' 도 존재하고 X'F1' 값도 존재한다 분명 앞에 F1은 Literal 이고 뒤의 F1은 INPUT이라는 label로 지정된 Byte이다. 하지만 이 두 개의 차이를 조건문으로 구현하다보니 계속 오류가 발생하였고 결국 operand[0][1] 이 'X' 이냐 , '=' 이냐 로 구분하여 예외처리 하였다.

이는 다른 input이 들어왔을 때 심각한 오류를 초래할 것이다.

솔직하게 말한다면 시간이 많이 부족했던거 같다 중간고사 기간이랑 겹쳐 완성하지 못하였는데 고지를 눈앞에 두고 내려놓은 것 같아 많이 후회된다. 코딩 실력이 부족하여 결과는 비슷하게 나오지만 분명 그 속의 내용은 썩어 있지 않을 까 싶다. 그래도 조금더 체계적으로 구현을 하는 부분에 대한 실마리를 얻은 것 같다.

5장 소스코드(+주석)

```
/*
 * 화일명 : my_assembler_20162449.c
 * 설 명 : 이 프로그램은 SIC/XE 머신을 위한
 간단한 Assembler 프로그램의 메인루틴으로,
 * 입력된 파일의 코드 중, 명령어에 해당하는
 OPCODE를 찾아 출력한다.
 */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h >
#include <stdlib.h >
#include <string.h >
#include <fcntl.h >
#include "my_assembler_20162449.h"
/
-----
-----
* 설명 : 사용자로부터 어셈블리 파일을 받아서
명령어의 OPCODE를 찾아 출력한다.
* 매 계 : 실행 파일, 어셈블리 파일
* 반환 : 성공 = 0, 실패 = < 0
* 주의 : 현재 어셈블리 프로그램의 리스트 파일
을 생성하는 루틴은 만들지 않았다.
*              또한 중간파일을 생성하지
않는다.
*
-----
-----
*/
int main(int args, char *arg[])
{
    if (init_my_assembler() <0)
    {
        printf("init_my_assembler: 프로
그램 초기화에 실패 했습니다.Wn");
        return -1;
    }
    if (assem_pass1() <0)
    {
        printf("assem_pass1: 패스1 과
정에서 실패하였습니다. Wn");
        return -1;
    }
    make_symtab_output("symtab_20162449");
    make_literaltab_output("literaltab_20162449");
    if (assem_pass2() <0)
    {
        printf(" assem_pass2: 패스2 과
정에서 실패하였습니다. Wn");
        return -1;
    }
}
```

```
make_objectcode_output("output_20162449");
    return 0;
}
/
-----
-----
* 설명 : 프로그램 초기화를 위한 자료구조 생성
및 파일을 읽는 함수이다.
* 매 계 : 없음
* 반환 : 정상종료 = 0 , 에러 발생 = -1
* 주의 : 각각의 명령어 테이블을 내부에 선언하
지 않고 관리를 용이하게 하기
*              위해서 파일 단위로 관리하
여 프로그램 초기화를 통해 정보를 읽어 올 수
있도록
*              구현하였다.
*
-----
-----
*/
int init_my_assembler(void)
{
    printf("*****init_my_assembler*
*****WnWn");
    int result;
    if ((result = init_inst_file("inst.data"))
<0)
        return -1;
    if ((result = init_input_file("input.txt"))
<0)
        return -1;

    printf("Wn*****
*****Wn");
    return result;
}
/
-----
-----
* 설명 : 머신을 위한 기계 코드목록 파일을 읽
어 기계어 목록 테이블(inst_table)을
*              생성하는 함수이다.
* 매 계 : 기계어 목록 파일
* 반환 : 정상종료 = 0 , 에러 < 0
* 주의 : 기계어 목록파일 형식은 자유롭게 구현
한다. 예시는 다음과 같다.
*
*
=====
=====
=====
*              | 이름 | 형식 | 기계어 코
```

