과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

<<Project 3>>

**서강대학교 수학과**

**[학번] 20121277**

**[이름] 김주호**

[1. 프로그램 개요 3](#_Toc71025869)

[2. 프로그램 설명 3](#_Toc71025870)

[2.1 프로그램 흐름도 3](#_Toc71025871)

[3. 모듈 정의 5](#_Toc71025872)

[3.1 모듈이름 : set\_PROGADDR() 5](#_Toc71025873)

[3.2 모듈이름 : loader() 5](#_Toc71025874)

[3.3 모듈이름 : load\_pass1() 5](#_Toc71025875)

[3.4 모듈이름 : load\_pass2() 6](#_Toc71025876)

[3.5 모듈이름 : push\_to\_ESTAB() 8](#_Toc71025877)

[3.6 모듈이름 : find\_ESNODE\_or\_NULL() 9](#_Toc71025878)

[3.7 모듈이름 : free\_ESTAB() 9](#_Toc71025879)

[3.8 모듈이름 : bp\_command() 9](#_Toc71025880)

[3.9 모듈이름 : run() 10](#_Toc71025881)

[3.10 모듈이름 : execute\_instruction() 10](#_Toc71025882)

[3.11 모듈이름 : LD\_related\_instruction() 12](#_Toc71025883)

[3.12 모듈이름 : J\_related\_instruction() 12](#_Toc71025884)

[3.13 모듈이름 : ST\_related\_instruction() 13](#_Toc71025885)

[3.14 모듈이름 : twos\_complement\_str\_to\_decint() 13](#_Toc71025886)

[3.15 모듈이름 : get\_opcode\_or\_NULL\_by\_opcode() 14](#_Toc71025887)

[4. 전역 변수 정의 14](#_Toc71025888)

[4.1 int ES\_NODE \*ESTAB[ESTAB\_HASH\_SIZE] 14](#_Toc71025889)

[4.2 int BP\_CHK[MEM\_SIZE] 14](#_Toc71025890)

[4.3 int BP\_ADDR[MEM\_SIZE] 14](#_Toc71025891)

[4.4 int PRINT\_FLAG 15](#_Toc71025892)

[4.5 int NUM\_OF\_BP 15](#_Toc71025893)

[4.6 int REG[REG\_NUM + 1] 15](#_Toc71025894)

[4.7 char CC; 15](#_Toc71025895)

[4.8 int PROGRAM\_ADDR 15](#_Toc71025896)

[4.9 int CS\_ADDR 15](#_Toc71025897)

[4.10 int CS\_LEN 15](#_Toc71025898)

[4.11 int PROGRAM\_LEN 15](#_Toc71025899)

[4.12 int FIRST\_INSTRUCTION\_ADDR 15](#_Toc71025900)

[5. 코드 16](#_Toc71025901)

[5.1 20121277.h 16](#_Toc71025902)

[5.2 20121277.c 17](#_Toc71025903)

[5.3 assembler\_commands.h 23](#_Toc71025904)

[5.4 assembler\_commands.c 25](#_Toc71025905)

[5.5 memory\_commands.h 43](#_Toc71025906)

[5.6 memory\_commands.c 43](#_Toc71025907)

[5.7 shell\_commands.h 48](#_Toc71025908)

[5.8 shell\_commands.c 49](#_Toc71025909)

[5.9 utils.h 52](#_Toc71025910)

[5.10 utils.c 54](#_Toc71025911)

[5.11 opcode\_table\_commands.h 61](#_Toc71025912)

[5.12 opcode\_table\_commands.c 62](#_Toc71025913)

[5.13 loader\_commands.h 66](#_Toc71025914)

[5.14 loader\_commands.c 67](#_Toc71025915)

# 1. 프로그램 개요

프로젝트 1, 2에서 구현한 셸에 linking과 loading 기능을 추가하는 프로그램이다. 프로젝트 2에서 구현한 assemble 명령을 통해서 생성된 object 파일을 link 시켜 메모리에 올리는 일을 수행한다. 이를 위해 4가지 기능을 구현하는데 첫째로 주소를 지정하는 명령인 progaddr과 linking loader 명령인 loader, 프로그램 실행 명령인 run, debug 명령인 bp이다.

# 2. 프로그램 설명

## 2.1 프로그램 흐름도

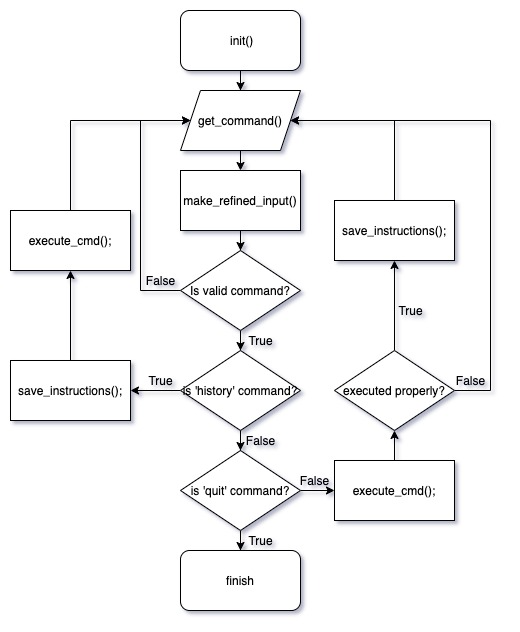


그림1> 프로그램 흐름도

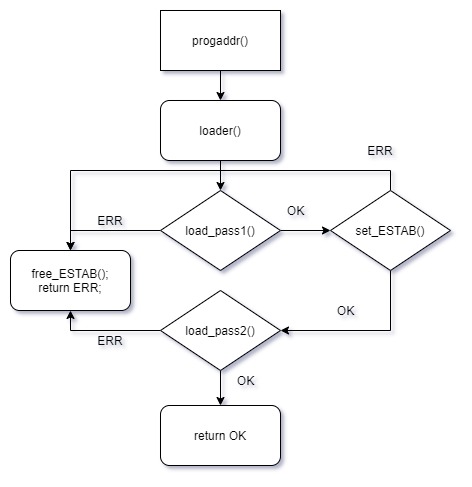


그림2> link loader 흐름도

# 3. 모듈 정의

## 3.1 모듈이름 : set\_PROGADDR()

### 3.1.1 *기능*

사용자로부터 입력 받은 address로 program이 load 될 메모리 시작 주소 위치를 set한다.

### 3.1.2 *사용변수*

int addr\_int - 사용자가 입력한 hex str 주소를 int형으로 변환하여 저장한다.

## 3.2 모듈이름 : loader()

### 3.2.1 *기능*

pass1과 pass2 알고리즘을 call하여, ES table을 생성(pass1)하고, 이를 이용한 load 작업을 수행(pass2)한다. pass1, pass2 알고리즘을 수행한 뒤에는 register를 모두 0으로 초기화한다. L 레지스터의 값은 프로그램의 길이로 한다는 명세서의 요구사항을 따른다.

### 3.2.2 *사용변수*

FILE \*fp - 입력 받은 object file을 여는데 사용하는 파일 포인터 변수이다.

## 3.3 모듈이름 : load\_pass1()

### 3.3.1 *기능*

object file을 하나하나 열면서, H레코드와 D레코드를 참조하여, ES table을 생성한다. H레코드에는 프로그램의 이름, 프로그램의 시작 주소와 그 길이가 적혀있다. H레코드를 만나면, 프로그램의 이름과 해당 location을 ES table에 push한다. 이 때, 이 전에 open한 object file의 다음 메모리 주소부터 올라갈 수 있도록, 이전 프로그램의 길이 만큼 더하여, CS\_ADDR를 조정한다. 다음 D레코드를 만나게 되면, symbol들을 ES table에 push한다. CS\_ADDR을 더하여 메모리에 실제 올라가게 될 address를 저장하여야 한다. 이렇게 CS name 혹은 symbol name과 해당 주소가 mapping된 ES table이 완성되게 되고, 이는 load\_pass2() 알고리즘에서 활용된다.

