숭실대학교 Soongsil University

시스템프로그래밍 2021 보고서

보고서 제출서약서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

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

과목	시스템프로그래밍 2021
과제명	ControlSection assembler 구현 (Program 프로젝트 #1)
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20180637 황성현 (출석번호 101번)
제출일	2021년 05월 05일

차 례

1장 프로젝트 동기/목적

assembly program을 컴퓨터에서 실행시키기 위한 첫 단계인 assembly program을 object program로 바꾸는 'assmbler'를 제작한다. assembly program은 control section 방식을 사용해 만들어져있다.

2장 설계/구현 아이디어

assembler가 올바르게 작동하기위한 전체적인 알고리즘은 다음과 같다.

1. machine instruction data를 읽어드린다.
2. 프로그램 내 instruction_table을 만든다.
3. assembly code를 읽어드려 input_data를 만든다.
4. assembly code 각 명령의 라인별로 명령어를 LABEL, OPERATOR, OPERAND로 parsing 해 token_table을 만든다.
5. token_tablel로 symbol_table을 만든다.
6. token_tablel로 literal_table을 만든다.
7. token_table, symbol_table, literal_table을 사용해 실제 object code를 만든다.
8. object code를 모아 object program을 만든다.

이중 1~4번은 이전 과제3에서 이미 구현한 것이고 5~8번을 추가로 구현하였다.
5,6번은 assem_pass1(), 7번은 assem_pass2(), 8번은 make_objectcode_output()함수로 구현하였다.

가)int token_parsing(char *str)

과제1에서 구현한 어셈블리 명령어 str을 parsing해 token_table을 만드는 함수이다. 여기에 어셈블리 명령에서 object code의 'nixbpe' bit를 결정하는 부분을 추가로 구현한다.
n=0,i=0이면 sic 명령어, n=1,i=0이면 indirect addressing, n=0,i=1이면 immediate addressing, n=1,i=1이면 simple addressing이다. token_table의 operand[0]이 '#'으로 시작하면 immediate addressing, '@'로 시작하면 indirect addressing이다.
x=1이면 indexed addressing, b=1이면 base-relative addressing, p=1이면 pc-relative addressing, b=0,p=0,e=1이면 exptend format이다. operand[1]이 "X"인데 operand[0]도 길이가 1이 아니면(register가 아니면) indexed addressing, operator가 '+'로 시작하면 exptend format, 그렇지 않으면 pc-relative이다.

나)static int assem_pass1(void)

assembly 프로그램이 들어있는 input_data파일을 한줄씩 읽어 parsing해 token_table을 만들고 이를 가지고 symbol_table과 literal_table을 만든다. symbol_table은 symbol 이름과 주소로 이루어져있다. token_table의 label이 symbol_table에 이미 존재하지 않으면 symbol의 이름을 label로하고 그 주소를 현재 loccation counter인 locctr로 한다. 만약 이미 존재하면 에러를 설정한다.
literal_table도 literal 이름과 주소로 이루어져있다. operand[0]이 '='로 시작하면 literal이다. 만약 literal_table에 현재 명령어의 literal이 없으면 literal_table에 이름을 추가한다. literal의 주소는 어셈블리 프로그램이 종료되거나(operator="END") operator가 "LTORG"일 때 locctr로 한다. literal_table에서 section마다 object code가 만들어지는 범위를 알아야 하므로 litfid[]배열을 만들어 각 section에서 LTORG로 object code를 만들시 생성되는 첫 번째 literal_table의 인덱스를 저장하였다.
주소를 정해주는 locctr는 현재 명령어의 메모리 주소를 나타내는데 start address부터 시작해 증가량이 명령어 object code의 절반 길이이다. operator가 기계어 명령어일 경우 1,2,3,4형식이면 object code의 길이가 각각 2,4,6,8로 locctr는 각각 1,2,3,4,만큼, 즉 format만큼 증가하게 된다. operator가 "WORD"이면 object code 길이가 6이므로 locctr가 3만큼 증가한다. "BYTE의" 경우 operand[0][0]이 'X'면 16진수로 숫자 두 개마다 1씩증가하고, 'C'면 char형으로 char당 1byte이므로 16진수로 바꾸면 길이가 두배가 되기 때문에 locctr

가 char당 1씩 늘어난다. “LTORG”나 “END”에의해 literal_table에 주소가 쓰여질 때 마다 “BYTE”와 똑같이 lottcr가 늘어난다.
operator가 “RESW”면 word의 개수인 operand*3씩 locctr가 늘어난다. “RESB”면 operand만큼 locctr이 늘어난다.
operator가 “EQU”인 연산은 symbol의 값을 알아보기 쉽게 정의하는 것이다. operand[0]이 “*”면 현재 주소값이 symbol의 값이 된다. 두 개의 symbol의 연산을 값으로 할수도 있는데 더하기와 빼기만 계산하였다.
예를 들어 “MAXLEN EQU BUFEND-BUFFER”이면 MAXLEN=BUFEND-BUFFER이다. ”EQU”의 값은 16진수 6자리이므로 locctr는 3만큼 증가한다.
operator가 “CSECT”이면 새로운 프로그램 section이 생기게된다. 이전 프로그램의 길이를 locctr과 startaddress의 차로 계산하고 locctr을 0으로 초기화한다.
operator가 “EXTDEF”이면 operand를 extdef[prog_num]에 저장한다. operator가 “EXTREF”이면 operand를 extref[prog_num]에 저장한다. 나중에 원활한 탐색을 위해 operand마다 ‘,’를 넣어 구분한다.

다)void make_symtab_output(char *file_name)

symbol_table을 출력한 file_name을 읽기전용 모드로 연다. symbol_table은 화면출력을 하라고 명세서에 적혀있어 main()에서 매게변수 file_name을 NULL로 호출하고 NULL일 때 stdout으로 출력하게 했다.
출력할 때 section을 나눠 출력하는데 그 구분을 ‘₩n’으로 한다. symbol_table에서 address가 0이면 각 section의 첫 번째 symbol이므로 address가 0일 때 마다 ‘₩n’을 출력한다.

라)void make_literaltab_output(char *file_name)

literal_table을 출력한 file_name을 읽기전용 모드로 연다. literal_table도 화면출력을 하라고 명세서에 적혀있어 main()에서 매게변수 file_name을 NULL로 호출하고 NULL일 때 stdout으로 출력하게 했다.
literal_table에는 {C’EOF’,30} 같은 형식으로 들어가는데, 출력시 “EOF 30”같이 출력되므로 literal은 ”EOF”만 따로 buffer에 넣고 출력한다.

마)static int assem_pass2(void)

assem_pass2에서 token_table, inst_table, symbol_table, literal_table로 실제 object code를 만든다. object code는 ob_table에 저장된다. ob_table은 obcode, 현재 명령 라인의 pc, object code가 저장되는 메모리 크기 length로 이루어져있다. obid는 object code table index이다.
object code가 생성되는 operator는 다음과 같다.

1)machine code

기계어 코드는 1,2,3,4형식이 있다. 1형식은 object code가 opcode 2자리 뿐이므로 inst_table[]->op를 ob_table[obid].obcode로 옮긴다.
2형식의 경우 object code가 명령어 두자리와 레지스터연산자 번호 두 개가 합친 4자리이다. 1형식과 똑같이 inst_table[]->op를 ob_table[obid].obcode로 옮겨야 하는데 4자리 중 앞 두자리가 opcode여서 opcode를 16^2배 해야 하므로 inst_table[]->op를 8번 shift left한 값을 더하는거로 한다. 그 뒤 레지스터 operand는 register_table에 저장해둔 번호로 한다. 예를들어 “COMPR A,S”이면 COMPR의 opcode ‘A0’와 A레지스터 번호 ‘0’, S 레지스터 번호 4를 합치기위해 ‘A0’<<8, ‘0’<<4, ‘4’<<0을 더해 16진수 A004를 만든다. 레지스터 operand가 하나일 경우 두 번째 레지스터 번호를 0으로 한다.
3형식의 경우 opcode, nixbpe, disp의 합으로 이루어진 3byte object code이다. 원래 opcode는 8bit지만 3형식에 쓰이는 opcode는 전부 하위 2bit가 ‘00’이므로 하위 2bit를 없애고 opcode를 6bit로 생각하기로 한다.
따라서 opcode의 하위 2bit와 nixbpe의 상위 2bit인 ni가 16진수로 환산 시 간은 자리에 위치하게 된다. 예를들어 opcode가 2진수 00001100이고 nixbpe가 110010이면 둘을 합치면 2진수 000011110010이되고 16진수 0F2가 돼야한다. 앞의 둘을 합친것이 3byte 이므로 disp은 3byte 이다. opcode와 nixbpe는 token_table에

저장되어 있다. disp은 operand의 위치를 가리키는데, n=1, i=1일 때 p=1이면 pc-relative, b=1이면 base-relative이지만 control section 어셈블리 방식에선 pc-relative만 사용하므로 p=1인 경우만 생각했다. 이 때 또 operand가 literal인 경우와 아닌 경우로 나눌 수 있다. literal인 경우 literal_table에서 literal이 정의된 주소를 찾아 현재 pc인 locctr룰 뺀 literal_table[litline].addr - locctr를 disp으로 한다. (literal_table의 index를 찾는 search_literal()함수를 만들어 사용하였다.) literal이 아닌 경우 symbol_table에서 정의된 주소를 찾아 pc를 뺀 sym_table[symid].addr - locctr를 disp으로 한다. (symbol_table의 index를 찾는 search_symbol()함수를 만들어 사용하였다.)

만약 n=1,i=0이면 indirect 명령이지만 disp을 결정하는 방식은 simple addressing과 같다. n=0,i=1이면 immediate addressing인데 immediate addressing은 operand가 10진수 일수도 있고 symbol 일수도 있다. 만약 symbol이면 simple addressing과 똑같이 disp을 pc-relative로 정한다. 만약 operand가 10진수이면 operand를 disp에 그대로 옮긴다.

이제 addressing 모드에 따라 정해진 disp과 opcode, nixpbe를 더해야 하는데, opcode는 상위 6bit, nixbpe는 그뒤 6bit, disp은 하위 12bit 이다. 그중 opcode의 상위 4bit가 object code의 상위 4bit 이므로 (obcode/16)<<20, obcode 하위4bit+ nixbpe 상위 2bit을 더해야 하므로 ((obcode/%16)+nixbpe>>4), 그뒤 4bit는 nixpbe의 하위 4bit이므로 (nixbpe%16)<<12, disp은 그대로 disp을 더해

object code= ((obcode/16)<<20) + ((obcode/%16)+nixbpe>>4) + ((nixbpe%16)<<12) + disp 이다.