드 | 오퍼랜드의 갯수 | NULL |

```

*
=====
=====
=====
*
-----
-----
-----
*/
int init_inst_file(char *inst_file)
{
    printf("-----init_inst_file-----\n");
    FILE *file;
    int err = 0;
    int i = 0;
    int num;
    char buffer[100];
    char *ptr;
    if((file = fopen(inst_file,"r"))<0){
        return -1;
    }
    while(feof(file)==0){
        i n s t _ t a b l e [ i ]
        =malloc(sizeof(inst));
        fgets(buffer,sizeof(buffer),file);
        check_Enter(buffer);
        ptr = strtok(buffer,",");
        //strtok buffer with ","
        strcpy(inst_table[i]->str,ptr);
        ptr= strtok(NULL,",");
        if(strchr(ptr,'/')!=NULL){
            ptr[strlen(ptr)-2]='W0';
            inst_table[i]->format
            =atoi(ptr); // save first letter in inst_table
            format , ex) 3/4 -> 3
        }
        else{
            inst_table[i]->format
            =atoi(ptr);
        }
        ptr=strtok(NULL,",");
        num = StringToHexNum(ptr);
        //StringToHexNum
        if(num <0){
            return -1;
        }
        inst_table[i]->op =num;
        ptr= strtok(NULL,",");
        inst_table[i]->ops = atoi(ptr);
        printf("%d
        % s / % d / % x / % d \n " , i
        +1,inst_table[i]->str,inst_table[i]->format,inst_t
        able[i]->op,inst_table[i]->ops);
        i++;
    }
}
```

```

}
    inst_index = i;
    fclose(file);
    /* add your code here */
    printf("sucess init_inst_file\n");

    printf("-----\n");
    return err;
}
/*-----
-----
-----
* 설명 : 문자열이지만 16진수의 의미를 담고
있기 때문에 이를 저장하기 위해서는 문자열을
정수형으로 바꿔줘야한다.
ex ) 1C -> 1*16 + 12
* 매개 : 문자열
* 반환 : 없음
-----
-----
-----
-----*/
int StringToHexNum(char *input){
    int i;
    unsigned char num =0;
    for(i =0;i <2;i ++){
        if(input[i]>='0'&&input[i]<='9')
            num = num *16
            +input[i]-'0'; // because num is hex num,
            *16 .
        e l s e
        if(input[i]>='A'&&input[i]<='Z')
            num = num *16
            +input[i]-'A'+10;
        else
            return -1;
    }
    return num;
}
/
-----
-----
* 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)
{
}
```

```

printf("-----init_input_file-----\n");
FILE *file;
int err = 0;
int i = 0;
char buffer[100]; // for one line
buffer
if((file = fopen(input_file,"r"))<0){
    return -1;
}
while(feof(file)==0){
    input_data[i]
=malloc(sizeof(char)*100);
    fgets(buffer,sizeof(buffer),file);
    strcpy(input_data[i], buffer);
    // printf("%s",input_data[i]);
    i++;
}
/* add your code here */
line_num = i - 1;
printf("sucess init_input_file\n");

printf("-----\n");
return err;
}
/
-----
* 설명 : 어셈블리 코드를 위한 패스1과정을 수행
하는 함수이다.
*
* 패스1에서는..
*
* 1. 프로그램 소스를 스캔하여
해당하는 토큰단위로 분리하여 프로그램 라인별
토큰
*
* 테이블을 생성한다.
*
*
* 매계 : 없음
* 반환 : 정상 종료 = 0 , 에러 = < 0
* 주의 : 현재 초기 버전에서는 에러에 대한 검사
를 하지 않고 넘어간 상태이다.
*
* 따라서 에러에 대한 검사 루틴을 추가
해야 한다.
*
*
-----
*/
static int assem_pass1(void)
{
printf("\n*****assem_pass1*****\n");
int i = 0;
/* add your code here */

```

```

printf("-----\n");
token_parsing
while(i < line_num){
if(token_parsing(input_data[i])<0)
    return -1;
    i++;
}
printf("token_parsing sucess\n");
line_num = token_line;

printf("-----\n");
input_locctr_token_table("intermediate");

printf("*****\n");
return 0;
/* input_data의 문자열을 한줄씩 입력
받아서
* token_parsing()을 호출하여
token_unit에 저장
*/
}
/
-----
* 설명 : 소스 코드를 읽어와 토큰단위로 분석하
고 토큰 테이블을 작성하는 함수이다.
*
* 패스 1로 부터 호출된다.
* 매계 : 파싱을 원하는 문자열
* 반환 : 정상종료 = 0 , 에러 < 0
* 주의 : my_assembler 프로그램에서는 라인단
위로 토큰 및 오브젝트 관리를 하고 있다.
*
-----
*/
int token_parsing(char * str)
{
/ / m e m o r y
init-----
token_table[token_line]
=malloc(sizeof(token));
token_table[token_line]->label
=malloc(sizeof(char) *20);
token_table[token_line]->operator
=malloc(sizeof(char) *20);

//-----
char * label =NULL, *operator =NULL,
* operand =NULL, * comment =NULL;
char * operand1 =NULL, * operand2

```

```

=NULL, * operand3 =NULL;
    char ** fields[] = { &label, &operator,
&operand, &comment, NULL };
    char ** operand_fields[] = {
&operand1, &operand2, &operand3 };
    // 주석 필터링
    if (str[0] =='.')
        return 0;
    // Tab을 구분자로 하여 각 필드 분리
    for (int i =0; fields[i] !=NULL; i ++) {
        *fields[i] = str;
        if ((str = strpbrk(str, "WtWn"))
==NULL)
            break;
        *str ++='W0'; // 중간에 NULL
문자를 넣어 문자열 분리
    }
    for (int i =0; operand_fields[i] !=NULL;
i ++) {
        *operand_fields[i] = operand;
        if ((operand =
strpbrk(operand, ",") ==NULL)
            break;
        *operand ++='W0'; // 중간에
NULL 문자를 넣어 문자열 분리
    }
    strcpy(token_table[token_line]->label,
label);

    strcpy(token_table[token_line]->operator
,operator);
    if (operand1 !='W0')

    strcpy(token_table[token_line]->operand[0],
operand1);
    else

    strcpy(token_table[token_line]->operand[0],
"");
    if (operand2 !='W0')

    strcpy(token_table[token_line]->operand[1],
operand2);
    else

    strcpy(token_table[token_line]->operand[1],
"");
    if (operand3 !='W0')

    strcpy(token_table[token_line]->operand[2],
operand3);
    else

    strcpy(token_table[token_line]->operand[2],
"");
    if (comment !='W0')

    strcpy(token_table[token_line]->comment,