### 3.3.2 *사용변수*

char \*ptr - object file로부터 line을 입력받게 되면, 이를 앞에서부터 탐색하면서, name 혹은 object code에 접근해야 한다. 문자열 line의 각 character를 탐색하는 char 포인터 변수이다.

char line[LINE\_LEN] - object file로부터 읽어들인 하나의 line을 저장하는 문자열 자료구조이다.

char CS\_name[SYMBOL\_LEN], char CS\_start\_addr\_hexstr[STR\_ADDR\_LEN], char int CS\_start\_addr\_int; - H레코드를 만나게 되었을 때, line을 탐색하면서, control section의 이름, 시작주소, 길이를 1차적으로 char 문자열로 얻어오고, 이 정보를 int 형으로 바꿔서 저장한다. 이를 위한 변수들이다.

char SYM\_name[SYMBOL\_LEN], char sym\_addr\_hexstr[STR\_ADDR\_LEN], int sym\_addr\_int - D레코드를 만나게 되었을 때, line을 탐색하면서, symbol의 이름, 저장될 메모리 주소(cs 시작 address에 상대적인)를 1차적으로 char 문자열로 얻어오고, 이 정보를 int 형으로 바꿔서 저장한다. 이를 위한 변수들이다.

## 3.4 모듈이름 : load\_pass2()

### 3.4.1 *기능*

pass1에서 완성한 ES table과 R, T, M 레코드를 참조하여 memory에 object 코드들을 적재한다.

* H 레코드를 만났을 때
* control section의 이름을 RFTAB[1]에 저장한다.
* R 레코드를 만났을 때
* R레코드에 reference number와 symbol name이 있다. RFTAB[reference number]에 symbol의 address를 저장해야 하는데, 이는 pass1 알고리즘에서 생성한 ES table을 탐색하여 찾는다.
* T 레코드를 만났을 때
* object code들을 메모리에 적재한다. T 레코드에 object code가 올라갈 MEM의 시작 주소(cs address에 상대적인)가 적혀있고, T 레코드의 1byte 씩 접근하여 메모리에 적재한다.
* M 레코드를 만났을 때
* M레코드에 modify를 적용하는 주소와 길이가 적혀있다. 해당 메모리 주소에 이미 machine code가 적재되어 있는데, 이를 다시 가져와서 M레코드가 지령하는 만큼의 조정을 가한 후, 다시 수정된 value를 메모리에 저장한다. 예를 들어, +02라고 되어있다면, RFTAB[2]에서 value를 얻어온 뒤 그 값만큼의 조정을 하는 것이다.

### 3.4.2 *사용변수*

char \*ptr - object file로부터 line을 입력받게 되면, 이를 앞에서부터 탐색하면서, name 혹은 object code에 접근해야 한다. 문자열 line의 각 character를 탐색하는 char 포인터 변수이다.

char line[LINE\_LEN] - object file로부터 읽어들인 하나의 line을 저장하는 문자열 자료구조이다.

int RFTAB[MAX\_RF\_NUM] - reference number가 index 역할을 하고, symbol의 address가 value로서 저장되어있는 int 배열이다. 결국 reference number와 address value를 mapping하고 있는 table로서 기능한다.

char rf\_ind\_hexstr[NAME\_LEN], int rf\_ind - object file에서 읽어들인 reference number를 1차적으로 문자열에 저장했다가, 다시 이를 int형으로 바꿔 저장한다.

char ES\_name[NAME\_LEN] - R 레코드에서 읽어들인 ES\_name을 저장하는 문자열

char one\_line\_T\_record\_len\_hexstr[STR\_ADDR\_LEN], int one\_line\_T\_record\_len\_int - T 레코드 한 line의 길이를 문자열로 읽어와서 이를 int값으로 바꿔 저장한다.

char starting\_addr\_hexstr[STR\_ADDR\_LEN] - object code가 올라갈 메모리의 시작주도를 저장하는 문자열 변수이다.

char byte\_hexstr[STR\_ADDR\_LEN], int byte\_int - T레코드에 적인 object code들에 대하여 1byte씩 접근하여 이를 메모리에 적재한다. 그 1 byte 값을 우선 문자열 형태로 저장한 뒤 이름 int 형으로 바꾸어 저장한다.

int LOC\_to\_push - 1 byte를 적재할 메모리의 주소를 관리하는 변수이다.

char LOC\_to\_be\_modified\_hexstr[STR\_ADDR\_LEN], int LOC\_to\_be\_modified\_int - modify를 해야하는 location의 주소 값을 나타내는 변수. 먼저 문자열로 해당 정보를 받았다가 이를 int 형 자료로 바꿔 저장한다.

char halfbytes\_len\_to\_be\_modified\_hexstr[STR\_ADDR\_LEN], int halfbytes\_len\_to\_be\_modified\_int - 수정될 half byte의 길이를 나타내는 변수이다. halfbytes\_len\_to\_be\_modified\_int d의 값은 5 또는 6이다.

char half\_bytes\_to\_be\_modified\_str[STR\_ADDR\_LEN] - 메모리가 가지고 있는 값 중에서 수정되어야할 값을 메모리에서 가져와서 문자열 형태로 저장한다.

char LOC\_to\_be\_modified\_hexstr[STR\_ADDR\_LEN], int LOC\_to\_be\_modified\_int - M레코드는 수정해야할 메모리 주소를 알려주게 되는데 이를 1차적으로 문자열에 저장하였다가, 후에 int형 자료로 변환하여 저장한다.

int dx - modify를 적용하는 증분(음수일 수 있음)이다. 다름 아닌 RFTAB[rf\_ind\_int]인데, 변화량이라는 어감이 느껴지도록 readability를 위해 dx라는 변수명을 사용하였다. target object code에 dx만큼의 조정을 가하게 된다.

## 3.5 모듈이름 : push\_to\_ESTAB()

### 3.5.1 *기능*

ES table에 es node를 push하는 작업을 수행한다. ES table은 hash size 20인 hash map으로 구현되어있으며, name과 address의 묶음을 하나의 정보단위로 가지고 있다. 2주차 project의 symbol table과 같은 형태로 구현되어있기 때문에, push 로직도 동일하다. hash function을 이용하여 index를 얻은뒤 이 index에 저장되어있는 linked list에 해당 es node를 삽입하면 된다.

### 3.5.2 *사용변수*

int ind - hash function을 이용하여 es node가 저장될 index를 구한다. 이를 저장하는 int형 변수이다.

## 3.6 모듈이름 : find\_ESNODE\_or\_NULL()

### 3.6.1 *기능*

es name을 입력받아 ES table을 탐색하여 이에 해당하는 node를 돌려준다. 해당 es name이 없다면, NULL 값을 return한다.

### 3.6.2 *사용변수*

int ind - hash function을 이용하여 es node가 저장된 index를 구한다. 이를 저장하는 int형 변수이다.

## 3.7 모듈이름 : free\_ESTAB()

### 3.7.1 *기능*

ESTAB에 할당된 메모리를 해제하는 함수이다.

### 3.7.2 *사용변수*

ES\_NODE \*cur\_node, \*pre\_node - cur\_node는 link를 타고 할당을 해제할 노드를 방문하기 위한 사용자정의 데이터타입 포인터 변수이고, pre\_node에 assign된 메모리를 해제한다.

## 3.8 모듈이름 : bp\_command()

### 3.8.1 *기능*

목적 : bp 관련한 사용자의 명령에 따라 관련 작업을 수행하고, 그 내용은 아래와 같다.

1. bp : bp가 걸린 메모리 주소를 보여준다.
2. bp clear : set 되어 있는 bp값들을 모두 해제한다.
3. bp [address] : 해당 address에 bp를 설정한다.

break point는 MEMORY[] 배열과 동일한 크기의 배열(전역변수 BP\_CHK[])을 생성하고, BP\_CHK[해당address] = true 로 표시함으로써 break point를 표시한다. 보다 자세한 설명은 아래 전역변수 BP\_CHK와 BP\_ADDR를 설명하는 란에서 한다.

### 3.8.2 *사용변수*

없음

## 3.9 모듈이름 : run()

### 3.9.1 *기능*

machine instruction을 수행시키며, break이 걸린 point들을 잡아내어 그 때마다 register의 내용을 print한다.