4형식의 경우 opcode, nixbpe, disp의 합으로 이루어진 4byte object code이다. opcode nixbpe는 똑같이 각각 6bit이고 nixbpe는 항상 110001이다. 따라서 disp은 20bit가된다.

operand가 symbol_table에 없으면 extref[]에 operand가 있는지 찾아본다. extref에 symbol간 ‘,’로 구분돼있으므로 ‘,’을 기준으로 strtok()를 사용해 operand와 같은지 확인한다. 만약 extref에 operand가 있으면 disp을 0으로 한다. operand가 symbol_table에 있으면 그 주소를 disp으로 해야하는데 control section에선 한 section의 크기가 12bit를 넘어가지 않으므로 굳이 구현할 필요가 없다. 4형식 object code를 만들 때 object program의 M Record를 만드는데 필요한 주소와 operand 이름을 저장한다.

2)WORD

operator가 “WORD”인 경우 extref의 symbol간 연산이 이뤄진다. 이번 입력의 경우 ‘-’연산만 있지만 후에 필요할 수 있어 ‘+’연산도 계산을 하였다. ‘-’와 ‘+’를 기준으로 operand에 strtok()를 시행해 operand 두 개를 찾는다. 두 operand를 extref에서 찾아보고 extref에 있을 경우 machine code의 4형식때와 마찬가지로 M Record를 위한 현재 명령어 주소와 operand를 각 operand에따라 따로 두 번 저장한다. 마지막으로 object code는 16진수‘000000’으로 3byte가 된다.

3)BYTE

operator가 BYTE면 assem_pass1에서 처럼 ‘X’로시작하는 16진수인지 ‘C’로 시작하는 char형인지를 확인하고 16진수일 경우 object code 길이의 절반이 사용되는 memory의 크기이고 char형이면 object code의 길이가 사용되는 mamory크기이다. 16진수면 operand를 obcode로 몲기도 char형이면 operand를 16진수 아스키코드로 바꿔 object code를 만든다.

4)LTORG

operator가 LTORG이면 object code를 만드는 방법은 operator “BYTE”와 똑같지만 object code를 만들 operand가 literal_table에 있고, 아직 object code가 만들어지지 않은 literal_table의 첫 번째 인덱스인 litfid[]부터 현재 section에서 만들어진 literal_table의 끝까지 object code를 생성한다. LTORG가 나올 때 마다 litid를 1씩 늘려 litfid[]값을 조정해 section마다 올바르게 literal이 정의되도록 하였다. 이작업은 operator가 ‘END’일 때도 일어난다.

이외에 operator “RESW”, “RESB”는 locctr를 증가하는 역할만 수행하고 “CSECT”는 locctr을 0으로 초기화하고, program number를 1 증가시키고, M Recode 인덱스를 증가시키고 object code의 start address부터 떨어진 위치 obloc을 0으로 초기화한다.

바)void make_objectcode_output(char *file_name)

assem_pass2에서 만등 object code를 가지고 실제 object program을 만든다. object program은 section별로 한 개씩 만들어 지고 각 program은 “H” Recod, “D” Recod, “R” Recod, “T” Record, “M” Record, “E” Record로 이루어져있다.

1)H Record

Header Record는 총 19개의 column으로 이루어져 있는데 첫 번째 column은 ‘H’이다. 두 번째부터 7번째까지 column은 program의 이름(START operator의 operand 또는 CSECT operator의 operand)의 왼쪽 정렬된 상태이다. 만약 프로그램 이름이 6글자 미만일 경우 남은 공간은 빈칸으로 한다. 8번째부터 13번째까지 column은 program의 시작 주소인데 control section 어셈블리 프로그램은 memory relocatable 하므로 시작 주소는 0으로한다. 14번째부터 19번째 column은 program의 길이의 16진수 6자리이다. 프로그램의 길이는 assem_pass1에서 미리 구해놓은 것이다.

2)D Record, R Record

Define Record와 Refer Record는 “EXTDEX”,“EXTREF” operator의 operand각 각각 순서대로 나열돼있다. 이는 각각 write_def(), write_ref() 함수에 구현했다. D Record의 첫 번째 column의 ‘D’이고 그 뒤로 extdef[]에서 ‘,’를 기준으로 EXTDEF의 operand를 구분해 buf에 저장하는데 항상 6자리 char로 왼쪽정렬로 저장한다. 그리고 그다음 6개 column으로 EXTDEF된 symbol이 정의된 주소를 16진수로 buf에 저장한다. 이 과정을 extdef[]의 끝까지 반복한다.

R Record는 첫 번째 column이 ‘R’이고 두 번째 column부턴 D Record와 다르게 EXTREF의 symbol은 주소를 알수 없으므로 extref[]에서 ‘,’를 기준으로 구분한 symbol 이름 뿐이다.

3)T Record

Text Record는 실제 object code로 이루어진 Record이다. 1번째 column은 ‘T’이다. 두 번째부터 7번째 column은 현재 Record에 쓸 첫 object code의 주소의 16진수 6자리이다. 8,9번째 column은 현재 Record의 object code 길이의 byte단위 16진수 2자리이다. 10번째부터 69번째 column은 실제 object code의 순차적 나열이다. object code column이 총 60개므로 8,9번째 column의 최대값은 1D이다.

object code의 길이는 2,4,6,8중 하나인데, ob_table의 length는 byte단위 이므로 이 두배를 object code의 길이로하여 switch문을 수행한다. Record의 출력에 사용되는 buf에 길이에 따라 object code를 추가한다. buf에 저장된 record가 실제로 출력파일에 출력되는 기준은 다음과 같다.

1. buf에 저장된 object code의 길이가 60 초과인 경우
2. 현재 프로그램의 object code를 모두 buf에 저장한 경우
3.object code를 만들지 않는 명령어를 맞닥드린 경우

이 세가지 경우에 buf의 내용을 출력하고 buf를 다음 T Record 형식으로 초기화하는 initial_bufT() 함수를 호출한다.

4)M Record

Modification Record는 EXTREF symbol의 주소를 object code에 추가하기 위해, 즉 변경을 위해 사용된다. M Record의 첫 번째 column은 ‘M’이다. 두 번째부터 7번째 column은 변경해야하는 object code의 주소이다. 8,9번째 column은 변경해야하는 objectcode의 길이이다. 예를들어 +STX LENGTH의 경우 object

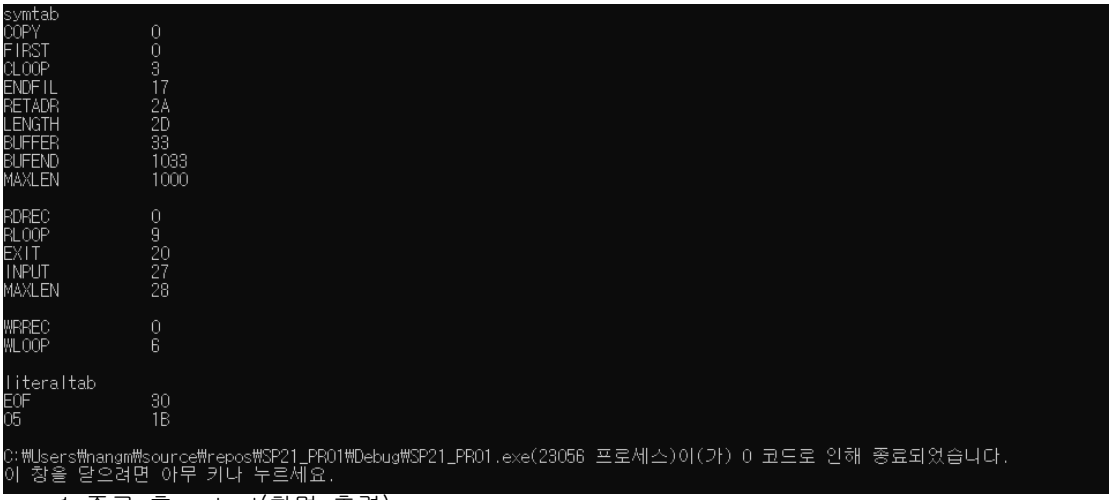
code가 “13100000“인데 이중 하위 5자리를 변경해야 하므로 8,9번째 column은 ‘05’이다.
10번째 column은 Modification flag인데 symbol값을 더해야하면 ‘+’, 빼야하면 ‘-’이다. 예를들어
WORD BUFEND-BUFFER이면 BUFEND에 대해선 ‘+’이고 BUFFER에 대해선 ‘-’이다.
11번째부터 16번째 column은 변경시키는 주소의 이름, 즉 symbol의 이름이다. M Record는 이전에 assem_pass2에서 미리 만들어둔 m_record[]를 그대로 출력하면 된다.