```

```

comment);
    else

    strcpy(token_table[token_line]->comment, "");
    token_line++;
    return 0;
}
/
*
* 설명 : 생성된 token_table에 locctr를 추가하는
함수
* 매개 : 생성할 오브젝트 파일명
* 반환 : 없음
*
*
*/
void input_locctr_token_table(void)
{
    printf("-----input_locctr_token_tabl
e-----Wn");
    search_ltorg_index =0;
    int temp_index1,temp_index2;
    int i =0;
    int index;
    csec_num =0;
    locctr=0;
    while (i < line_num) { // 시작주소를 찾
기위해 START 뒤의 operand 값을 저장한다.
        i
        if
        (strcmp(token_table[i]->operator,"START")
==0) {
            locctr
            =
            atoi(token_table[i]->operand[0]);
            break;
        }
        i++;
    }
    strloc = locctr;
    i =0;
    while (i < line_num) {
        i
        f
        (strcmp(token_table[i]->operator,"CSECT")
==0)
            csec_num++;
            token_table[i]->csect
            =
            csec_num;
            token_table[i]->locctr
            =
            locctr;
            index
            =
            search_opcode(token_table[i]->operator);
            if (index <0) {
                i
                f
                (strcmp(token_table[i]->operator,"RESW") ==0)

```

```

{
    locctr += (3 *
atoi(token_table[i]->operand[0])); //word 는
1개당 3바이트!
    }
    else if
(strcmp(token_table[i]->operator,"RESB") ==0)
{
    locctr +=
atoi(token_table[i]->operand[0]);
    }
    else if
(strcmp(token_table[i]->operator,"CSECT")
==0) { // 새로운 서브루틴 의미하기 때문에 0오
로 locctr 초기화

token_table[i]->locctr =0;

    locctr =0;
    }
    else if
(strcmp(token_table[i]->operator,"LTORG")
==0) { //LRORG 를 찾으면

    locctr +=
(strlen(token_table[Search_LTORG(i)]->operand
[0]) -4); // Search_LTORG 함수로 이동하여
search_ltorg_index 부터 현재 index 까지에 '='이
들어간 operand를 찾는다.

search_ltorg_index = i;
    }
    else if
(strcmp(token_table[i]->operator,"BYTE") ==0)
{
    i f
(token_table[i]->operand[0][0] =='X')
    locctr
+= (strlen(token_table[i]->operand[0]) -3) /2;

//if(token_table[i]->operand[0][0] == '"') // 이
곳에 'X' 말고 다른 조건을 넣으면 된다.
    }
    else if
(strchr(token_table[i]->operand[0], '-')
!=NULL &&
strcmp(token_table[i]->operator,"EQU")==0) {
// BUFFER -BUFEND 같은 경우에대한 예외 처
리

    char * ptr =
strtok(token_table[i]->operand[0], "-");
    temp_index1
= Search_Label(ptr);
    ptr
=
strtok(NULL, "-");
    temp_index2
= Search_Label(ptr);

token_table[i]->locctr
token_table[temp_index1]->locctr
=
-

```

```

token_table[temp_index2]->locctr;
    }
    else {
        locctr +=0;
    }
    printf("locctr : %x
W t % s W t % s W t % s W t % s W t % d W n ",
token_table[i]->locctr, token_table[i]->label,
token_table[i]->operator,token_table[i]->opera
n d [ 0 ],
token_table[i]->operand[1],token_table[i]->cs
ect);

    i++;
    continue;
}
if (token_table[i]->operator[0]
=='+'){
    locctr +=4;
}
else
    locctr +=

inst_table[index]->format;
    printf("locctr : %x
W t % s W t % s W t % s W t % s W t % d W n ",
token_table[i]->locctr, token_table[i]->label,
token_table[i]->operator,token_table[i]->opera
nd[0], token_table[i]->operand[1],
token_table[i]->csect);

    i++;
}
printf("sucessWn");

printf("-----
-----Wn");
}
/*-----
-----
-----
*설명 : (추가) LTORG 가 나왔을 때 그 위의
'='으로 시작하는 operand를 찾는다
*매개 : void
*반환 : 그 token_table의 index
-----
-----
-----*/
int Search_LTORG(int index) {
    for (int i = search_ltorg_index; i <
index; i ++) {
        i f
(token_table[i]->operand[0][0] =='=') //
=C'EOF' 라고 할때 operand[0][0]= '=' 이 되고
이때의 index를 반환.

        return i;
    }
}
/*-----