### 3.9.2 *사용변수*

END - PROGRAM\_ADDR에 CS\_LEN을 더한 값을 저장하며, 프로그램을 위해 할당된 메모리공간 주소의 마지막을 지정한다.

## 3.10 모듈이름 : execute\_instruction()

### 3.10.1 *기능*

메모리에 적재된 object code들을 메모리에서 읽어들여 machine instruction을 수행시킨다. 구체적인 logic은 다음과 같다.

1. 메모리에서 1byte를 읽어온다
2. & 0b00000011을 수행하여 8bit 중 뒤의 2bit에 자리한 n, i 값을 얻는다.
3. 메모리에서 읽어온 값에서 ni값을 빼면 opcode가 무엇인지 알게된다.
4. opcode를 이용하여, op node를 얻는다.
5. op node에 기록된 format을 살필 수 있게 되고, 이로써 메모리에서 몇 byte를 가져와야하는지 알 수 있게 된다.
6. format에 맞도록 메모리에서 데이터를 가져와서 operand를 설정한다.
   1. format이 3이고 operand가 2의 보수표현 음수인 경우, hex str으로 표현된 메모리의 값을 그대로 decimal int로 변환하게 되면 기본적으로 양수로 변환되므로 이를 보정해야한다.
   2. format이 3 또는 4라면, b, p, x의 값을 추가적으로 확인하여, disp를 한번 더 조정한다.
7. operation을 크게 LD 관련 명령어, ST 관련 명령어, J 관련 명령어가 있다고 분류하고, 각각 해당하는 subroutine을 불러 해당 operation을 수행한다.

project 3에서 I/O 관련 3가지 명령을 다음과 같이 처리한다.

* RD 명령 : RD는 항상 input device로부터 아무것도 받지 못한다는 가정을하고, NULL 값(ascii == 0)을 가져오는 방식으로 동작하도록 설정한다
* TD 명령 : CC(Condition Code)를 '<'로 set하고, 바로 다음 instuction으로 넘어간다고 가정한다.
* WD 명령 : 바로 다음 instruction으로 넘어간다고 가정한다.

### 3.10.2 *사용변수*

int opcode - opcode의 값을 담는 변수이다.

int format - op 명령의 format을 저장하는 변수이다.

int reg1, reg2 - 레지스터 번호를 저장하는 변수이다.

int disp - 3형식 insturcion의 하위 1.5 byte에 적힌 disp이다. int형으로 저장한다.

int ni, x, b, p, e - 각각 n, i, x, b, p, e bit를 나타낸다. 해당 변수들의 정보를 바탕으로 addressing 방법을 결정하게 된다.

int TA - disp에서 relative를 반영한 실질적 Target Address를 구해 저장한다.

int start\_loc - 프로그램 수행의 첫 location 주솟값을 저장한다.

int operand - machine insctruction의 operand에 해당하는 메모리 값을 저장하는 변수이다.

## 3.11 모듈이름 : LD\_related\_instruction()

### 3.11.1 *기능*

operand로 (m..m+2)가 오는 즉, operand로 'MEM가 가지고 있는 value'가 오는 경우의 instruction을 처리한다. n과 i의 값을 읽어서 ni == 00 인 경우는 COPY 프로그램에 존재하지 않는다는 가정을 하고, ni == 01인 경우 ni == 10인 경우 ni == 11인 경우로 나누어 해당되는 방식으로 addressing을 수행하여 Val 값을 정한다.

* ni == 01인 경우(immediate addressing)
* Val은 인자로 들어온 TA 자체가 된다.
* ni == 10인 경우(indirect addressing)
* TA를 MEMORY[TA]로 갱신하고, 해당 메모리 주소로 접근하여 메모리가 가지고 있는 값을 Val 값으로 정한다.
* ni == 11인 경우(simple addressing)
* TA 메모리 주소로 접근하여 메모리가 가지고 있는 값을 Val 값으로 정한다.

### 3.11.2 *사용변수*

int Val - 해당 address로 접근하여 MEMORY[TA]의 값을 가져오게 되는데 이를 저장하는 변수이다.

## 3.12 모듈이름 : J\_related\_instruction()

### 3.12.1 *기능*

operand로 m 가 오는 즉, operand로 MEM의 주소가 오는 경우의 mnemonic을 처리한다. LD 관련 명령과 마찬가지로 n과 i의 값을 읽는다. ni == 00과 ni==01인 경우는 COPY 프로그램에 존재하지 않는다는 가정을 하고, ni == 10인 경우와 ni == 11인 경우로 나누어 해당되는 방식으로 addressing을 수행하여 Addr 값을 정한다.

* ni == 10인 경우(indirect addressing)
* MEMORY[TA]의 값을 Addr에 저장한다.
* ni == 11인 경우(simple addressing)
* Addr이 TA가 된다.

### 3.12.2 *사용변수*

int Addr - TA를 저장하는 변수이다. ni == 10인 경우 TA를 MEMORY[TA]로 조정한다.

## 3.13 모듈이름 : ST\_related\_instruction()

### 3.13.1 *기능*

operand로 m..m+2가 오며, 해당 메모리 주소에 val\_to\_push를 적재한다. 적재하려는 데이터가 몇 byte인지를 인자로 받아 해당하는 만큼 메모리에 적재한다. 이를테면, STCH는 1byte만을 적재하는 명령어이고, STA는 3byte를 STF는 6byte를 적재하는 명령이다.

### 3.13.2 *사용변수*

없음

## 3.14 모듈이름 : twos\_complement\_str\_to\_decint()

### 3.14.1 *기능*

modify해야할 대상이 되는 byte code가 2의 보수표현 음수인 경우가 있는데, 이를 그대로 int로 변환하게 되면 양수 값이 되어 보정하는 작업이 필요하다. 문자열 hexstr을 signed 32bit hexstr로 간주하고, 조응하는 dec int값으로 변환한다. hex str 값이 2의 보수표현 음수라면, 해당 값에 " | 0xFF000000" 연산을 하여, 앞에 FF를 붙여 음수 sign bit가 반영된 32bit의 수로 변환한다.

### 3.14.2 *사용변수*

int res - return할 결과를 저장하는 변수이다.

## 3.15 모듈이름 : get\_opcode\_or\_NULL\_by\_opcode()

### 3.15.1 *기능*

project 2에서 구현하였던 get\_opcode() 함수는 인자로 menmonic을 주었을 때, 해당하는 op node를 얻을 수 있는 함수였다. 하지만, project 3 run을 수행할 때는 opcode를 이용하여 op node를 찾아야한다. 해당 함수는 opcode를 입력 받아 해당 opcode에 맞는 OP\_node를 return 하는 작업을 수행한다.

### 3.15.2 *사용변수*

int ind - hash function으로 특정 opcode가 저장된 index에 접근한다. 이 index를 저장하는 변수이다.

# 4. 전역 변수 정의

## 4.1 int ES\_NODE \*ESTAB[ESTAB\_HASH\_SIZE]

ES NODE는 char name[SYMBOL\_LEN], int addr, struct ES\_NODE\* nxt 데이터를 가지고 있는 사용자 정의 데이터 타입 구조체이다. 이를 가리키는 포인터변수로 이루어진 hash table을 정의하였다. ESTAB\_HASH\_SIZE는 20이다.

## 4.2 int BP\_CHK[MEM\_SIZE]

break point가 걸려있는 address를 저장하기 위한 방법으로 위의 자료구조를 정의하였다. BP\_CHK는 메모리의 개수와 동일한 크기를 가지고 있는 배열로 break point가 걸려있는 지점을 true로 바꿈으로써 위치를 표시한다.

## 4.3 int BP\_ADDR[MEM\_SIZE]

4.2와 같이 자료구조를 정의하게 되면 bp 지점들을 조회할 때마다, 메모리 전체를 탐색하게 된다. 효율성을 위해서 BP\_CHK[address] 값이 true인 address를 저장하는 1차원 배열을 정의하였다. 만약, bp가 걸린 지점들을 조회하는 명령을 수행할 때, BP\_CHK[BP\_ADDR[i]]의 값을 가져오면 된다.