5)E Record

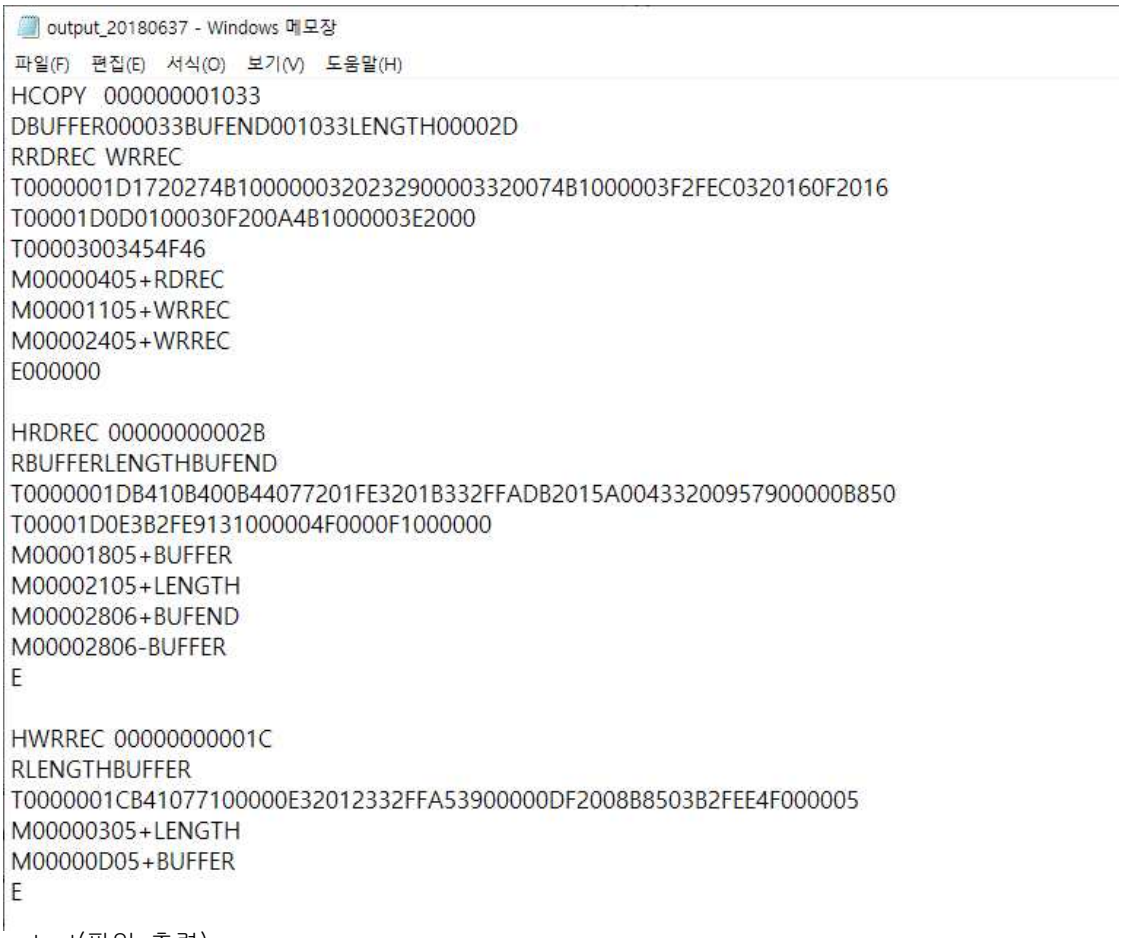
End Record는 object program의 끝을 알리는 Record이다. E Record의 첫 번째 column은 ‘E’이고 두 번째부터 7번째 column은 프로그램의 시작주소이다. 하지만 첫번째 section의 경우 start address를 알기 때문에 문제가 없지만 두 번째 section부터는 시작주소가 없기 때문에 E Record를 ‘E’하나로 한다.

H Record가 출력되면 D Record를 D Record가 출려되면 R Record를, R Record가 출려되면 T Record를, T Record가 전부 출력되면 M Record를 M Record가 전부 출력되면 E Record를 출력한다. M Record는 m_record[] 베열로 관리해 전부 출력된걸 알수 있지만 T Record는 전부 출력한걸 알수 없으므로 ob_table을 순회하면서 동시에 token_table을 순회하고 operator가 “CSECT” 이거나 “END”인지를 동시에 확인하면서 T Record를 출력한다.

3장 수행결과(구현 화면 포함)



pass1 종료 후 output(화면 출력)



output(파일 출력)

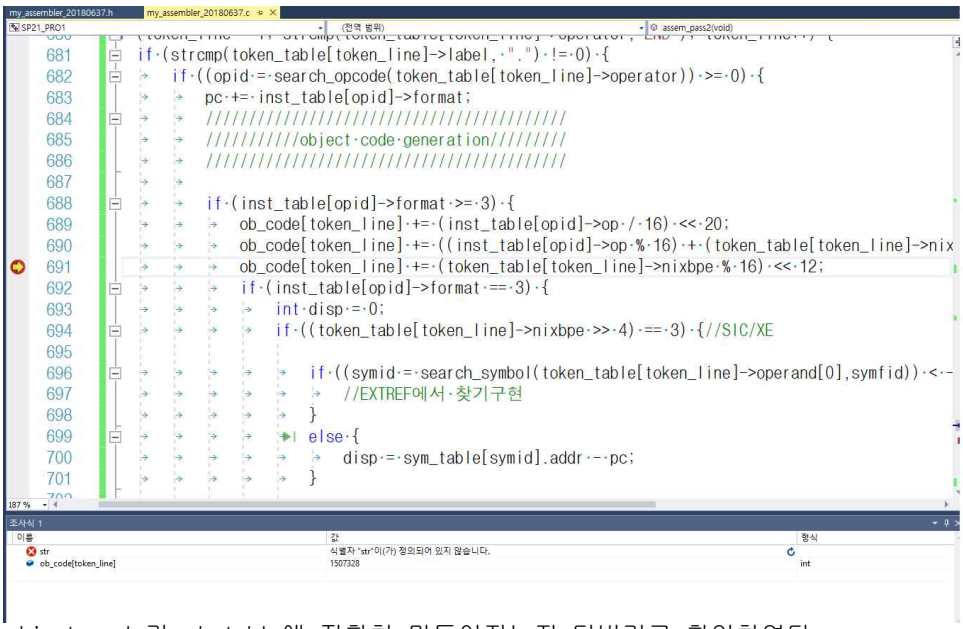
4장　결론 및 보충할 점

comtrol section 어셈블리 프로그램을 object 프로그램으로 만드는 어셈블러를 제작하였다. control section은 프로그램을 여러 개로 쪼개 각각의 프로그램에 독립적으로 memory relocate를 실행하게 만든다. inst_table. token_table, symbol_table, literal_table을 pass1에서 만들었고 pass2에서 실제 object code를 만들었다. object code와 프로그램의 정보를 가지고 object program을 제작하였다.

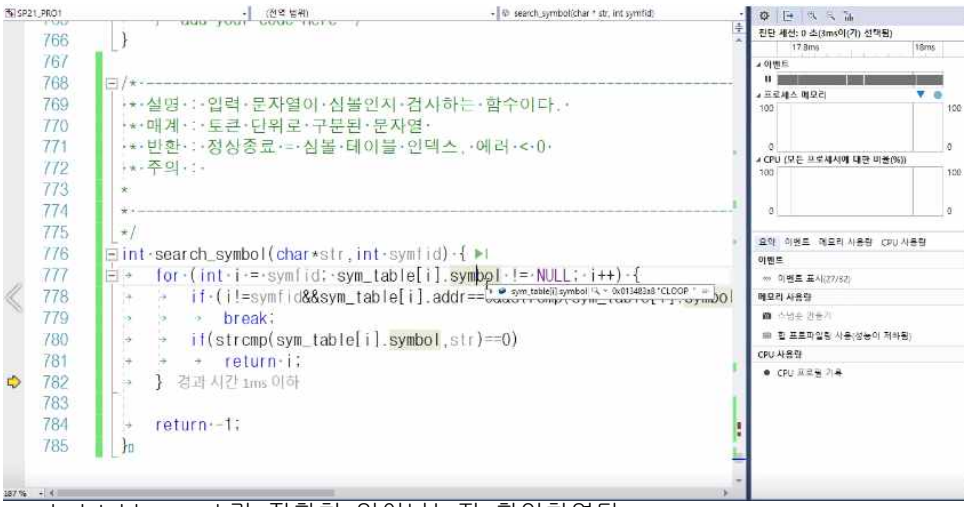
어셈블러를 구현하면서 control section이 가지는 독립성의 의미와 독립적인 data structure를 만드는 법을 다양하게 익힐 수 있었다.

프로그램을 길게 구현하면서 같은 내용을 구현하더라도 나중에 좀더 좋은 방법이 생각나서 적용하기도 하였지만 미처 이전 것을 고치지 못한것도 있어 다음 java 구현때 일관성 있는 구현법으로 만들고 싶다. input과 output에 관한 처리를 더 가독성이 좋게, 간단하게 구현하는 법을 중간중간 학습을 통해 익혀 같은 입출력도 다르게 된 부분이 있어 나중에 일관성있게 구현해야 한다. 이번 프로젝트 때 table search를 binary search로 구현하려 했으나 당장 시간이 부족하여 하지 못해 나중에 변경하고 싶다.

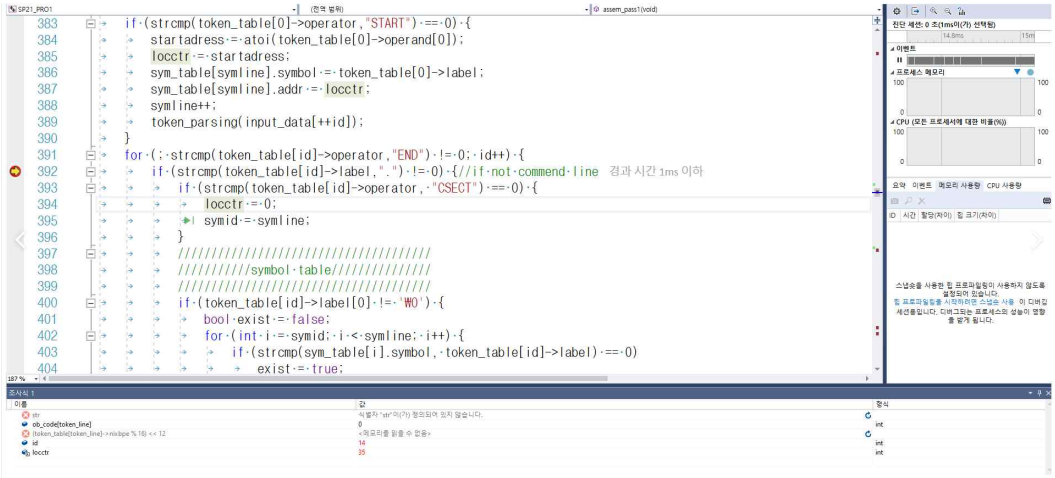
5장　디버깅



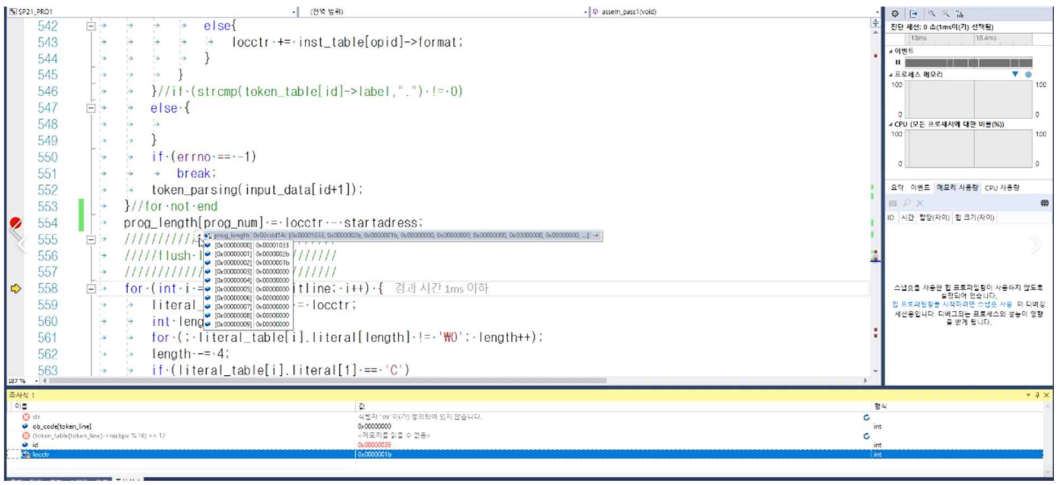
object code가 ob_table에 정확히 만들어지는지 디버거로 확인하였다.



symbol_table serch가 정확히 일어나는지 확인하였다.



location counter가 올바르게 증가하는지 디버거로 확인하였다.



ovject progam에 사용할 프로그램의 크기가 정확히 저장되었는지 디버거를 사용해 16진수로 확인하였다.