```



```

-----
* 설명 : 주어진 label이 존재하는 token_table의
inst_table의 op를 비교하여 존재하면 그 op를 반
환,
        이때 +jsub 같은 경우가 존재할
수 있으므로 문자열 첫번째 문자를 없애고 다시
같은 과정 반복.
* 매개 : 문자열
* 반환 : 정상종료 = 기계어 테이블 인덱스, 에러
< 0
-----
-----*/
int Search_Label(char * label) {
    int i;
    for (i = 0; i < line_num; i++) {
        i                                     f
        (strcmp(token_table[i]->label, label) == 0) {
            return i;
        }
    }
    return -1;
}
/*-----
-----
* 설명 : 주어진 input 문자열과 inst_table의 op
를 비교하여 존재하면 그 op를 반환,
        이때 +jsub 같은 경우가 존재할
수 있으므로 문자열 첫번째 문자를 없애고 다시
같은 과정 반복.
* 매개 : 문자열
* 반환 : 정상종료 = 기계어 테이블 인덱스, 에러
< 0
-----
-----*/
int search_opcode(char *input){
    int i,j;
    char * tmp_input
=malloc(sizeof(input));
    strcpy(tmp_input, input);
    for(i = 0 ; i < inst_index;i++){

if(strcmp(inst_table[i]->str,input)==0)
    return i;
    }
    for(j = 1;tmp_input[j];j++){
        tmp_input[j-1]=tmp_input[j];
    }//첫 문자 삭제
    tmp_input[j-1]='W0';
    for(i = 0;i < inst_index;i++){

if(strcmp(inst_table[i]->str,tmp_input)==0)
    return i;
    }
    return -1;
}
/*-----
-----
* 설명 : fgets() 함수로 받은 각 라인들에 마지막

```

```

에 개행문자가 들어가면서
        각 테이블로 파싱되는 과정에서
개행문자들이 섞여 들어가
        원하는 output.txt를 출력하지 못
하길래 개행문자를 없애는 함수를 만들었습니다.
* 매개 : 문자열
* 반환 : 없음
-----
-----*/
void check_Enter(char *input){
    int i = 0;
    for(i = 0;input[i] != 0;i++){
        if(input[i] == '\n'){
            input[i] = 0;
            break;
        }
    }
}
/*-----
-----
* 설명 : 입력된 문자열의 이름을 가진 파일에 프
로그램의 결과를 저장하는 함수이다.
* 매개 : 여기서 출력되는 내용은 SYMBOL별 주
소값이 저장된 TABLE이다.
* 매개 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프
로그램의 결과를 표준출력으로 보내어
* 화면에 출력해준다.
*
*
-----
-----
*/
void make_symtab_output(char * file_name)
{
    symbol_line = 0;
    FILE* fp;
    fp = fopen(file_name, "w");
    int i = 0;
    int cset_num = 0;
    while (i < line_num) {
        i                                     f
        (strcmp(token_table[i]->operator,"CSECT")
==0)
            cset_num++;
            if (token_table[i]->label[0]
!= 'W0') {
                sym_table[symbol_line].symbol
=malloc(sizeof(symbol)*20);

                strcpy(sym_table[symbol_line].symbol,
token_table[i]->label);
                printf("%sWt",
sym_table[symbol_line].symbol);
            }
        }
    }
}

```

```

sym_table[symbal_line].addr =
token_table[i] -> locctr;
printf("%xWt", sym_table[symbal_line].addr);

sym_table[symbal_line].csec = cset_num;
printf("%xWn", sym_table[symbal_line].csec);
fprintf(fp, "%sWt",
sym_table[symbal_line].symbol);
fprintf(fp, "%xWn",
sym_table[symbal_line].addr);
symbal_line++;
}
i++;
}
printf("Wnfin symtableWn");
fclose(fp);
}
/
-----
-----
* 설명 : 입력된 문자열의 이름을 가진 파일에 프
로그래밍의 결과를 저장하는 함수이다.
* 여기서 출력되는 내용은 LITERAL별 주
소값이 저장된 TABLE이다.
* 매개 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프
로그래밍의 결과를 표준출력으로 보내어
화면에 출력해준다.
*
*
-----
-----
*/
void make_literal_output(char * file_name)
{
    FILE* fp;
    fp = fopen(file_name, "w");
    char temp[100];
    int i = 0;
    int flag = 0;
    literal_line = 0;
    while (i < line_num) {

literal_table[i].literal = malloc(sizeof(char)*20);
i
(token_table[i] -> operand[0][0] == '=') {
for (int j = i; j <
line_num; j++) { // literal 의 index 부터 ~ 끝까
지 LTORG가 operator로 나오는지 검사한다.
i
f
(strcmp(token_table[j] -> operator, "LTORG")
== 0) {
i
f
(check_literal_table(token_table[i] -> operand[0])