## 4.4 int PRINT\_FLAG

메모리에 적재된 instruction들을 수행할 때, bp를 만나게되면, run() 수행을 멈추면서, 그 지점의 register 값들을 화면에 출력하여 보여주게 된다. 이 다음 다시 run()을 수행할 때는 해당 지점에서 register 값들을 화면에 보여줄 필요가 없으므로, 이를 조정하는 flag를 전역변수로 정의하였다.

## 4.5 int NUM\_OF\_BP

bp의 개수를 저장하는 전역변수이다.

## 4.6 int REG[REG\_NUM + 1]

register의 값을 가지고 있는 1차원 배열 자료구조이다. 0번 레지스터(A)에 17이 저장되어있다면, REG[0] == 17로 저장하는 방식이다.

## 4.7 char CC;

Condition Code를 저장하는 전역변수이다. '<', '>', '=' 3가지 값 중 하나의 값을 가지게 된다.

## 4.8 int PROGRAM\_ADDR

프로그램의 시작 주소를 저장하는 전역 변수이다.

## 4.9 int CS\_ADDR

Control Section의 시작 주소를 저장하는 전역 변수이다.

## 4.10 int CS\_LEN

Control Section의 길이를 저장하는 전역 변수이다.

## 4.11 int PROGRAM\_LEN

프로그램의 길이를 저장하는 전역 변수이다.

## 4.12 int FIRST\_INSTRUCTION\_ADDR

프로그램에서 첫번째 instruction이 수행되는 주소를 저장하는 변수이다.

# 5. 코드

## 5.1 20121277.h

//  
// Created by 김주호 on 2021/03/10.  
//  
  
#ifndef SP\_PROJ1\_20121277\_H  
#define SP\_PROJ1\_20121277\_H  
  
/\* 포함되는 파일 \*/  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
  
#include "utils.h"  
#include "shell\_commands.h"  
#include "memory\_commands.h"  
#include "opcode\_table\_commands.h"  
#include "assembler\_commands.h"  
#include "loader\_commands.h"  
  
/\* 정의되는 상수 \*/  
#define MAX\_INPUT\_LEN 100  
#define MAXNUM\_OF\_TOKEN 5  
#define MAX\_TOKEN\_LEN 15  
  
typedef enum SUCCESS\_or\_FAIL {  
 FAIL = 0,  
 SUCCESS  
}SUCCESS\_or\_FAIL;  
  
/\* 전역 변수 \*/  
char INPUT[MAX\_INPUT\_LEN];  
char INPUT\_SPLIT[MAXNUM\_OF\_TOKEN][MAX\_TOKEN\_LEN];  
char REFINED\_INPUT[MAX\_INPUT\_LEN];  
int NUM\_OF\_TOKENS;  
  
/\* 사용자 정의 data type \*/  
typedef enum COMMAND {  
 // 0 ~ 4 : shell commands  
 HELP\_CMD = 0,  
 DIR\_CMD,  
 QUIT\_CMD,  
 HISTORY\_CMD,  
 TYPE\_CMD,  
  
 // 5 ~ 8 : memory commands  
 DUMP\_CMD,  
 EDIT\_CMD,  
 FILL\_CMD,  
 RESET\_CMD,  
  
 // 9, 10 : opcode table commands  
 OPCODE\_MNEMONIC\_CMD,  
 OPCODELIST\_CMD,  
  
 // 11, 12 : assemble commands  
 ASSEMBLE\_CMD,  
 SYMBOL\_CMD,  
  
 // 13, 14, 15, 16 : loader commands  
 PROGADDR\_CMD,  
 LOADER\_CMD,  
 BP\_CMD,  
 RUN\_CMD,  
  
 // 17 : INPUT error  
 WRONG\_CMD  
} COMMAND;  
  
/\* 함수 원형 \*/  
void init();  
void refresh\_input();  
SUCCESS\_or\_FAIL input\_split\_by\_comma();  
COMMAND get\_command();  
SUCCESS\_or\_FAIL execute\_cmd(COMMAND cmd);  
void make\_refined\_input();  
  
#endif //SP\_PROJ1\_20121277\_H

## 5.2 20121277.c

#include "20121277.h"  
  
/\* 프로그램 시작 \*/  
int main() {  
 COMMAND cmd;  
 init();  
 while (1) {  
 printf("sicsim> ");  
  
 refresh\_input();  
 scanf("%[^\n]", INPUT);  
 getchar();  
  
 // 1. 긴 명령어를 입력하거나 2. token의 개수가 5개 이상이면 error 처리  
 if (!input\_split\_by\_comma()) {  
 printf("COMMAND err!\n");  
 continue;  
 }  
 // 3. 예약되어있지 않은 명령어를 입력하거  
 cmd = get\_command();  
 if (cmd == WRONG\_CMD) {  
 printf("err : wrong cmd or wrong number of parameters!\n");  
 continue;  
 } else if (cmd == HISTORY\_CMD) {  
 make\_refined\_input();  
 save\_instructions(REFINED\_INPUT);  
 execute\_cmd(cmd);  
 continue;  
 } else if (cmd == QUIT\_CMD) {  
 free\_log\_of\_instructions();  
 free\_OPTAB();  
 free\_ESTAB();  
 break;  
 }  
  
 if (execute\_cmd(cmd) == FAIL) continue;  
  
 // store instructions  
 make\_refined\_input();  
 save\_instructions(REFINED\_INPUT);  
 }  
 return 0;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : init\*/  
/\*목적 : 명령어들의 history를 저장하기 위한 linked list를 초기화하며, memory의 마지막 주소를 초기화한다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void init() {  
 HEAD = NULL;  
 TAIL = NULL;  
 NUM\_OF\_TOKENS = 0;  
 LAST\_ADDR = -1;  
 init\_OPTAB("opcode.txt");  
 for (int i = 0; i < MEM\_SIZE; i++){  
 BP\_CHK[i] = 0;  
 }  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : refresh\_input\*/  
/\*목적 : 사용자로부터 다시 입력을 받을 수 있는 상태가 되도록, INPUT char 배열, INPUT\_SPLIT[][] char 배열을 초기화\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void refresh\_input() {  
 INPUT[0] = '\0';  
 for (int i = 0; i < MAXNUM\_OF\_TOKEN; i++) {  
 strcpy(INPUT\_SPLIT[i], "\0");  
 }  
 NUM\_OF\_TOKENS = 0;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : input\_split\_by\_comma\*/  
/\*목적 : 사용자로부터 입력 받은 명령어들을 ‘,’ 기호를 기준하여, parsing\*/  
/\*리턴값 : SUCCESS - 성공인 경우, FAIL - 실패인 경\*/  
/\*------------------------------------------------------------------------------------\*/  
SUCCESS\_or\_FAIL input\_split\_by\_comma() {  
 int cur\_ind, dx;  
  
 for (cur\_ind = 0; INPUT[cur\_ind] == ' ' || INPUT[cur\_ind] == '\t'; cur\_ind++);  
 while (cur\_ind < MAX\_INPUT\_LEN) {  
 dx = get\_dx\_to\_nxt\_token(INPUT + cur\_ind);  
  
 if (dx == -1) break;  
 if (NUM\_OF\_TOKENS >= MAXNUM\_OF\_TOKEN - 1) return FAIL;  
 if (strlen(INPUT + cur\_ind) > MAX\_TOKEN\_LEN) return FAIL;  
  
 if (IS\_COMMA == YES\_COMMA) {  
 if (strcmp(INPUT + cur\_ind, "") == 0) {  
 strcat(INPUT\_SPLIT[--NUM\_OF\_TOKENS], ",");  
 NUM\_OF\_TOKENS++;  
 }  
 else {  
 strcpy(INPUT\_SPLIT[NUM\_OF\_TOKENS], INPUT + cur\_ind);  
 strcat(INPUT\_SPLIT[NUM\_OF\_TOKENS++], ",");  
 }  
 }  
 else {  
 strcpy(INPUT\_SPLIT[NUM\_OF\_TOKENS++], INPUT + cur\_ind);  
 }  
 cur\_ind += dx;  
 }  
 return SUCCESS;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : get\_command\*/  
/\*목적 : 사용자로부터 command를 입력받는다.\*/  
/\*리턴값 : \*\_command - 성공인 경우 해당 COMMAND, wrong\_command - 잘못된 명령어인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
COMMAND get\_command() {  
 char \*cmd = INPUT\_SPLIT[0];  
 // shell commands  
 if ((strcmp(cmd, "h") == 0 || strcmp(cmd, "help") == 0) && NUM\_OF\_TOKENS == 1) return HELP\_CMD;  
 else if ((strcmp(cmd, "q") == 0 || strcmp(cmd, "quit") == 0) && NUM\_OF\_TOKENS == 1) return QUIT\_CMD;  
 else if ((strcmp(cmd, "d") == 0 || strcmp(cmd, "dir") == 0) && NUM\_OF\_TOKENS == 1) return DIR\_CMD;  
 else if ((strcmp(cmd, "hi") == 0 || strcmp(cmd, "history") == 0) && NUM\_OF\_TOKENS == 1) return HISTORY\_CMD;  
 else if (strcmp(cmd, "type") == 0 && NUM\_OF\_TOKENS == 2) return TYPE\_CMD;  
  