6장 소스코드(+주석)

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

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

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

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

/*
 * 어셈블리 할 소스코드를 토큰단위로 관리하기 위한 구조체 변수이다.
 * operator는 renaming을 허용한다.
 * nixbpe는 8bit 중 하위 6개의 bit를 이용하여 n,i,x,b,p,e를 표시한다.
 */
struct token_unit
{
	char *label;
	char *operator;
	char operand[MAX_OPERAND][20];
	char comment[100];
	char nixbpe;
};

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

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

/*
* 리터럴을 관리하는 구조체이다.
* 리터럴 테이블은 리터럴의 이름, 리터럴의 위치로 구성된다.
* 추후 프로젝트에서 사용된다.
*/
struct literal_unit
{
	char *literal;
	int addr;
};

typedef struct symbol_unit symbol;
symbol sym_table[MAX_LINES];

typedef struct literal_unit literal;
literal literal_table[MAX_LINES];

static int locctr;
//---------------

static char *input_file;
static char *output_file;
```

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

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


//-----additional define---------

struct register_unit {
        char name[3];
        int num;
};
typedef struct register_unit registers;

registers register_table[9] = {

{"A",0},{"X",1},{"L",2},{"B",3},{"S",4},{"T",5},{"F
",6},{"PC",8},{"SW",9}
};

struct object {
        int ob_code;
        int pc;
        int length;
};
struct object ob_table[MAX_LINES];
static char* extdef[10];
static char* extref[10];
static int prog_length[10];
static int pc[MAX_LINES];
static int symfid[MAX_LINES] = { 0 };
static int litfid[MAX_LINES] = { 0 };//현재 정의
되지 않은 첫번째 literal_table index
static char m_record[MAX_LINES][17];
int search_symbol(char*str,int symfid);
int search_literal(char*str, int litfid);
void write_def(FILE *fp, int prognum);
void write_ref(FILE *fp, int prognum);
void initial_bufT(char* buf, int obid, int
*buf_length);
```

```
/*
 * 화일명 : my_assembler_20180637.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 <stdbool.h>
#include <ctype.h>

// 파일명의 "00000000"은 자신의 학번으로 변경할 것.
#include "my_assembler_20180637.h"

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

        if (assem_pass1() < 0)
        {
                printf("assem_pass1: 패스1 과정에서 실패하였습니다.  ₩n");
                return -1;
        }
        /                                       /
make_opcode_output("output_00000000");


make_symtab_output(/*"symtab_20180637"*/NULL);

make_literaltab_output(/*"literaltab_20180637"*/NULL);
        if (assem_pass2() < 0)
        {
                printf(" assem_pass2: 패스2 과정에서 실패하였습니다.  ₩n");
                return -1;
        }


make_objectcode_output("output_20180637");

        return 0;
}

/*
------------------------------------------------------------------------------
 * 설명 : 프로그램 초기화를 위한 자료구조 생성 및 파일을 읽는 함수이다.
 * 매계 : 없음
 * 반환 : 정상종료 = 0 , 에러 발생 = -1
 * 주의 : 각각의 명령어 테이블을 내부에 선언하지 않고 관리를 용이하게 하기
 *                  위해서 파일 단위로 관리하여
```

```
프로그램 초기화를 통해 정보를 읽어 올 수 있도록
 *                      구현하였다.
 *
------------------------------------
------------------------------------
------
 */
int init_my_assembler(void)
{
        int result;

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

/                                                *
------------------------------------
------------------------------------
------
 * 설명 : 머신을 위한 기계 코드목록 파일을 읽어
기계어 목록 테이블(inst_table)을
 *         생성하는 함수이다.
 * 매계 : 기계어 목록 파일
 * 반환 : 정상종료 = 0 , 에러 < 0
 * 주의 : 기계어 목록파일 형식은 자유롭게 구현한
다. 예시는 다음과 같다.
 *
 *
====================================
====================================
===
 *                    | 이름 | 형식 | 기계어 코드
| 오퍼랜드의 갯수 | NULL|
 *
====================================
====================================
===
 *
 *
------------------------------------
------------------------------------
------
 */
int init_inst_file(char *inst_file)
{
        FILE *file;
        int errno;
        char buf[20];
        /* add your code here */
        file = fopen(inst_file, "r");
        inst_index = 0;
        while (!feof(file)) {
                fgets(buf, 20, file);
                inst_table[inst_index]          =
malloc(sizeof(inst));

memset(inst_table[inst_index]->str,          '₩0',
sizeof(inst_table[inst_index]->str));

                int i;
                unsigned char temp[5] = { '0',
};
                int start = 0;
                ///명령어 저장
                for (i = start; buf[i] != '|';
i++) {

inst_table[inst_index]->str[i - start] = buf[i];
                }
                inst_table[inst_index]->str[i   -
start] = '₩0';
                ++i;
                ///format 저장
                for (; buf[i] != '|'; i++)

inst_table[inst_index]->format  =  (int)(buf[i]  -
48);
                i++;
                ///opcode 저장
                start = i;
                for (; buf[i] != '|'; i++) {
                        temp[i   -   start]   =
buf[i];
                }
                inst_table[inst_index]->op = 0;
                ////////////////////////////
                ///////16^1의자리 bit////////
```

```
                ////////////////////////////
                if ('A' <= temp[0] && temp[0]
<= 'F') {

inst_table[inst_index]->op      +=      (unsigned
char)(temp[0] - 55) * 16;
                }
                else if ('0' <= temp[0] &&
temp[0] <= '9')

inst_table[inst_index]->op      +=      (unsigned
char)(temp[0] - 48) * 16;
                else
                        errno = -1;
                ////////////////////////////
                ///////16^1의자리 bit////////
                ////////////////////////////
                ////////////////////////////
                ///////16^0의자리 bit////////
                ////////////////////////////
                if ('A' <= temp[1] && temp[1]
<= 'F') {

inst_table[inst_index]->op += (temp[1] - 55);
                }
                else if ('0' <= temp[1] &&
temp[1] <= '9')

inst_table[inst_index]->op += (temp[1] - 48);
                else
                        errno = -1;
                ////////////////////////////
                ///////16^1의자리 bit////////
                ////////////////////////////
                ++i;
                //operand개수 저장
                for (; buf[i] != '|'; i++)

inst_table[inst_index]->ops = (int)(buf[i] - 48);
                ++inst_index;
        }
        fclose(file);
        return errno;
}

/                                               *
-----------------------------------
-----------------------------------
------
 * 설명 : 어셈블리 할 소스코드를 읽어 소스코드
테이블(input_data)를 생성하는 함수이다.
 * 매계 : 어셈블리할 소스파일명
 * 반환 : 정상종료 = 0 , 에러 < 0
 * 주의 : 라인단위로 저장한다.
 *
 *
-----------------------------------
-----------------------------------
------
 */
int init_input_file(char *input_file)
{
        FILE *file;
        int errno;

        /* add your code here */
        file = fopen(input_file, "r");
        line_num = 0;
        while (!feof(file)) {//eof전까지 read
                input_data[line_num]            =
malloc(sizeof(char) * 170);
                fgets(input_data[line_num], 170,
file);
                ++line_num;
        }
        fclose(file);
        return errno;
}

/                                               *
-----------------------------------
-----------------------------------
------
 * 설명 : 소스 코드를 읽어와 토큰단위로 분석하고
토큰 테이블을 작성하는 함수이다.
 *        패스 1로 부터 호출된다.
 * 매계 : 파싱을 원하는 문자열
 * 반환 : 정상종료 = 0 , 에러 < 0
 * 주의 : my_assembler 프로그램에서는 라인단위
로 토큰 및 오브젝트 관리를 하고 있다.
 *
-----------------------------------
```

```
------------------------------------------------------
 */
int token_parsing(char *str)
{
	/* add your code here */
	int label_size;
	int operator_size;
	int operand_num = 0;
	int strid = 0;//str index
	int i = 0;
	int operand_size[MAX_OPERAND] = { 0 };
	bool activeN = true;
	bool activeI = true;
	bool activeX = false;
	bool activeB = false;
	bool activeP = false;
	bool activeE = false;
	if (str[0] == '.') {//asemmbliy의 주석
		i = 1;
		label_size = 1;
		if (str[i++] == '₩n') {
			operator_size = 0;
		}
		else {
			for (; str[i] != '₩n'; i++);
			operator_size = i - (label_size + 1);
		}
	}
	else if (str[0] != '₩t') {//label이 있는 경우
		for (i = 0; str[i] != '₩t'; i++);

		label_size = i++;

		for (; str[i] != '₩t'&& str[i] != '₩n'; i++);
		operator_size = i - (label_size + 1);
	}
	else {//label이 없는 경우
		label_size = i++;
		for (; str[i] != '₩t'&& str[i] != '₩n'; i++);
		operator_size = i - (label_size + 1);
	}

//////////////////////////////////////////////////////////
	/*********token_table포인터 메모리 할당 및 초기화***********/

//////////////////////////////////////////////////////////
	token_table[token_line] = malloc(sizeof(char) * 162 + sizeof(char)*(label_size+1) + sizeof(char)*(operator_size+1));
	token_table[token_line]->label = malloc(sizeof(char)*(label_size + 1));
	token_table[token_line]->operator = malloc(sizeof(char)*operator_size + 1);
	memset(token_table[token_line]->label, '₩0', sizeof(char)*(label_size + 1));

memset(token_table[token_line]->operator, '₩0', sizeof(char)*operator_size + 1);
	for (int k = 0; k < 3; k++)

memset(token_table[token_line]->operand[k], '₩0', 20);

memset(token_table[token_line]->comment, '₩0', sizeof(token_table[token_line]->comment));
	token_table[token_line]->nixbpe = 0;

//////////////////////////////////////////////////////////
	/**********token_table LABEL,OPERATOR 할당**********/

//////////////////////////////////////////////////////////
	for (int k = 0; k < label_size; k++)

token_table[token_line]->label[k] = str[k];

token_table[token_line]->label[label_size] = '₩0';
	for (int k = i - operator_size; k < i; k++)
```

```
token_table[token_line]->operator[k - (i - operator_size)] = str[k];

token_table[token_line]->operator[operator_size] = '₩0';
        i++;

//////////////////////////////////////////////////////////
        /***************token_table OPERAND 할당******************/

//////////////////////////////////////////////////////////
        if (str[0] != '.') {
                if (str[i] != '₩0') {
                        for (; str[i] != '₩0'&&str[i] != '₩t'&&str[i] != '₩n'; i++) {
                                if (str[i] == ',') {

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

++operand_num;
                                        strid = 0;

continue;
                                }
                                if (operand_num >= 3) {

fprintf(stderr, "too many operand: %d line₩n", token_line);
                                        errno = -1;
                                }

token_table[token_line]->operand[operand_num][strid++] = str[i];
                        }
                        if (operand_num > 0)
                                if (strcmp(token_table[token_line]->operand[1], "X") == 0) {

                                        if (strlen(token_table[token_line]->operand[0]) != 1)

                                                activeX = true;
                                        }

//////////////////////////////////////////////////////////

/***************token_table comment 할당******************/

//////////////////////////////////////////////////////////

                }
                strid = 0;
                if (str[i++] != '₩0')
                        for (; str[i] != '₩0'&&str[i] != '₩n'; i++) {

token_table[token_line]->comment[strid++] = str[i];