```

```

<0) { //literal_table 중복검사

strcpy(literal_table[literal_line].literal,
token_table[i] -> operand[0]);

p r i n t f ( " % s W t " ,
literal_table[literal_line].literal);

literal_table[literal_line].addr =
token_table[j] -> locctr;

p r i n t f ( " % d W n " ,
literal_table[literal_line].addr);

fprintf(fp, "%sWt%xWn",
literal_table[literal_line].literal,
literal_table[literal_line].addr);

strcpy(token_table[j] -> operator, "BYTE");

strcpy(token_table[j] -> operand[0],
literal_table[literal_line].literal);

literal_line++;

flag = 1;

break;

}
}
if (flag == 0) { // '='이
포함된 operand 밑에 LTORG가 없다는 뜻!
i
f
(check_literal_table(token_table[i] -> operand[0])
<0) { //literal_table 중복검사

strcpy(literal_table[literal_line].literal,
token_table[i] -> operand[0]);

printf("%sWt", literal_table[literal_line].literal);

literal_table[literal_line].addr = locctr;

printf("%dWn", literal_table[literal_line].addr);

locctr
+= (strlen(token_table[i] -> operand[0]) - 3) / 2;

fprintf(fp, "%sWt%xWn",
literal_table[literal_line].literal,
literal_table[literal_line].addr);

strcpy(token_table[line_num-1] -> operator, "BYT
E");

strcpy(token_table[line_num - 1] -> operand[0],
literal_table[literal_line].literal);

```

```

literal_line++;
    }
}
flag =0;
i++;
}
printf("Wnfin literal tabWn");
fclose(fp);
/* add your code here */
}
/
-----
-----
* 설명 : literal table에 중복된 값이 들어가지
않도록 확인하는 함수이다.
* 매개 : 문자열 (operand[0])
* 반환 : 이미 존재하면 1 없다면 -1
*
-----
-----
*/
int check_literal_table(char *str) {
    int i;
    for (i =0; i < literal_line; i ++) {
        if (strcmp(literal_table[i].literal,
str) ==0) {
            return 1;
        }
    }
    return -1;
}
/
-----
-----
* 설명 : 어셈블리 코드를 기계어 코드로 바꾸기
위한 패스2 과정을 수행하는 함수이다.
* 패스 2에서는 프로그램을 기
계어로 바꾸는 작업은 라인 단위로 수행된다.
* 다음과 같은 작업이 수행되
어 진다.
1. 실제로 해당 어셈블
리 명령어를 기계어로 바꾸는 작업을 수행한다.
2. 어셈블리 명령어들을
opcode_table 저장한다.
* 매개 : 없음
* 반환 : 정상종료 = 0, 에러발생 = < 0
* 주의 :
*
-----
-----
*/
static int assem_pass2(void)

```

```

{
    int i =0;

    while (i < line_num) {
        printf("locctr : %x
W t % s W t % s W t % s W t % s W n " ,
token_table[i]->locctr, token_table[i]->label,
token_table[i]->operator, token_table[i]->opera
nd[0], token_table[i]->operand[1]);
        i++;
    }
    i =0;
    int j;
    opcode_table_line_num =0;
    csec_num =0;

    printf("Wn*****assem_pass2*****
*****Wn");
    int temp =0;
    int index =0;
    int temp_index;
    int temp_hex =0;
    char temp_char[100];
    csec_num =0;
    while (i < line_num) {
        i
        (strcmp(token_table[i]->operator, "CSECT")
==0)
        csec_num ++;
        token_table[i]->nixbpe =0;
        index =
search_opcode(token_table[i]->operator);
        if (index >=0) { // operator 가
inst_table 에 존재할때
            i
            (inst_table[index]->format ==3) {
                i
                (token_table[i]->operand[0][0]
=='@')
                { //immediate
                    token_table[i]->nixbpe +=1;

                    token_table[i]->nixbpe =
                    token_table[i]->nixbpe <<5; // n:1 i:0 x:0 p:0
                    e:0

                    token_table[i]->nixbpe +=2; // n:1 i:0 x:0 p:1
                    e:0 indirect 는 pc relative 사용

                    temp
                    = inst_table[index]->op; temp = temp <<4;
                    temp
                    += token_table[i]->nixbpe;
                    temp
                    = temp <<12;

                    opcode_table[opcode_table_line_num].opcode
                    = temp;
                }
            }
        }
    }
}

```

```

opcode_table[opcode_table_line_num].format
=3;

opcode_table[opcode_table_line_num].addr =
token_table[i]->locctr;

opcode_table[opcode_table_line_num].csec_num = token_table[i]->csect;

opcode_table_line_num++;

i++;

continue;
}
else if
(token_table[i]->operand[0][0] == '#') {
//indirect

token_table[i]->nixbpe +=1;

token_table[i]->nixbpe =
token_table[i]->nixbpe <<4;

temp
= inst_table[index]->op; temp = temp <<4;
temp
+= token_table[i]->nixbpe;

temp
= temp <<12;

strcpy(temp_char, token_table[i]->operand[0]);

//printf("!!%sWn", temp_char);

temp_char[0] = '0'; // 문자열로 되어있는 " # '
주소" 에서 주소만 파싱하기 위해 '#'을 '0'으로
치환함

//빈칸
temp

+= atoi(temp_char);

opcode_table[opcode_table_line_num].opcode
= temp;

opcode_table[opcode_table_line_num].format
=3;

opcode_table[opcode_table_line_num].addr =
token_table[i]->locctr;

opcode_table[opcode_table_line_num].csec_num = token_table[i]->csect;

opcode_table_line_num++;

i++;

continue;
}