 // memory commands  
 else if ((strcmp(cmd, "du") == 0 || strcmp(cmd, "dump") == 0) && (NUM\_OF\_TOKENS >= 1 && NUM\_OF\_TOKENS <= 3))  
 return DUMP\_CMD;  
 else if ((strcmp(cmd, "e") == 0 || strcmp(cmd, "edit") == 0) && NUM\_OF\_TOKENS == 3) return EDIT\_CMD;  
 else if ((strcmp(cmd, "f") == 0 || strcmp(cmd, "fill") == 0) && NUM\_OF\_TOKENS == 4) return FILL\_CMD;  
  
 // opcode table commands  
 else if (strcmp(cmd, "reset") == 0 && NUM\_OF\_TOKENS == 1) return RESET\_CMD;  
 else if (strcmp(cmd, "opcode") == 0 && NUM\_OF\_TOKENS == 2) return OPCODE\_MNEMONIC\_CMD;  
 else if (strcmp(cmd, "opcodelist") == 0 && NUM\_OF\_TOKENS == 1) return OPCODELIST\_CMD;  
  
 // assembler commands  
 else if (strcmp(cmd, "assemble") == 0 && NUM\_OF\_TOKENS == 2) return ASSEMBLE\_CMD;  
 else if (strcmp(cmd, "symbol") == 0 && NUM\_OF\_TOKENS == 1) return SYMBOL\_CMD;  
  
 // loader commands  
 else if (strcmp(cmd, "progaddr") == 0 && NUM\_OF\_TOKENS == 2) return PROGADDR\_CMD;  
 else if (strcmp(cmd, "loader") == 0 && NUM\_OF\_TOKENS <= 4) return LOADER\_CMD;  
 else if (strcmp(cmd, "bp") == 0 && (NUM\_OF\_TOKENS >= 1 && NUM\_OF\_TOKENS <= 2)) return BP\_CMD;  
 else if (strcmp(cmd, "run") == 0 && NUM\_OF\_TOKENS == 1) return RUN\_CMD;  
 return WRONG\_CMD;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : execute\_cmd\*/  
/\*목적 : 사용자로부터 입력받은 command를 수행한다.\*/  
/\*리턴값 : SUCCESS - 성공인 경우, FAIL - 실패인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
SUCCESS\_or\_FAIL execute\_cmd(COMMAND cmd) {  
 SUCCESS\_or\_FAIL RESULT = SUCCESS;  
 OK\_or\_ERR state = OK;  
 switch (cmd) {  
 // shell commands  
 case HELP\_CMD:  
 help();  
 break;  
 case QUIT\_CMD:  
 break;  
 case DIR\_CMD:  
 dir();  
 break;  
 case HISTORY\_CMD:  
 history();  
 break;  
 case TYPE\_CMD:  
 type(INPUT\_SPLIT[1]);  
 break;  
  
 // memory commands  
 case DUMP\_CMD:  
 RESULT = dump(NUM\_OF\_TOKENS, INPUT\_SPLIT[1], INPUT\_SPLIT[2]);  
 break;  
 case EDIT\_CMD:  
 RESULT = edit(INPUT\_SPLIT[1], INPUT\_SPLIT[2]);  
 break;  
 case FILL\_CMD:  
 RESULT = fill(INPUT\_SPLIT[1], INPUT\_SPLIT[2], INPUT\_SPLIT[3]);  
 break;  
 case RESET\_CMD:  
 reset();  
 break;  
  
 // opcode table commands  
 case OPCODE\_MNEMONIC\_CMD:  
 RESULT = print\_opcode\_by\_mnemonic(INPUT\_SPLIT[1]);  
 break;  
 case OPCODELIST\_CMD:  
 opcodelist();  
 break;  
  
 // assembler commands  
 case ASSEMBLE\_CMD:  
 state = assemble(INPUT\_SPLIT[1]);  
 if (state < 0) {  
 RESULT = FAIL;  
 }  
 break;  
 case SYMBOL\_CMD:  
 print\_symbols();  
 break;  
  
 // loader commands  
 case PROGADDR\_CMD:  
 RESULT = set\_PROGADDR(INPUT\_SPLIT[1]);  
 break;  
 case LOADER\_CMD:  
 RESULT = loader(INPUT\_SPLIT);  
 break;  
 case BP\_CMD:  
 RESULT = bp\_command(NUM\_OF\_TOKENS, INPUT\_SPLIT[1]);  
 break;  
 case RUN\_CMD:  
 RESULT = run();  
 break;  
  
 default:// WRONG\_CMD  
 printf("COMMAND err!\n");  
 break;  
 }  
 if (RESULT <= 0) RESULT = FAIL;  
 return RESULT;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : make\_refined\_input\*/  
/\*목적 : 사용자로부터 입력 받은 입력어를 정제하여 INPUT\_SPLIT[][] char 배열에 저장한다. \*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void make\_refined\_input() {  
 for (int i = 0; i < NUM\_OF\_TOKENS; i++) {  
 switch (i) {  
 case 0:  
 strcpy(REFINED\_INPUT, INPUT\_SPLIT[i]);  
 break;  
 case 1:  
 strcat(REFINED\_INPUT, " ");  
 strcat(REFINED\_INPUT, INPUT\_SPLIT[i]);  
 break;  
 default:  
 strcat(REFINED\_INPUT, ", ");  
 strcat(REFINED\_INPUT, INPUT\_SPLIT[i]);  
 break;  
 }  
 }  
}

## 5.3 assembler\_commands.h

//  
// Created by 김주호 on 2021/03/29.  
//  
  
#ifndef SP\_PROJ1\_ASSEMBLER\_COMMANDS\_H  
#define SP\_PROJ1\_ASSEMBLER\_COMMANDS\_H  
  
#include <stdio.h>  
#include <string.h>  
#include <dirent.h>  
#include <sys/stat.h>  
#include <stdlib.h>  
  
#include "utils.h"  
#include "opcode\_table\_commands.h"  
  
#define SYMBOL\_LEN 100  
#define LABEL\_LEN 50  
#define MNEMONIC\_LEN 50  
#define OPERAND\_LEN 50  
#define LINE\_NUM\_SCALE 5  
#define MAX\_LINE\_NUM 1000000  
  
#define M\_RECORD\_LEN 10 // T.001000.1E  
#define TOTAL\_M\_RECORD\_SIZE 100  
#define ONELINE\_T\_RECORD\_BYTE\_SIZE 30  
#define ONELINE\_T\_RECORD\_LINE\_SIZE 9+30\*20+1 // T.001000.1D.(30BYTE).\0  
#define OBJ\_CODE\_LEN 20  
#define FORMAT4\_TA\_LEN 5  
  
  
typedef struct SYM\_node {  
 char symbol[SYMBOL\_LEN];  
 int address;  
 struct SYM\_node \*nxt;  
} SYM\_node;  
SYM\_node \*SYMTAB\_HEAD;  
SYM\_node \*LATEST\_SYMTAB;  
  
int B\_val;  
int M\_RECORD\_NUM;  
char M\_RECORDS[TOTAL\_M\_RECORD\_SIZE][M\_RECORD\_LEN];  
  