                                if (strid >= 100) {

fprintf(stderr, "too long comment: %d line₩n", token_line);
                                        errno = -1;
                                        break;
                                }
                        }


        }

//////////////////////////////////////////////////////////
        ///////////////token_table nixbpe 할당////////////////////////

//////////////////////////////////////////////////////////
        token_table[token_line]->nixbpe = 0b110000;
        if (token_table[token_line]->operand[0][0] == '@') {//n
```

```
token_table[token_line]->nixbpe -= (1 << 4);
                activeI = false;
        }
        i                                       f
(token_table[token_line]->operand[0][0] == '#')
{//i

token_table[token_line]->nixbpe -= (1 << 5);
                activeN = false;
        }
        if(activeX)//x

token_table[token_line]->nixbpe += (1 << 3);
        char            nixbpe          =
token_table[token_line]->nixbpe;
        if (token_table[token_line]->operator[0]
== '+') {//e

token_table[token_line]->nixbpe += 1;
                activeE = true;
        }
        else if (!activeN && activeI) {//n=0,
i=1

if(isalpha(token_table[token_line]->operand[0][1
]))//p

token_table[token_line]->nixbpe += (1 << 1);

        }
        else{//n=1
                i                               f
(token_table[token_line]->operand[0][0]      !=
'₩0')

token_table[token_line]->nixbpe += (1 << 1);

        }


        ++token_line;
}

/                                               *
-------------------------------------
-------------------------------------
------
 * 설명 : 입력 문자열이 기계어 코드인지를 검사하
는 함수이다.
 * 매계 : 토큰 단위로 구분된 문자열
 * 반환 : 정상종료 = 기계어 테이블 인덱스, 에러
< 0
 * 주의 :
 *
 *
-------------------------------------
-------------------------------------
------
 */
int search_opcode(char *str)
{
        /* add your code here */
        inst_index = 0;
        if (str[0] == '+')//4형식 명령어 59번째부
터 table에 따로 정렬
                inst_index = 59;


        for (; inst_index <= 100; inst_index++)
{
                i                               f
(strcmp(inst_table[inst_index]->str, str) == 0)
                        break;
        }
        if (inst_index == 101)
                inst_index = -1;
        return inst_index;
}

/                                               *
-------------------------------------
-------------------------------------
------
* 설명 : 어셈블리 코드를 위한 패스1과정을 수행하
는 함수이다.
*                       패스1에서는..
*                        1. 프로그램 소스를 스캔하여
해당하는 토큰단위로 분리하여 프로그램 라인별 토
큰
*                       테이블을 생성한다.
*
* 매계 : 없음
* 반환 : 정상 종료 = 0 , 에러 = < 0
```

```
* 주의 : 현재 초기 버전에서는 에러에 대한 검사를 하지 않고 넘어간 상태이다.
*          따라서 에러에 대한 검사 루틴을 추가해야 한다.
*
*
-------------------------------------------------------------------------------
*/
static int assem_pass1(void)
{
        /* add your code here */
        int startadress = 0;
        int symline = 0;
        //int symfid = 0;//현재 section의 첫번째 simbol_table index
        int litline = 0;
        int litid = 0;//litfid의 index
        int opid;
        int id = 0;
        int prog_num = 0;
        locctr = 0;
        char buf[100];
        token_line = 0;
        token_parsing(input_data[0]);
        if (strcmp(token_table[0]->operator,"START") == 0) {
                startadress = atoi(token_table[0]->operand[0]);
                locctr = startadress;
                sym_table[symline].symbol = token_table[0]->label;
                sym_table[symline].addr = locctr;
                symline++;

                token_parsing(input_data[++id]);
        }
        for (; strcmp(token_table[id]->operator,"END") != 0; id++) {
                if (strcmp(token_table[id]->label,".") != 0) {//if not commend line
                        if (strcmp(token_table[id]->operator, "CSECT") == 0) {


                                prog_length[prog_num++] = locctr - startadress;

                                symfid[prog_num] = symline;
                                locctr = 0;
                        }

                        ////////////////////////////////////
                        //////////symbol table///////////////

                        ////////////////////////////////////
                        if (token_table[id]->label[0] != '₩0') {
                                bool exist = false;
                                for (int i = symfid[prog_num]; i < symline; i++) {
                                        if (strcmp(sym_table[i].symbol, token_table[id]->label) == 0)

                                                exist = true;
                                }
                                if (exist) {

                                        fprintf(stderr,"twice define symbol: %s₩n", token_table[id]->label);
                                        errno = -1;

                                        //return errno;
                                }
                                else {

                                        sym_table[symline].symbol = token_table[id]->label;

                                        sym_table[symline++].addr = locctr;
                                }
                        }
```

```
////////////////////////////////////
                        //////////////literal table//////////////

////////////////////////////////////
                        if (token_table[id]->operand[0][0] == '=') {
                                bool exist = false;
                                for (int i = litfid[litid]; i < litline; i++) {
                                        if (strcmp(literal_table[i].literal, token_table[id]->operand[0]) == 0)

                                exist = true;
                                }
                                if (!exist) {

                                literal_table[litline++].literal = token_table[id]->operand[0];
                                }
                        }


//////////////////////////////////////

////////////////EXTDEF/////////////////

//////////////////////////////////////
                        if (strcmp(token_table[id]->operator,"EXTDEF") == 0) {
                                int length = 0;
                                for (int i = 0; i < MAX_OPERAND; i++) {
                                        if (token_table[id]->operand[i][0] == '₩0')

                                break;
                                        for (int k = 0; token_table[id]->operand[i][k] != '₩0'; k++)

                                length++;
                                }

extdef[prog_num] = malloc(sizeof(char)*(length)+3);

memset(extdef[prog_num],'₩0',sizeof(char)*(length)+3);
                                length = 0;
                                for (int i = 0; i < MAX_OPERAND; i++) {
                                        if (token_table[id]->operand[i][0] == '₩0')

                                break;
                                        for (int k = 0; token_table[id]->operand[i][k] != '₩0'; k++)

                                extdef[prog_num][length++] = token_table[id]->operand[i][k];

extdef[prog_num][length++] = ',';
                                }

extdef[prog_num][length] = '₩0';
                        }

//////////////////////////////////////

////////////////EXTREF/////////////////

//////////////////////////////////////
                        else if (strcmp(token_table[id]->operator,"EXTREF") == 0) {
                                int length = 0;
                                for (int i = 0; i < MAX_OPERAND; i++) {
                                        if (token_table[id]->operand[i][0] == '₩0')

                                break;
                                        for (int k = 0; token_table[id]->operand[i][k] != '₩0'; k++)
```

```
        length++;
                                }

extref[prog_num]                                =
malloc(sizeof(char)*(length)+4);

memset(extref[prog_num],                '₩0',
sizeof(char)*(length)+4);
                                length = 0;
                                for (int i = 0;
i < MAX_OPERAND; i++) {
                                        i       f
(token_table[id]->operand[i][0] == '₩0')

        break;
                                        f   o   r
(int k = 0; token_table[id]->operand[i][k] !=
'₩0'; k++)

        extref[prog_num][length++]              =
token_table[id]->operand[i][k];

extref[prog_num][length++] = ',';
                                }

extref[prog_num][length] = '₩0';
                        }

////////////////////////////////////////////////////
                        else                    if
(strcmp(token_table[id]->operator,"RESW")    ==
0) {

                                locctr += 3 *
atoi(token_table[id]->operand[0]);
                        }
                        else                    if
(strcmp(token_table[id]->operator,"RESB") == 0)
{

                                locctr      +=
atoi(token_table[id]->operand[0]);
                        }
                        else                    if
(strcmp(token_table[id]->operator,"EQU")  ==  0)
{
                                        i       f
(strcmp(token_table[id]->operand[0],  "*")  !=  0)
{

char*ptr;


//////////////////////////////////////

//////////////'-'caculation//////////////

//////////////////////////////////////
                                                ptr  =
strchr(token_table[id]->operand[0], '-');
                                                if (ptr
!= NULL) {


        int operand1 = -1, operand2 = -1;

        ptr                                     =
strtok(token_table[id]->operand[0], "-");


        for (int i = symfid[prog_num]; i <
symline; i++) {

                if  (strcmp(sym_table[i].symbol,
ptr) == 0)

                        operand1            =
sym_table[i].addr;

        }

        if (operand1 == -1) {

                fprintf(stderr, "EQU error: not
defime symbol %s", ptr);

                errno = -1;

        }


        ptr = strtok(NULL, "₩0");
```

```
        for (int i = symfid[prog_num]; i < symline; i++) {
                if (strcmp(sym_table[i].symbol, ptr) == 0)
                        operand2 = sym_table[i].addr;
        }
        if (operand2 == -1) {
                errno = -1;
                fprintf(stderr, "EQU error: not defime symbol %s", ptr);
        }
        sym_table[symline - 1].addr = operand1 - operand2;
                                        }
////////////////////////////////////////
//////////////'+'caculation/////////////
////////////////////////////////////////
                                        ptr = strchr(token_table[id]->operand[0], '+');
                                        if (ptr != NULL) {
        int operand1 = -1, operand2 = -1;
        ptr = strtok(token_table[id]->operand[0], "+");
        for (int i = symfid[prog_num]; i < symline; i++) {
                if (strcmp(sym_table[i].symbol, ptr) == 0)
                        operand1 = sym_table[i].addr;
        }
        if (operand1 == -1)
                errno = -1;
        ptr = strtok(NULL, "₩0");
        for (int i = symfid[prog_num]; i < symline; i++) {
                if (strcmp(sym_table[i].symbol, ptr) == 0)
                        operand2 = sym_table[i].addr;
        }