```

```

else {

token_table[i]->nixbpe +=3;

token_table[i]->nixbpe =
token_table[i]->nixbpe <<4;
}
i f
(token_table[i]->operand[1][0] == 'X') {

token_table[i]->nixbpe +=8;
} // x 즉 루프
문일때

i f
(token_table[i]->operator[0] == '+') { //4형식
일때

token_table[i]->nixbpe +=1;

temp
= inst_table[index]->op; temp = temp <<4;
temp
+= token_table[i]->nixbpe;
//
printf("%xWt", temp);
temp

= temp <<20;

opcode_table[opcode_table_line_num].opcode
= temp;

opcode_table[opcode_table_line_num].format
=4;

opcode_table[opcode_table_line_num].addr =
token_table[i]->locctr;

opcode_table[opcode_table_line_num].csec_num = token_table[i]->csect;

opcode_table_line_num++;

i++;

continue;
}
else if
(strcmp(token_table[i]->operator, "RSUB") ==0)
{

temp

= inst_table[index]->op;

temp
= temp <<4; // op코드를 4번 왼쪽으로 쉬프트하
여 nixbpe가 들어올 자리를 만든다.

temp

+= token_table[i]->nixbpe;

temp

= temp <<12;

opcode_table[opcode_table_line_num].opcode

```

```

= temp;

opcode_table[opcode_table_line_num].format
=3;

opcode_table[opcode_table_line_num].addr =
token_table[i]->locctr;

opcode_table[opcode_table_line_num].csec_num = token_table[i]->csect;

opcode_table_line_num++;

i++;

continue;
}
else {

token_table[i]->nixbpe +=2;

here:
temp =

inst_table[index]->op;
temp = temp
<<4;// op코드를 4번 왼쪽으로 쉬프트하여
nixbpe가 들어올 자리를 만든다.
temp +=

token_table[i]->nixbpe;
temp = temp
<<12;// opcode 와 nixbpe 가 더하여진 값이 뒤
에 16진수 000을 넣기 위해 왼쪽으로 12번 시프트

//printf("%.6xWn", temp);

//printf("%s
:%x, %x Wn",
token_table[i]->operand[0],addr_found(token_t
able[i]->operand[0]), token_table[i
+1]->locctr);

i f
(addr_found(token_table[i]->operand[0],
csec_num) - token_table[i +1]->locctr >=0) {

//printf("%x %xWt",
addr_found(token_table[i]->operand[0]),
token_table[i + 1]->locctr);

temp
= temp +
addr_found(token_table[i]->operand[0],
csec_num) - token_table[i +1]->locctr;
}
else { //target
주소 - 현재 program counter <0 이라면 보수를
이용하여 푼다

//printf("%s %x %xWt",
token_table[i]->operand[0],
addr_found(token_table[i]->operand[0]),

```

```

token_table[i + 1]->locctr);

temp
+= (4096 - token_table[i +1]->locctr +
addr_found(token_table[i]->operand[0],
csec_num) );
}

opcode_table[opcode_table_line_num].opcode
= temp;

opcode_table[opcode_table_line_num].format
=3;

opcode_table[opcode_table_line_num].addr =
token_table[i]->locctr;

opcode_table[opcode_table_line_num].csec_num = token_table[i]->csect;

opcode_table_line_num++;
}
else if
(inst_table[index]->format ==2) { //2형식일때
temp =
inst_table[index]->op <<8; //8번 시프트 했다는
것은 16진수 B4 라면 B400이 됨을 의미한다.
i f
(strcmp(token_table[i]->operand[0], "X") ==0)
{
temp
+= (16 *1); // 16을 곱하는 이유는 2번째 자리
에 넣어야 하기 때문이다!
}
else if
(strcmp(token_table[i]->operand[0], "A") ==0)
{
temp
+= (16 *0);
}
else if
(strcmp(token_table[i]->operand[0], "T") ==0)
{
temp
+= (16 *5);
}
else if
(strcmp(token_table[i]->operand[0], "S") ==0)
{
temp
+= (16 *4);
}
i f
(strcmp(token_table[i]->operand[1], "X") ==0)
{
temp
+=1;
}

```

```

        else if
        (strcmp(token_table[i]->operand[1], "A") ==0)
        {
            temp
            +=0;
        }
        else if
        (strcmp(token_table[i]->operand[1], "T") ==0)
        {
            temp
            +=5;
        }
        else if
        (strcmp(token_table[i]->operand[1], "S") ==0)
        {
            temp
            +=4;
        }

//printf("%xWn", temp);

opcode_table[opcode_table_line_num].opcode
= temp;

opcode_table[opcode_table_line_num].format
=2;

opcode_table[opcode_table_line_num].addr =
token_table[i]->locctr;

opcode_table[opcode_table_line_num].csec_num
= token_table[i]->csect;

opcode_table_line_num++;
    }
    else { //해당 operator가
inst_table에 존재하지 않을 때
        i f
        (strcmp(token_table[i]->operator, "BYTE") ==0)
        {
            i f
            (token_table[i]->operand[0][0] != '=') { //
X'F1' 에 대한 예외 처리

strcpy(temp_char, token_table[i]->operand[0]);

//printf("%sWn", temp_char);

temp_char[strlen(temp_char) -1] ='W0';

temp_char[0] ='0';

temp_char[1] ='0';//temp_char = "00F1"

//printf("%dWn", strtol(temp_char, NULL, 16));

opcode_table[opcode_table_line_num].opcode