// assemble 함수  
OK\_or\_ERR assemble(char \*filename);  
OK\_or\_ERR pass1(FILE \*fp, char \*filename, int \*PROGRAM\_SIZE);  
OK\_or\_ERR pass2(char \*filename, int PROGRAM\_SIZE);  
OK\_or\_ERR make\_obj\_code(char \*obj\_code, int PC\_val, char \*MNEMONIC, char \*OP1, char \*OP2, int STARTING\_ADDR);  
  
// symtab 관련 함수  
void print\_symbols();  
SYM\_node \*find\_symbol\_or\_NULL(char \*symbol);  
void push\_to\_symtab(char \*symbol, int addr);  
void free\_SYMTAB(SYM\_node \*head);  
  
  
#endif //SP\_PROJ1\_ASSEMBLER\_COMMANDS\_H

## 5.4 assembler\_commands.c

//  
// Created by 김주호 on 2021/03/29.  
//  
  
#include "assembler\_commands.h"  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : assemble\*/  
/\*목적 : filename에 해당하는 소스파일을 읽어서 object파일(.obj)과 리스팅파일(.lst)을 만든다.\*/  
/\*리턴값 : OK - 성공, FILE\_ERR - 파일 에러\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR assemble(char \*filename) {  
 FILE \*fp = fopen(filename, "r");  
 DIR \*dir = opendir(filename);  
 if (!fp) {  
 printf("There's no such file!\n");  
 return FILE\_ERR;  
 }  
 if (dir) return FILE\_ERR;  
  
 char \*pure\_filename, \*extension;  
 int PROGRAM\_SIZE;  
  
 // 파일 열기에서 오류가 생겼을 때 처리  
 char tmp[NAME\_LEN];  
 strcpy(tmp, filename);  
 pure\_filename = strtok(tmp, ".");  
 if (!pure\_filename) {  
 printf("File name err!\n");  
 fclose(fp);  
 return FILE\_ERR;  
 }  
 extension = strtok(NULL, "");  
 if (strcmp(extension, "asm") != 0) {  
 printf("File extension err!\n");  
 fclose(fp);  
 return FILE\_ERR;  
 }  
  
 // SYMTAB과 B register의 값을 초기화  
 SYMTAB\_HEAD = NULL;  
 B\_val = -1;  
  
 // pass1 실행  
 OK\_or\_ERR pass1\_state = pass1(fp, pure\_filename, &PROGRAM\_SIZE);  
 fclose(fp);  
 if (pass1\_state != OK) {  
 free\_SYMTAB(SYMTAB\_HEAD);  
 return FILE\_ERR;  
 }  
  
 // pass2 실행  
 OK\_or\_ERR pass2\_state = pass2(pure\_filename, PROGRAM\_SIZE);  
 if (pass2\_state != OK) {  
 free\_SYMTAB(SYMTAB\_HEAD);  
 return FILE\_ERR;  
 }  
 free\_SYMTAB(LATEST\_SYMTAB);  
  
 // 가장 최근의 assemble에서 생성한 SYMTAB을 저장  
 LATEST\_SYMTAB = SYMTAB\_HEAD;  
 SYMTAB\_HEAD = NULL;  
  
 // assemble 성공 메세지  
 printf("[%s.lst], [%s.obj]\n", pure\_filename, pure\_filename);  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : pass1\*/  
/\*목적 : 1. .asm 파일을 한 줄씩 읽어들이면서, 각 line에 location counter를 할당하면서 intermediate(확장자 .imt) 파일을 생성한다.  
 2. pass2에서 object code를 만들 때 참조할 SYMBOL TABLE을 생성한다.\*/  
/\*리턴값 : OK - 성공, FILE\_ERR 또는 ASSEMBLY\_CODE\_ERR - 실패\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR pass1(FILE \*fp, char \*filename, int \*PROGRAM\_SIZE) {  
 if (!fp) return FILE\_ERR;  
  
 char line[LINE\_LEN];  
 char LABEL[LABEL\_LEN], MNEMONIC[MNEMONIC\_LEN], OP1[MNEMONIC\_LEN], OP2[MNEMONIC\_LEN];  
 char filename\_itm[NAME\_LEN];  
 FILE \*fp\_itm;  
 INSTRUCTION type; // asm파일 각 line에서 지정하는 명령의 종류를 9가지로 분류하였고, 어떤 유형에 해당하는 지를 저장하는 변수이다.  
 OP\_NODE \*opcode\_mnemonic\_map\_node;  
  
 int LOCCTR = 0, STARTING\_ADDR = 0;  
 int dl, LINE\_NUM = 0; // 'dl' means 'delta LOCCTR'  
  
 // itm file 쓰기모드로 open  
 strcpy(filename\_itm, filename);  
 strcat(filename\_itm, ".itm");  
 fp\_itm = fopen(filename\_itm, "w");  
  
 // read a line from .asm file and parsing  
 fgets(line, LINE\_LEN, fp);  
 line[strlen(line) - 1] = '\0';  
 type = line\_split(line, LABEL, MNEMONIC, OP1, OP2);  
  
 // asm 파일의 가장 가장 첫 line에 START 슈도명령이 없다면 오류  
 if (type != \_START) {  
 printf("Error! Check line number \"%d\"\n", 0);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
 //STARTING\_ADDR = atoi(OP1);  
 STARTING\_ADDR = hexstr\_to\_decint(OP1);  
 LOCCTR = STARTING\_ADDR;  
  
 // asm 첫 명령을 itm 파일에 기록  
 fprintf(fp\_itm, "%04X %-10s %-10s %s %s\n", LOCCTR, LABEL, MNEMONIC, OP1, OP2);  
 //printf("%04X %-10s %-10s %s %s\n", LOCCTR, LABEL, MNEMONIC, OP1, OP2);  
  
 // asm 파일을 읽고 parsing  
 fgets(line, LINE\_LEN, fp);  
 line[strlen(line) - 1] = '\0';  
 type = line\_split(line, LABEL, MNEMONIC, OP1, OP2);  
  
 // END type의 line이 나올때까지 계속 명령을 읽어들이고 parsing을 수행한다.  
 while (1) {  
 // 무한loop에 빠지지 않도록 line을 충분히 많이 읽어들였음에도, END symbol을 만나지 않았다면, 오류를 return  
 if (LINE\_NUM++ >= MAX\_LINE\_NUM) {  
 printf("Err! There's no 'END' pseudo instruction!\n");  
 fclose(fp\_itm);  
 remove(filename\_itm);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
 if (type == \_END) {  
 // itm 파일에 기록  
 fprintf(fp\_itm, "%04X %-10s %-10s %s %s\n", LOCCTR, LABEL, MNEMONIC, OP1, OP2);  
 //printf("%04X %-10s %-10s %s %s\n", LOCCTR, LABEL, MNEMONIC, OP1, OP2);  
  
 // 파일을 닫고, 프로그램의 SIZE를 기록  
 fclose(fp\_itm);  
 \*PROGRAM\_SIZE = LOCCTR - STARTING\_ADDR;  
 break;  
 }  
 if (feof(fp)) {  
 printf("Error! Check line number \"%d\"\n", (LINE\_NUM + 1) \* LINE\_NUM\_SCALE);  
 ////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 여기서 itm 파일 삭제 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///////////  
 fclose(fp\_itm);  
 remove(filename\_itm);  
 return ASSEMBLY\_CODE\_ERR;  
  
 }  
 if (type == \_COMMENT) {  
 while (type == \_COMMENT) {  
 // itm 파일에 기록  
 fprintf(fp\_itm, "%04X %-10s %s\n", LOCCTR, LABEL, MNEMONIC);  
 //printf("%04X %-10s %s\n", LOCCTR, LABEL, MNEMONIC);  
  
 // asm 파일을 읽고 parsing  
 fgets(line, LINE\_LEN, fp);  
 line[strlen(line) - 1] = '\0';  
 type = line\_split(line, LABEL, MNEMONIC, OP1, OP2);  
 }  
 continue;  
 }  
  