        if (operand2 == -1)
                errno = -1;
        sym_table[symline - 1].addr = operand1 - operand2;
                                        }
                                }//if not *
                        }//if EQU
                        else if (strcmp(token_table[id]->operator,"WORD") == 0) {
                                locctr += 3;
                        }
                        else if (strcmp(token_table[id]->operator,"BYTE") == 0) {
```

```
                                int length = 0;
                                for (; token_table[id]->operand[0][length] != '₩0'; length++);
                                length -= 3;
                                if (token_table[id]->operand[0][0] == 'X')
                                        locctr += (length + 1) % 2;
                                else if (token_table[id]->operand[0][0] == 'C')
                                        locctr += length;
                        }
                        else if (strcmp(token_table[id]->operator, "LTORG") == 0) {
                                for (int i = litfid[litid]; i < litline; i++) {
                                        literal_table[i].addr = locctr;
                                        int length = 0;
                                        for (; literal_table[i].literal[length] != '₩0'; length++);
                                        length -= 4;
                                        if (literal_table[i].literal[1] == 'C')
                                                locctr += length;
                                        else if (literal_table[i].literal[1] == 'X')
                                                locctr += (length + 1) / 2;
                                }
                                litfid[++litid] = litline;
                        }
                        else if (strcmp(token_table[id]->operator, "CSECT") == 0) {

                        }
                        else {//
                                if ((opid = search_opcode(token_table[id]->operator)) == -1) {
                                        errno = -1;
                                }
                                else{
                                        locctr += inst_table[opid]->format;
                                }
                        }
                }//if (strcmp(token_table[id]->label,".") != 0)
                else {

                }
                if (errno == -1)
                        break;

                token_parsing(input_data[id+1]);
        }//for not end
        ////////////////////////////////
        /////flush literal table////////
        ////////////////////////////////
        for (int i = litfid[litid]; i < litline; i++) {
                literal_table[i].addr = locctr;
                int length = 0;
                for (; literal_table[i].literal[length] != '₩0'; length++);
                length -= 4;
                if (literal_table[i].literal[1] == 'C')
                        locctr += length;
                else if (literal_table[i].literal[1] == 'X')
                        locctr += (length + 1) / 2;
        }
        prog_length[prog_num] = locctr - startadress;
        /* input_data의 문자열을 한줄씩 입력 받아서
         * token_parsing()을 호출하여 token_unit에 저장
         */
```

```
        return errno;
}

/                                                                    *
-------------------------------------------------------------------
-------------------------------------------------------------------
------
* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
*            여기서 출력되는 내용은 명령어 옆에 OPCODE가 기록된 표(과제 3번) 이다.
* 매게 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프로그램의 결과를 표준출력으로 보내어
*         화면에 출력해준다.
*          또한 과제 4번에서만 쓰이는 함수이므로 이후의 프로젝트에서는 사용되지 않는다.
*
-------------------------------------------------------------------
-------------------------------------------------------------------
-------
*/
// 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.
* 매게 : 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 */
        int symline = 0;
        int symid = 0;
        FILE *fp;
        if (file_name == NULL) {
                fp = stdout;
                fprintf(fp, "symtab");
        }
        else
                fp = fopen(file_name, "w");



        for (int i = 0; sym_table[i].symbol != NULL; i++) {
                if (strcmp(sym_table[i].symbol, "FIRST") != 0 && sym_table[i].addr == 0)
                        fprintf(fp, "₩n");
                fprintf(fp,        "%s₩t₩t%X₩n", sym_table[i].symbol, sym_table[i].addr);
        }
}

/                                                                    *
-------------------------------------------------------------------
-------------------------------------------------------------------
------
* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.
*         여기서 출력되는 내용은 LITERAL별 주소값이 저장된 TABLE이다.
* 매게 : 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) {
		fp = stdout;
		fprintf(fp, "₩nliteraltab₩n");
	}
	else
		fp = fopen(file_name, "w");
	for (int i = 0; literal_table[i].literal != NULL; i++) {
		char buf[10];
		int length = strlen(literal_table[i].literal);

		for (int k = 0; k < length - 4; k++)
			buf[k] = literal_table[i].literal[k + 3];
		buf[length - 4] = '₩0';
		fprintf(fp, "%s₩t₩t%X₩n", buf, literal_table[i].addr);
	}
}

/*
-------------------------------------------------------------------------
* 설명 : 어셈블리 코드를 기계어 코드로 바꾸기 위한 패스2 과정을 수행하는 함수이다.
*		패스 2에서는 프로그램을 기계어로 바꾸는 작업은 라인 단위로 수행된다.
*		다음과 같은 작업이 수행되어진다.
*		1. 실제로 해당 어셈블리 명령어를 기계어로 바꾸는 작업을 수행한다.
* 매개 : 없음
* 반환 : 정상종료 = 0, 에러발생 = < 0
* 주의 :
*
-------------------------------------------------------------------------
*/

static int assem_pass2(void)
{
	/* add your code here */
	int opid = 0;
	int obid = 0;
	int symid = 0;
	int litline = 0;
	int litid = 0;
	int prog_num = 0;
	int mline = 1;
	int obloc = 0;
	//////////////////////////
	/////input first line//////
	//////////////////////////
	if (strcmp(token_table[0]->operator,"START") == 0)
		locctr = atoi(token_table[0]->operand[0]);


	for (token_line = 1; strcmp(token_table[token_line]->operator,"END"); token_line++) {
		if (errno < 0)
			break;
		if (strcmp(token_table[token_line]->label, ".") != 0) {
			if (strcmp(token_table[token_line]->operator,"RESW") == 0) {

				locctr += 3 * atoi(token_table[token_line]->operand[0]);
			}
			else if (strcmp(token_table[token_line]->operator,"RESB") == 0) {

				locctr += atoi(token_table[token_line]->operand[0]);
			}
			else if ((opid = search_opcode(token_table[token_line]->operator)) >= 0) {
```

```
                                locctr += inst_table[opid]->format;

//////////////////////////////////////
                                ///////////object code generation////////
//////////////////////////////////////
                                if (inst_table[opid]->format == 1) {
ob_table[obid].ob_code = (inst_table[opid]->op / 16);
                                }
//////////////////////////////////////
/////////////format 2/////////////////
//////////////////////////////////////
                                else if (inst_table[opid]->format == 2) {
ob_table[obid].ob_code += (inst_table[opid]->op) << 8;//! ! _ _
                                        if (token_table[token_line]->operand[0][0] != '₩0') {
        int i;
        for (i = 0; i < 9; i++) {
                if (strcmp(token_table[token_line]->operand[0], register_table[i].name) == 0) {
                        ob_table[obid].ob_code += register_table[i].num << 4;
                        break;
                }
        }
        if (i == 9) {
                fprintf(stderr, "wrong register name: %s₩n", token_table[token_line]->operand[0]);
                errno = -1;
        }
        if (token_table[token_line]->operand[1][0] != '₩0') {
                for (i = 0; i < 9; i++) {
                        if (strcmp(token_table[token_line]->operand[1], register_table[i].name) == 0) {
ob_table[obid].ob_code += register_table[i].num;
                                break;
                        }
                }
                if (i == 9) {
                        fprintf(stderr, "wrong register name: %s₩n", token_table[token_line]->operand[0]);
                        errno = -1;
                }
        }
                                }
ob_table[obid].length = 2;
                                obloc
```

```
+= ob_table[obid].length;
                                    }//if format=2

//////////////////////////////////////

//////////////format 3//////////////////

//////////////////////////////////////
                                    else if (inst_table[opid]->format == 3) {
                                        int disp = 0;

ob_table[obid].ob_code += (inst_table[opid]->op / 16) << 20;//! _ _ _ _ _

ob_table[obid].ob_code += ((inst_table[opid]->op % 16) + (token_table[token_line]->nixbpe >> 4)) << 16;//_ ! _ _ _ _

ob_table[obid].ob_code += (token_table[token_line]->nixbpe % 16) << 12;//_ _ ! _ _ _
                                        if (token_table[token_line]->operand[0][0] != '₩0') {

          if ((token_table[token_line]->nixbpe >> 4) == 3) {//SIC/XE

                    if (token_table[token_line]->operand[0][0] == '=') {


//////////////////////////////

////////////literal/////////////


//////////////////////////////

                                    if ((litline = search_literal(token_table[token_line]->operand[0], litfid[litid])) < 0) {

                                        fprintf(stderr, "non-existent literal : %s₩n", token_table[token_line]->operand[0]);

                                        errno = -1;

                                    }

                                    disp = literal_table[litline].addr - locctr;

                    }

                    ////////////////////////////////

                    else {

                                    if ((symid = search_symbol(token_table[token_line]->operand[0], symfid[prog_num])) < 0) {


                                        fprintf(stderr, "non-existent symbol : %s₩n", token_table[token_line]->operand[0]);

                                        errno = -1;


                                    }

                                    else {

                                        disp = sym_table[symid].addr - locctr;


                                    }

                    }

          }
```

```
        else if ((token_table[token_line]->nixbpe >> 4) == 2) {//indirect adressing
                if ((symid = search_symbol(token_table[token_line]->operand[0] + 1, symfid[prog_num])) < 0) {

                        fprintf(stderr, "non-existent symbol : %s₩n", token_table[token_line]->operand[0]);
                        errno = -1;

                }
                else {
                        disp = sym_table[symid].addr - locctr;

                }
        }
        ////////////////////////////////////////
        ///////////immediate adressing/////////
        ////////////////////////////////////////
        else if ((token_table[token_line]->nixbpe >> 4) == 1) {
                if (isalpha(token_table[token_line]->operand[0][1])) {//operand=symbol