```

```

= strtol(temp_char, NULL, 16);

opcode_table[opcode_table_line_num].format
=0;

opcode_table[opcode_table_line_num].addr =
token_table[i]->locctr;

opcode_table[opcode_table_line_num].csec_num
= token_table[i]->csect;

        printf("BYTE X %x %dWn",
opcode_table[opcode_table_line_num].opcode,
opcode_table[opcode_table_line_num].format);

opcode_table_line_num++;
    }
    i f
    (token_table[i]->operand[0][1] == 'X') { //
=X'F1'에 대한 예외처리

strcpy(temp_char, token_table[i]->operand[0]);

//printf("%sWn", temp_char);

temp_char[strlen(temp_char) -1] ='W0';

temp_char[0] ='0';

temp_char[1] ='0';

temp_char[2] ='0'; // temp_char = "000F1"

//printf("%dWn", strtol(temp_char, NULL, 16));

opcode_table[opcode_table_line_num].opcode
= strtol(temp_char, NULL, 16);

opcode_table[opcode_table_line_num].format
=0;

opcode_table[opcode_table_line_num].addr =
token_table[i]->locctr;

opcode_table[opcode_table_line_num].csec_num
= token_table[i]->csect;

        //
        printf("BYTE X %x %dWn",
opcode_table[opcode_table_line_num].opcode,
opcode_table[opcode_table_line_num].format);

opcode_table_line_num++;
    }
    i f
    (token_table[i]->operand[0][1] == 'C') { //
=C' ' 일때

        //
        printf("/%sWn", token_table[i]->operand[0]);

```

```

strcpy(temp_char,token_table[i]->operand[0]);

        f o r
(int k =0; k < strlen(temp_char)-4; k ++) { //
temp_char0i      "C'EOF'"      일때      strlen
(temp_char)-4는 3, 즉 EOF 의 strlen0이 나온다.

        temp_hex = temp_hex << (8); //16진
수 상에서 한칸 옆으로 옮기기 위하여

        temp_hex += temp_char[k +3];
//temp_char[3] = 'E'= 45 temp_char[4] = 'O'
= 4F temp_char[5] ='F'=46

        //000000 + 000045 = 000045, 000045
<< 8 = 004500 , 004500 + 00004F = 00454F,
00454F << 8 = 454F00 , 454F00 + 000046 =
454F46.

        }

opcode_table[opcode_table_line_num].opcode
= temp_hex;

opcode_table[opcode_table_line_num].format
=0;

opcode_table[opcode_table_line_num].addr =
token_table[i]->locctr;

opcode_table[opcode_table_line_num].csec_nu
m = token_table[i]->csect;

        //
printf("BYTE      C      %x      %dWn",
opcode_table[opcode_table_line_num].opcode,
opcode_table[opcode_table_line_num].format);

opcode_table_line_num++;

        }

        e l s e
if(strcmp(token_table[i]->operator,"WORD")
==0) {

        temp =0;

opcode_table[opcode_table_line_num].opcode
= temp;

opcode_table[opcode_table_line_num].format
=0;

opcode_table[opcode_table_line_num].addr =
token_table[i]->locctr;

opcode_table[opcode_table_line_num].csec_nu
m = token_table[i]->csect;

opcode_table_line_num++;

```

```

        }

        //      printf("0Wn");

        }
        i++;

    }

    //printf("%xWn",addr_found("CLOOP"));
    printf("-----Wn");
    for (int i =0; i <
opcode_table_line_num; i ++) {
        printf("%.6x      %x      %dWn",
opcode_table[i].opcode,opcode_table[i].addr,o
pcode_table[i].csec_num);
    }

    printf("*****Wn");
    return 0;

}

/

*
-----
-----
* 설명 : 주어진 csec_num와 문자열을 가지고
심볼테이블과 리터럴테이블을 뒤지면서 존재하는
지 찾는 함수이다.
* 매개 : 문자열 서브루틴 번호
* 반환 : 주소 반환
*
-----
-----
*/
int addr_found(char *str,int csec_num) {
    int i;
    int index;
    int count =0;
    for (i =0; i < symbal_line;i ++) {
        if (strcmp(str,
sym_table[i].symbol) ==0 &&
sym_table[i].csec ==csec_num) {
            r e t u r n
sym_table[i].addr;
        }
    }
    for (i =0; i < literal_line; i ++) {
        if (strcmp(str,
literal_table[i].literal) ==0) {
            //printf("in literal_table
: ");
            r e t u r n
literal_table[i].addr;
        }
    }
    return 0;
}

/
*
```

```

-----
-----
* 설명 : 입력된 문자열의 이름을 가진 파일에 프
로그래밍의 결과를 저장하는 함수이다.
*      여기서 출력되는 내용은 object code
(프로젝트 1번) 이다.
* 매 계 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프
로그래밍의 결과를 표준출력으로 보내어
*      화면에 출력해준다.
*
*
-----
-----
*/
void make_objectcode_output(char * file_name)
{
    FILE* fp;
    fp = fopen(file_name, "w");
    int j =0;
    int i =0;
    int index =0;
    int count =0;
    char buffer[5000] = { 0, };
    int temp_addr =0;
    char temp_buffer[5000] = { 0, };
    for (i =0; i <=csec_num; i ++ ) {
        for (j =0; j < symbal_line; j
++ ) {
            if (sym_table[j].csec
== i) {
                fprintf(fp, "H");
                fprintf(fp,
"%s", sym_table[j].symbol);
                fprintf(fp,
"%0.6x", sym_table[j].addr);
                fprintf(fp,
"%0.6x", found_bigaddr(i));
                fprintf(fp,
"Wn");
                if (i ==0) {
                    while
(index < line_num) {
                        i
                        f
                        (strcmp(token_table[index]->operator, "EXTDEF"
) ==0 && token_table[index]->csect == i) {
                            fprintf(fp,
"%s",
token_table[index]->operand[0]);
                            fprintf(fp,
"%0.6x",
addr_found(token_table[index]->operand[0], i));