 // LABEL을 symtab에 기록  
 if (\*LABEL != '\0') {  
 // LABEL이 SYMTAB에 있다면 duplication error이다.  
 if (find\_symbol\_or\_NULL(LABEL) != NULL) {  
 printf("Error! Check line number \"%d\"\n", (LINE\_NUM + 1) \* LINE\_NUM\_SCALE);  
 ////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 여기서 itm 파일 삭제 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///////////  
 fclose(fp\_itm);  
 remove(filename\_itm);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
 push\_to\_symtab(LABEL, LOCCTR);  
 }  
 // itm 파일에 기록  
 fprintf(fp\_itm, "%04X %-10s %-10s %s %s\n", LOCCTR, LABEL, MNEMONIC, OP1, OP2);  
 //printf("%04X %-10s %-10s %s %s\n", LOCCTR, LABEL, MNEMONIC, OP1, OP2);  
  
  
 // MNEMONIC의 유형에 따라 각기 다른 LOCCTR의 증가량을 부여  
 char mnemonic\_refined[MNEMONIC\_LEN];  
 if (MNEMONIC[0] == '+') strcpy(mnemonic\_refined, MNEMONIC + 1);  
 else  
 strcpy(mnemonic\_refined, MNEMONIC);  
  
 if (type == \_OPERATION && (opcode\_mnemonic\_map\_node = get\_opcode\_or\_NULL\_by\_mnemonic(mnemonic\_refined)) != NULL) {  
 if (strcmp(opcode\_mnemonic\_map\_node->format, "3/4") == 0) {  
 if (MNEMONIC[0] == '+') dl = 4;  
 else dl = 3;  
 }  
 else if (strcmp(opcode\_mnemonic\_map\_node->format, "1") == 0) dl = 1;  
 else if (strcmp(opcode\_mnemonic\_map\_node->format, "2") == 0) dl = 2;  
 }  
 else if (type == \_BASE) dl = 0;  
 else if (type == \_WORD) dl = 3;  
 else if (type == \_RESW) dl = 3 \* atoi(OP1);  
 else if (type == \_RESB) dl = atoi(OP1);  
 else if (type == \_BYTE) dl = get\_byte\_size(OP1);  
 else {  
 printf("Error! Check line number \"%d\"\n", (LINE\_NUM + 1) \* LINE\_NUM\_SCALE);  
 ////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 여기서 itm 파일 삭제 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///////////  
 fclose(fp\_itm);  
 remove(filename\_itm);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
 // LOCCTR 증가  
 LOCCTR += dl;  
  
 // asm 파일을 읽고 parsing  
 fgets(line, LINE\_LEN, fp);  
 line[strlen(line) - 1] = '\0';  
 type = line\_split(line, LABEL, MNEMONIC, OP1, OP2);  
 }  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : pass2\*/  
/\*목적 : itm 파일의 line을 읽으면서 lst 파일과 obj 파일을 생성하는 함수이다.\*/  
/\*리턴값 : OK - 성공, ERR - 실패\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR pass2(char \*filename, int PROGRAM\_SIZE) {  
 // sub routine 실행의 결과 상태를 관리하는 변수  
 OK\_or\_ERR state;  
  
 // fp\_itm은 읽기모드로, fp\_lst와 fp\_obj는 쓰기모드로 파일을 연다  
 FILE \*fp\_itm, \*fp\_lst, \*fp\_obj;  
 if (file\_open(filename, &fp\_itm, &fp\_lst, &fp\_obj) != OK) {  
 return FILE\_ERR;  
 }  
  
 // init M\_records  
 M\_RECORD\_NUM = 0;  
 for (int i = 0; i < TOTAL\_M\_RECORD\_SIZE; i++) {  
 strcpy(M\_RECORDS[i], "\0");  
 }  
  
 // 하나의 line에 대한 parsing과 관련한 변수들  
 INSTRUCTION type;  
 SYM\_node \*symbol\_node;  
 char obj\_code[OBJ\_CODE\_LEN];  
 char line[LINE\_LEN];  
 char LABEL[LABEL\_LEN], MNEMONIC[MNEMONIC\_LEN], OP1[OPERAND\_LEN], OP2[OPERAND\_LEN];  
  
 // 프로그램의 정보들을 관리 및 저장하는 변수들  
 char PROGRAM\_NAME[NAME\_LEN];  
 int LOCCTR = 0, LINE\_NUM = 1, STARTING\_ADDR = 0;  
 char ONELINE\_T\_RECORD[ONELINE\_T\_RECORD\_LINE\_SIZE];  
 int T\_RECORD\_ACCUMULATED\_BYTE = 0;  
 int var\_flag = 0, new\_line\_flag = 0;  
  
 // itm 파일을 읽고 parsing  
 fgets(line, LINE\_LEN, fp\_itm);  
 line[strlen(line) - 1] = '\0';  
 line\_split2(line, &LOCCTR, LABEL, MNEMONIC, OP1, OP2);  
  
 // 프로그램 이름과 starting address, START symbol lst 파일에 기록  
 fprintf(fp\_lst, "%3d %-40s\n", (LINE\_NUM++) \* LINE\_NUM\_SCALE, line);  
 //printf("-lst-%3d %-40s\n", (LINE\_NUM++) \* LINE\_NUM\_SCALE, line);  
 STARTING\_ADDR = LOCCTR;  
 strcpy(PROGRAM\_NAME, LABEL);  
  
 // itm 파일의 두번째 line을 읽고 parsing  
 fgets(line, LINE\_LEN, fp\_itm);  
 line[strlen(line) - 1] = '\0';  
 type = line\_split2(line, &LOCCTR, LABEL, MNEMONIC, OP1, OP2);  
  
 // obj 파일에 H record 기록한다  
 // line byte size를 일단 비워두고, char 1차원 배열에 T 레코드를 기록해간다  
 //printf("-obj-H%-6s%06X%06X\n", PROGRAM\_NAME, STARTING\_ADDR, PROGRAM\_SIZE);  
 fprintf(fp\_obj, "H%-6s%06X%06X\n", PROGRAM\_NAME, STARTING\_ADDR, PROGRAM\_SIZE);  
 sprintf(ONELINE\_T\_RECORD, "T%06X\_\_", STARTING\_ADDR);  
  
 int cnt = 0;  
 while (1) {  
 cnt++;  
 if (type == \_END) {  
 // line byte size를 T 레코드에 기록  
 char line\_byte\_size[3];  
 sprintf(line\_byte\_size, "%02X", T\_RECORD\_ACCUMULATED\_BYTE);  
 ONELINE\_T\_RECORD[7] = line\_byte\_size[0];  
 ONELINE\_T\_RECORD[8] = line\_byte\_size[1];  
 fprintf(fp\_obj, "%s\n", ONELINE\_T\_RECORD);  
 //printf("-obj-%s\n", ONELINE\_T\_RECORD);  
  
 // M\_record 출력  
 for (int i = 0; i < M\_RECORD\_NUM; i++) {  
 fprintf(fp\_obj, "%s\n", M\_RECORDS[i]);  
 //printf("-obj-%s\n", M\_RECORDS[i]);  
 }  
 fprintf(fp\_obj, "E%06X", STARTING\_ADDR);  
 fprintf(fp\_lst, "%3d %-4s %-35s", (LINE\_NUM++) \* LINE\_NUM\_SCALE, "", line + 5);  
 //printf("-obj-E%06X", STARTING\_ADDR);  
 //printf("-lst-E%06X", STARTING\_ADDR);  
  
 fclose(fp\_obj);  
 fclose(fp\_lst);  
 fclose(fp\_itm);  
 ////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 여기서 itm 파일 삭제 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///////////  
 char filename\_itm[NAME\_LEN];  
 strcpy(filename\_itm, filename);  
 strcat(filename\_itm, ".itm");  
 remove(filename\_itm);  
 break;  
 }  
 if (feof(fp\_itm)) {  
 printf("Error! Check line number \"%d\"\n", LINE\_NUM \* LINE\_NUM\_SCALE);  
 ////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 여기서 파일 닫고 itm, obj, lsm 파일 삭제 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///////////  
 fclose(fp\_obj); fclose(fp\_lst); fclose(fp\_itm);  
 char filename\_itm[NAME\_LEN]; char filename\_lst[NAME\_LEN]; char filename\_obj[NAME\_LEN];  
 strcpy(filename\_itm, filename); strcpy(filename\_lst, filename); strcpy(filename\_obj, filename);  
 strcat(filename\_itm, ".itm"); strcat(filename\_lst, ".lst"); strcat(filename\_obj, ".obj");  
 remove(filename\_itm); remove(filename\_lst); remove(filename\_obj);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
 if (type == \_COMMENT) {  
 while (type == \_COMMENT) {  
 strcpy(obj\_code, "\0");  
 fprintf(fp\_lst, "%3d %-4s %-35s %-s\n", (LINE\_NUM++) \* LINE\_NUM\_SCALE, "", line + 5, obj\_code);  
 //printf("-lst-%3d %-4s %-35s %-s\n", (LINE\_NUM++)\*LINE\_NUM\_SCALE, "", line + 5, obj\_code);  
  