                        if ((symid = search_symbol(token_table[token_line]->operand[0] + 1, symfid[prog_num])) < -1) {
                                fprintf(stderr, "non-existent symbol : %s₩n", token_table[token_line]->operand[0]);
                                errno = -1;
                        }
                        else {
                                disp = sym_table[symid].addr - locctr;
                        }
                }
                else {
                        disp = atoi(token_table[token_line]->operand[0] + 1);
                }

        }
        ////////////////////////////////////////
        ob_table[obid].ob_code += disp;
        if (disp < 0)
                ob_table[obid].ob_code += 1 << 12;
                        }

ob_table[obid].length = 3;
                        obloc += ob_table[obid].length;
                }//if format=3
////////////////////////////////////////
///////////////format 4/////////////////
////////////////////////////////////////
```

```
                                else if (inst_table[opid]->format == 4) {
                                    int disp = 0;

ob_table[obid].ob_code += (inst_table[opid]->op / 16) << 28;//! _ _ _ _ _ _ _

ob_table[obid].ob_code += ((inst_table[opid]->op % 16) + (token_table[token_line]->nixbpe >> 4)) << 24;//_ ! _ _ _ _ _ _

ob_table[obid].ob_code += (token_table[token_line]->nixbpe % 16) << 20;//_ _ ! _ _ _ _ _

                                    if ((symid = search_symbol(token_table[token_line]->operand[0], symfid[prog_num])) < 0) {

        //EXTREF에서 찾기구현

        char temp[30] = { 0 };

        memcpy(temp, extref[prog_num], strlen(extref[prog_num]) + 1);

        char *ptr = strtok(temp, ",");

        while (ptr != NULL) {



            if (strcmp(ptr, token_table[token_line]->operand[0]) == 0)

                break;

            ptr = strtok(NULL, ",");

        }

        if (ptr == NULL) {

            fprintf(stderr, "non-existent symbol : %s₩n", token_table[token_line]->operand[0]);

            errno = -1;

        }

        else

            sprintf(m_record[mline++], "M%06X05+%s", obloc + 1, token_table[token_line]->operand[0]);
                                    }
                                    else {

        disp = sym_table[symid].addr;
                                    }

ob_table[obid].ob_code += disp;
                                    if (disp < 0)

        ob_table[obid].ob_code += 1 << 20;

ob_table[obid].length = 4;
                                    obloc += ob_table[obid].length;
                                }

ob_table[obid].pc = locctr;
                                obid++;
                            }//if opcode
                            else if (strcmp(token_table[token_line]->operator,"WORD") == 0) {
                                locctr += 3;
                                if (isdigit(token_table[token_line]->operand[0][0]))

ob_table[obid].ob_code = atoi(token_table[token_line]->operand[0]);
                                else {

char*ptr1, *ptr2, *ptr3;

////////////////////////////////////
```

```
//////////////'-'caculation/////////////

//////////////////////////////////////
                                        ptr1 = strchr(token_table[token_line]->operand[0], '-');//'-'인지 확인
                                        i f (ptr1 != NULL) {

        int operand1 = -1, operand2 = -1;

        ptr1                                    = strtok(token_table[token_line]->operand[0], "-");

        ptr3 = strtok(NULL, "₩0");

        operand1      =      search_symbol(ptr1, symfid[prog_num]);

        if (operand1 == -1) {

                //EXTREF에서 찾기 구현

                char temp[30] = { 0 };

                m e m c p y ( t e m p , extref[prog_num], strlen(extref[prog_num]) + 1);

                ptr2 = strtok(temp, ",");

                while (ptr2 != NULL) {

                        if (strcmp(ptr2, ptr1) == 0)

                                break;

                        ptr2 = strtok(NULL, ",");

                }

                if (ptr2 == NULL) {

                        f p r i n t f ( s t d e r r , "non-existent symbol : %s₩n", token_table[token_line]->operand[0]);

                        errno = -1;

                }

                else

sprintf(m_record[mline++], "M%06X06+%s", obloc, ptr1);

        }

        operand2      =      search_symbol(ptr3, symfid[prog_num]);

        if (operand2 == -1) {

                //EXTREF에서 찾기 구현

                char temp[30] = { 0 };

                m e m c p y ( t e m p , extref[prog_num], strlen(extref[prog_num]) + 1);

                ptr2 = strtok(temp, ",");

                while (ptr2 != NULL) {

                        if (strcmp(ptr2, ptr3) == 0)

                                break;

                        ptr2 = strtok(NULL, ",");

                }
```

```
                if (ptr2 == NULL) {

                        fprintf(stderr, "non-existent symbol : %s₩n", token_table[token_line]->operand[0]);

                        errno = -1;

                }

                else

sprintf(m_record[mline++], "M%06X06-%s", obloc, ptr3);

        }

        //EXTREF가 아닌 경우는 EQU로 계산
                                        }

////////////////////////////////////////

//////////////'+'caculation/////////////

////////////////////////////////////////
                                        ptr1 = strchr(token_table[token_line]->operand[0], '+');//'-'인지 확인
                                        if (ptr1 != NULL) {

        int operand1 = -1, operand2 = -1;

        ptr1 = strtok(token_table[token_line]->operand[0], "+");

        ptr3 = strtok(NULL, "₩0");

        operand1 = search_symbol(ptr1, symfid[prog_num]);

        if (operand1 == -1) {

                //EXTREF에서 찾기 구현

                char temp[30] = { 0 };

                memcpy(temp, extref[prog_num], strlen(extref[prog_num]) + 1);

                ptr2 = strtok(temp, ",");

                while (ptr2 != NULL) {

                        if (strcmp(ptr2, ptr1) == 0)

                                break;

                        ptr2 = strtok(NULL, ",");

                }

                if (ptr2 == NULL) {

                        fprintf(stderr, "non-existent symbol : %s₩n", token_table[token_line]->operand[0]);

                        errno = -1;

                }

                else

sprintf(m_record[mline++], "M%06X06+%s", obloc, ptr1);

        }
```

```
operand2 = search_symbol(ptr3, symfid[prog_num]);

if (operand2 == -1) {

//EXTREF에서 찾기 구현

char temp[30] = { 0 };

memcpy(temp, extref[prog_num], strlen(extref[prog_num]) + 1);

ptr2 = strtok(temp, ",");

while (ptr2 != NULL) {

if (strcmp(ptr2, ptr3) == 0)

break;

ptr2 = strtok(NULL, ",");

}

if (ptr2 == NULL) {

fprintf(stderr, "non-existent symbol : %s₩n", token_table[token_line]->operand[0]);

errno = -1;

}

else

sprintf(m_record[mline++], "M%06X06+%s", obloc, ptr3);

}

//EXTREF가 아닌 경우는 EQU로 계산

}

}

ob_table[obid].length =3;

obloc += ob_table[obid].length;

ob_table[obid].pc = locctr;

obid++;

}//if word

else if (strcmp(token_table[token_line]->operator,"BYTE") == 0) {

int length =strlen(token_table[token_line]->operand[0]) - 3;

if (token_table[token_line]->operand[0][0] == 'X') {

locctr += (length + 1) % 2;

ob_table[obid].length = (length + 1) % 2;

obloc += ob_table[obid].length;

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

if (isalpha(token_table[token_line]->operand[0][i + 2]))

ob_table[obid].ob_code += (int)(token_table[token_line]->operand[0][i + 2] - 0X37) << 4 * (length - 1 - i);

else

ob_table[obid].ob_code += (int)(token_table[token_line]->operand[0][i + 2] - 0X30) << 4 * (length - 1 - i);

}

}

else if
```

```
(token_table[token_line]->operand[0][0] == 'C')
{
                                        locctr
+= length;
ob_table[obid].length = length;
                                        obloc
+= ob_table[obid].length;
                                        for
(int i = 0; i < length; i++) {
        if                                      f
(isalpha(token_table[token_line]->operand[0][i
+ 2]))
                ob_table[obid].ob_code      +=
(int)(token_table[token_line]->operand[0][i    +
2]) << 8 * i;
        else
                ob_table[obid].ob_code      +=
(int)(token_table[token_line]->operand[0][i    +
2]) << 8 * i;
                                        }
                                    }
ob_table[obid].pc = locctr;
                                    obid++;
                            }//if operator = BYTE
                            else                if
(strcmp(token_table[token_line]->operator,"CSEC
T") == 0) {
                                    locctr = 0;
                                    prog_num++;
                                    mline++;
                                    obloc = 0;
                            }
                            else                if
(strcmp(token_table[token_line]->operator,"LTO
RG") == 0) {
                                    for (int i =
litfid[litid]; i < litfid[litid + 1]; i++) {
                                        i n t
length = strlen(literal_table[i].literal) - 4;
                                        i      f
(literal_table[i].literal[1] == 'X') {
        locctr += (length + 1) % 2;
        ob_table[obid].length = (length + 1) %
2;
        obloc += ob_table[obid].length;
        for (int k = 0; k < length; k++) {
                i                               f
(isalpha(literal_table[i].literal[k + 3]))
                        ob_table[obid].ob_code
+= (int)(literal_table[i].literal[k + 3] - 0X37)
<< 4 * (length - 1 - k);
                else
                        ob_table[obid].ob_code
+= (int)(literal_table[i].literal[k + 3] - 0X30)
<< 4 * (length - 1 - k);
        }
                                    }
                                    else if
(literal_table[i].literal[1] == 'C') {
        locctr += length;
        ob_table[obid].length = length;
        obloc += ob_table[obid].length;
        for (int k = 0; k < length; k++) {
                i                               f
(isalpha(literal_table[i].literal[k + 3]))
                        ob_table[obid].ob_code
+= (int)(literal_table[i].literal[k + 3]) << 8 *
(length - 1 - k);
                else
```

```
                        ob_table[obid].ob_code
+= (int)(literal_table[i].literal[k + 3]) << 8 *
(length - 1 - k);

        }
                                        }
                                }
                                litid++;

ob_table[obid].pc = locctr;
                                obid++;
                        }//if LOORG
                }//if not comment
        }//for not end
        ////////////////////////////////
        ////////flush literal table/////////
        ////////////////////////////////
        for (int i = litfid[litid];
literal_table[i].literal != NULL; i++) {
                int                     length
=strlen(literal_table[i].literal) - 4;
                if (literal_table[i].literal[1] ==
'X') {
                        locctr += (length + 1)
% 2;
                        ob_table[obid].length =
(length + 1) % 2;
                        obloc           +=
ob_table[obid].length;
                        for (int k = 0; k <
length; k++) {
                                i       f