```

```

                        i
                        f
                        (token_table[index]->operand[1][0] !=NULL) {
                            f p r i n t f ( f p , " % s " ,
token_table[index]->operand[1]);
                            fprintf(fp,
"%0.6x",
addr_found(token_table[index]->operand[1],
i));
                        }
                    }
                    index++;
                }
            }
            fprintf(fp, "Wn");
        }
        f p r i n t f ( f p ,
"R");
        index =0;
        while (index <
line_num) {
            i
            f
            (strcmp(token_table[index]->operator, "EXTREF"
) ==0 && token_table[index]->csect == i) {
                fprintf(fp,
"%s",
token_table[index]->operand[0]);
            }
            i
            f
            (token_table[index]->operand[1][0] !=NULL) {
                fprintf(fp,
"%s",
token_table[index]->operand[1]);
            }
            if (token_table[index]->operand[2][0]
!=NULL) {
                fprintf(fp,
"%s",
token_table[index]->operand[2]);
            }
        }
    }
}

```



```

    }
    }

    index++;
    }
    fprintf(fp,
"Wn");
    for (int k =0;
k < opcode_table_line_num; k ++) {
        i f
(opcode_table[k].csec_num == i) {

            if (opcode_table[k].format ==2) {

                sprintf(temp_buffer, "%.4x",
opcode_table[k].opcode);

                strcat(buffer, temp_buffer);

                memset(temp_buffer, 0,
sizeof(temp_buffer));

            }

            else if (opcode_table[k].format ==4) {

                sprintf(temp_buffer, "%.8x",
opcode_table[k].opcode);

                strcat(buffer, temp_buffer);

                memset(temp_buffer, 0,
sizeof(temp_buffer));

            }

            else {

                if (opcode_table[k].opcode
==0) {

                    sprintf(temp_buffer,
 "%.6x", opcode_table[k].opcode);

                }

                else if
(opcode_table[k].opcode <256) {

                    sprintf(temp_buffer,
 "%.2x", opcode_table[k].opcode);

                }

                else

                    sprintf(temp_buffer,
 "%.6x", opcode_table[k].opcode);

```

```

                strcat(buffer, temp_buffer);

                memset(temp_buffer, 0,
sizeof(temp_buffer));

            }

            i f
(strlen(buffer) >54) {

                strcat(buffer, "Wn");

                fprintf(fp, "T");

                fprintf(fp, "%.6x ", temp_addr);

                temp_addr = opcode_table[k +1].addr;

                fprintf(fp, "%.2x ", strlen(buffer)/2);

                fprintf(fp, "%s", buffer);

                memset(buffer, 0, sizeof(buffer));

            }

            i f
(strlen(buffer) /2 >0) {

                fprintf(fp, "T");

                fprintf(fp, "%.6x ", temp_addr);

                fprintf(fp, "%.2x ", strlen(buffer) /2);

                fprintf(fp, "%s", buffer);

                memset(buffer, 0, sizeof(buffer));

                fprintf(fp, "Wn");

            }

            temp_a d d r
=0;

            break;

        }

        }fprintf(fp, "Wn");

    }

    fclose(fp);
    /* add your code here */

}

/

-----
-----
-----

* 설명 : 같은 csec_num 중에서 제일 큰 주소
를 찾기위한 함수이다.

```

```

*          object파일을 만들때 각각의
서브루틴의 크기를 저장한다.
* 매개 : 서브루틴 번호
* 반환 : 제일큰 주소 반환
*

```

```

-----
-----
-----

```

```

*/
int found_bigaddr(int csec) {
    int temp =0;
    int i =-0;
    while (i < line_num) {
        if (token_table[i]->csect ==
csec) {
            i
            f
            (token_table[i]->locctr > temp) {
                temp
                =
            token_table[i]->locctr;
            }
        }
        i++;
    }
    return temp;
}

```