(isalpha(literal_table[i].literal[k + 3]))

ob_table[obid].ob_code                  +=
(int)(literal_table[i].literal[k + 3] - 0X37) << 4
* (length - 1 - k);
                                else

ob_table[obid].ob_code                  +=
(int)(literal_table[i].literal[k + 3] - 0X30) << 4
* (length - 1 - k);
                        }
                }
                else if (literal_table[i].literal[1]
== 'C') {
                        locctr += length;
                        ob_table[obid].length =
length;
                        obloc           +=
ob_table[obid].length;
                        for (int k = 0; k <
length; k++) {
                                i       f
(isalpha(literal_table[i].literal[k + 3]))

ob_table[obid].ob_code                  +=
(int)(literal_table[i].literal[k + 3]) << 8 * k;
                                else

ob_table[obid].ob_code                  +=
(int)(literal_table[i].literal[k + 3]) << 8 * k;
                        }
                }
                ob_table[obid].pc = locctr;
                obid++;
        }//end flush

        return errno;
}

/                                               *
------------------------------------
------------------------------------
------
* 설명 : 입력된 문자열의 이름을 가진 파일에 프로
그램의 결과를 저장하는 함수이다.
*       여기서 출력되는 내용은 object code (프
로젝트 1번) 이다.
* 매계 : 생성할 오브젝트 파일명
* 반환 : 없음
* 주의 : 만약 인자로 NULL값이 들어온다면 프로그
램의 결과를 표준출력으로 보내어
*       화면에 출력해준다.
*
*
------------------------------------
------------------------------------
-------
*/
void make_objectcode_output(char *file_name)
{
```

```
        /* add your code here */
        FILE*fp;
        char buf[71];
        int prognum = 0;
        int buf_length = 0;
        int obid = 0;
        int mline = 0;
        if (file_name == NULL)
                fp = stdout;
        else
                fp = fopen(file_name, "w");
        ////////////////////////////////
        ////////first section H//////////
        ////////////////////////////////
        sprintf(buf, "H%-6s%06X%06X₩0",
token_table[0]->label, 0,
prog_length[prognum]);
        fprintf(fp, "%s₩n", buf);
        if (extdef[prognum] != NULL)
                write_def(fp, prognum);
        if (extref[prognum] != NULL)
                write_ref(fp, prognum);
        ////////////////////////////////
        /******T_ _ _ _ _ _ _ _ ~~******/
        ////////////////////////////////
        buf[0] = 'T';
        buf[1] = '₩0';
        sprintf(buf, "%s%06X₩0", buf,
atoi(token_table[0]->operand[0]));
        buf[8] = '₩0';
        buf[9] = '₩0';
        ////////////////////////////////
        for (token_line = 1;
ob_table[obid].length != 0; token_line++) {
                i f
(strcmp(token_table[token_line]->label, ".") != 0)
{
                        i f
(strcmp(token_table[token_line]->operator,"CSEC
T") == 0) {

////////////////////////
                                ////////flush
buf//////////

////////////////////////
                                if (buf_length
> 0) {

sprintf(buf, "%s%02X%s", buf, buf_length, buf +
9);

fprintf(fp, "%s₩n", buf);
                                }

////////////////////////////
                                ///////////M
RECORD////////////

////////////////////////////
                                w h i l e
(m_record[++mline][0] != '₩0')

fprintf(fp, "%s₩n", m_record[mline]);

//////////////////////////
                                if (prognum
== 0)

sprintf(buf, "E%06X",
atoi(token_table[0]->operand[0]));
                                else

sprintf(buf, "E");
                                f p r i n t f ( f p ,
"%s₩n", buf);




////////////////////////////
                                //////new
section program//////

////////////////////////////
                                s p r i n t f ( b u f ,
" H % - 6 s % 0 6 X % . 6 X ₩ 0 " ,
token_table[token_line]->label, 0,
prog_length[++prognum]);
                                f p r i n t f ( f p ,
"₩n%s₩n", buf);
```

```
initial_bufT(buf, obid, &buf_length);
                                    i          f
(extdef[prognum] != NULL)

write_def(fp, prognum);
                                    i          f
(extref[prognum] != NULL)

write_ref(fp, prognum);

                          }//if CSECT
                          else          if
(strcmp(token_table[token_line]->operator,"EXT
DEF") == 0) {

                          }
                          else          if
(strcmp(token_table[token_line]->operator,"EXT
REF") == 0) {

                          }
                          else          if
(strcmp(token_table[token_line]->operator,"RES
W") == 0) {
                                    if   (buf_length
> 0) {

sprintf(buf, "%s%02X%s", buf, buf_length, buf +
9);

fprintf(fp, "%s₩n", buf);

initial_bufT(buf, obid, &buf_length);
                                    }
                          }
                          else          if
(strcmp(token_table[token_line]->operator,"RESB
") == 0) {
                                    if   (buf_length
> 0) {

sprintf(buf, "%s%02X%s", buf, buf_length, buf +
9);

fprintf(fp, "%s₩n", buf);

initial_bufT(buf, obid, &buf_length);
                                    }
                          }
                          else          if
(strcmp(token_table[token_line]->operator,"EQU"
) == 0) {
                                    if   (buf_length
> 0) {

sprintf(buf, "%s%02X%s", buf, buf_length, buf +
9);

fprintf(fp, "%s₩n", buf);

initial_bufT(buf, obid, &buf_length);
                                    }
                          }
                          else {//write obcode
                                    int   length   =
ob_table[obid].length;
                                    int   object   =
ob_table[obid].ob_code;

                                    if   (buf_length
+ length > 30) {//over max objectcode line

sprintf(buf, "%s%02X%s", buf, buf_length, buf +
9);

fprintf(fp, "%s₩n", buf);

initial_bufT(buf, obid, &buf_length);
                                    }
                                    buf_length   +=
length;
                                    switch (length)
{
                                    case 1:

sprintf(buf   +   9,   "%s%02X₩0",   buf   +   9,
ob_table[obid].ob_code);
                                              break;
                                    case 2:

sprintf(buf   +   9,   "%s%04X₩0",   buf   +   9,
ob_table[obid].ob_code);
```

```
                                        break;
                                case 3:

sprintf(buf + 9, "%s%06X₩0", buf + 9,
ob_table[obid].ob_code);
                                        break;
                                case 4:

sprintf(buf + 9, "%s%08X₩0", buf + 9,
ob_table[obid].ob_code);
                                        break;
                                default:
                                        break;
                                }
                                obid++;
                        }/**///write      obcode
end
                }//if not comment
        }//for not END
        ////////////////////////
        ////////flush buf//////////
        ////////////////////////
        if (buf_length > 0) {
                sprintf(buf,  "%s%02X%s",  buf,
buf_length, buf + 9);
                fprintf(fp, "%s₩n", buf);
        }
        ////////////////////////////
        ///////////M RECORD///////////
        ////////////////////////////
        while (m_record[++mline][0] != '₩0')
                fprintf(fp,                "%s₩n",
m_record[mline]);
        ////////////////////////////
        if (prognum == 0)
                sprintf(buf,          "E%06X₩n",
atoi(token_table[0]->operand[0]));
        else
                sprintf(buf, "E₩n");
        fprintf(fp, "%s₩n", buf);

        fclose(fp);
}

/                                               *
-------------------------------------
------
 * 설명 : 입력 문자열이 심볼인지 검사하는 함수이
다.
 * 매계 : 토큰 단위로 구분된 문자열
 * 반환 : 정상종료 = 심볼 테이블 인덱스, 에러 <
0
 * 주의 :
*
*
-------------------------------------
-------------------------------------
-------
*/
int search_symbol(char*str,int symfid) {
        for (int i = symfid; sym_table[i].symbol
!= NULL; i++) {
                i                       f
(i!=symfid&&sym_table[i].addr==0&&strcmp(sym
_table[i].symbol,"FIRST")!= 0)
                        break;

if(strcmp(sym_table[i].symbol,str)==0)
                        return i;
        }

        return -1;
}

/                                               *
-------------------------------------
-------------------------------------
------
 * 설명 : 입력 문자열이 리터럴인지 검사하는 함수
이다.
 * 매계 : 토큰 단위로 구분된 문자열
 * 반환 : 정상종료 = 심볼 테이블 인덱스, 에러 <
0
 * 주의 :
*
*
-------------------------------------
-------------------------------------
-------
*/
int search_literal(char*str, int litfid) {
```

```
        for (int i = litfid; literal_table[i].literal
!= NULL; i++) {
                if (strcmp(literal_table[i].literal,
str) == 0)
                        return i;
        }

        return -1;
}
/                                                    *
---------------------------------------
---------------------------------------
------
 * 설명 : Define Record를 object program에 작
성하는 함수이다.
 * 매계 : object program을 작성할 file pointer,
program section number
 * 반환 : 없음
 * 주의 :
*
*
---------------------------------------
---------------------------------------
-------
*/
void write_def(FILE *fp, int prognum) {
        char buf[100] = { '₩0' };
        int symid = 0;
        buf[0] = 'D';
        char*ptr= strtok(extdef[prognum], ",");
        sprintf(buf, "%s%-6s", buf, ptr);
        symid          =          search_symbol(ptr,
symfid[prognum]);
        sprintf(buf,          "%s%06X",          buf,
sym_table[symid].addr);
        while ((ptr = strtok(NULL, ",")) !=
NULL) {
                strcat(buf, ptr);
                symid = search_symbol(ptr,
symfid[prognum]);
                sprintf(buf, "%s%06X", buf,
sym_table[symid].addr);
        }
        fprintf(fp, "%s₩n", buf);
}
/                                                    *
---------------------------------------
---------------------------------------
------
 * 설명 : Refer Record를 object program에 작성
하는 함수이다.
 * 매계 : object program을 작성할 file pointer,
program section number
 * 반환 : 없음
 * 주의 :
*
*
---------------------------------------
---------------------------------------
-------
*/
void write_ref(FILE *fp, int prognum) {
        char buf[100] = { '₩0' };
        int symid = 0;
        buf[0] = 'R';
        char*ptr = strtok(extref[prognum], ",");
        sprintf(buf, "%s%-6s", buf, ptr);
        while ((ptr = strtok(NULL, ",")) !=
NULL)
                strcat(buf, ptr);

        fprintf(fp, "%s₩n", buf);
}
/                                                    *
---------------------------------------
---------------------------------------
------
 * 설명 : Text Record를 작성하는데 사용될
buffer를 초기화한다.
 * 매계 : object program에 작성할 buffer, 첫번째
object code index, 현재 buffer에 있는 obcode의
length
 * 반환 : 없음
 * 주의 :
*
*
---------------------------------------
---------------------------------------
-------
*/
void initial_bufT(char* buf, int obid, int
```

```
*buf_length) {
        sprintf(buf,                    "T%06X₩0",
ob_table[obid].pc - ob_table[obid].length);
        buf[8] = '₩0';
        buf[9] = '₩0';
        *buf_length = 0;
}
```