 // itm 파일을 읽고 parsing  
 fgets(line, LINE\_LEN, fp\_itm);  
 line[strlen(line) - 1] = '\0';  
 type = line\_split2(line, &LOCCTR, LABEL, MNEMONIC, OP1, OP2);  
 }  
 continue;  
 }  
  
 // line의 type에 따라 다른 방식으로 obj\_code를 생성한다  
 char mnemonic\_refined[OBJ\_CODE\_LEN];  
 strcpy(obj\_code, "\0");  
 if (MNEMONIC[0] == '+') strcpy(mnemonic\_refined, MNEMONIC + 1);  
 else  
 strcpy(mnemonic\_refined, MNEMONIC);  
  
 if (type == \_OPERATION && get\_opcode\_or\_NULL\_by\_mnemonic(mnemonic\_refined) != NULL) {  
 state = make\_obj\_code(obj\_code, LOCCTR, MNEMONIC, OP1, OP2, STARTING\_ADDR);  
 if (state != OK) {  
 printf("Error! Check line number \"%d\"\n", LINE\_NUM \* LINE\_NUM\_SCALE);  
 ////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 여기서 파일 닫고 itm, obj, lsm 파일 삭제 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///////////  
 fclose(fp\_obj); fclose(fp\_lst); fclose(fp\_itm);  
 char filename\_itm[NAME\_LEN]; char filename\_lst[NAME\_LEN]; char filename\_obj[NAME\_LEN];  
 strcpy(filename\_itm, filename); strcpy(filename\_lst, filename); strcpy(filename\_obj, filename);  
 strcat(filename\_itm, ".itm"); strcat(filename\_lst, ".lst"); strcat(filename\_obj, ".obj");  
 remove(filename\_itm); remove(filename\_lst); remove(filename\_obj);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
 }  
 else if (type == \_BYTE) {  
 if (OP1[0] == 'C') {  
 char characters[OPERAND\_LEN];  
 strcpy(characters, OP1);  
 char hexstr[10];  
 char \*ptr = strtok(characters, " C'`");  
 for (int i = 0; i < (int) strlen(ptr); i++) {  
 sprintf(hexstr, "%02X", (int) \*(ptr + i));  
 strcat(obj\_code, hexstr);  
 }  
 }  
 else if (OP1[0] == 'X') {  
 char hexstr[OPERAND\_LEN];  
 strcpy(hexstr, OP1);  
 char \*ptr = strtok(hexstr, " X'`");  
 strcpy(obj\_code, ptr);  
 }  
 }  
 else if (type == \_WORD) sprintf(obj\_code, "%06X", atoi(OP1));  
 else if (type == \_BASE) {  
 symbol\_node = find\_symbol\_or\_NULL(OP1);  
 if (!symbol\_node) {  
 printf("Error! Check line number \"%d\"\n", LINE\_NUM \* LINE\_NUM\_SCALE);  
 ////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 여기서 파일 닫고 itm, obj, lsm 파일 삭제 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///////////  
 fclose(fp\_obj); fclose(fp\_lst); fclose(fp\_itm);  
 char filename\_itm[NAME\_LEN]; char filename\_lst[NAME\_LEN]; char filename\_obj[NAME\_LEN];  
 strcpy(filename\_itm, filename); strcpy(filename\_lst, filename); strcpy(filename\_obj, filename);  
 strcat(filename\_itm, ".itm"); strcat(filename\_lst, ".lst"); strcat(filename\_obj, ".obj");  
 remove(filename\_itm); remove(filename\_lst); remove(filename\_obj);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
 B\_val = symbol\_node->address;  
 }  
 else if (type == \_RESW || type == \_RESB) {  
 while (type == \_RESW || type == \_RESB) {  
 fprintf(fp\_lst, "%3d %-40s %-s\n", (LINE\_NUM++) \* LINE\_NUM\_SCALE, line, obj\_code);  
 //printf("-lst-%3d %-40s %-s\n", (LINE\_NUM++) \* LINE\_NUM\_SCALE, line, obj\_code);  
 fgets(line, LINE\_LEN, fp\_itm);  
 line[strlen(line) - 1] = '\0';  
 type = line\_split2(line, &LOCCTR, LABEL, MNEMONIC, OP1, OP2);  
 }  
 var\_flag = 1;  
 new\_line\_flag = 1;  
 }  
 else {  
 printf("Error! Check line number \"%d\"\n", LINE\_NUM \* LINE\_NUM\_SCALE);  
 ////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 여기서 파일 닫고 itm, obj, lsm 파일 삭제 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///////////  
 fclose(fp\_obj); fclose(fp\_lst); fclose(fp\_itm);  
 char filename\_itm[NAME\_LEN]; char filename\_lst[NAME\_LEN]; char filename\_obj[NAME\_LEN];  
 strcpy(filename\_itm, filename); strcpy(filename\_lst, filename); strcpy(filename\_obj, filename);  
 strcat(filename\_itm, ".itm"); strcat(filename\_lst, ".lst"); strcat(filename\_obj, ".obj");  
 remove(filename\_itm); remove(filename\_lst); remove(filename\_obj);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
  
 // 개행을 하는 경우 1. line byte size를 초과했을 때, 2. RESW, RESB 명령을 만나서 메모리공간을 사용할 때,  
 if (T\_RECORD\_ACCUMULATED\_BYTE + strlen(obj\_code) / 2 > ONELINE\_T\_RECORD\_BYTE\_SIZE) new\_line\_flag = 1;  
 if (new\_line\_flag) {  
 char line\_byte\_size[3];  
 sprintf(line\_byte\_size, "%02X", T\_RECORD\_ACCUMULATED\_BYTE);  
 ONELINE\_T\_RECORD[7] = line\_byte\_size[0];  
 ONELINE\_T\_RECORD[8] = line\_byte\_size[1];  
  
 // T레코드 라인 출력  
 fprintf(fp\_obj, "%s\n", ONELINE\_T\_RECORD);  
 //printf("-obj-%s\n", ONELINE\_T\_RECORD);  
 T\_RECORD\_ACCUMULATED\_BYTE = 0;  
  
 // 새로운 line에서 T레코드 기록 시작  
 sprintf(ONELINE\_T\_RECORD, "T%06X\_\_", LOCCTR);  
 new\_line\_flag = 0;  
 }  
 T\_RECORD\_ACCUMULATED\_BYTE += (int) strlen(obj\_code) / 2;  
 strcat(ONELINE\_T\_RECORD, obj\_code);  
  
 if (var\_flag) {  
 var\_flag = 0;  
 continue;  
 }  
  
 // lst 파일에 기록  
 fprintf(fp\_lst, "%3d %-40s %-s\n", (LINE\_NUM++) \* LINE\_NUM\_SCALE, line, obj\_code);  
 //printf("-lst-%3d %-40s %-s\n", (LINE\_NUM++)\*LINE\_NUM\_SCALE, line, obj\_code);  
  
 // itm 파일을 읽고 parsing  
 fgets(line, LINE\_LEN, fp\_itm);  
 line[strlen(line) - 1] = '\0';  
 type = line\_split2(line, &LOCCTR, LABEL, MNEMONIC, OP1, OP2);  
 }  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : make\_obj\_code\*/  
/\*목적 : mnemonic과 operand를 입력받아 object code를 생성하는 함수\*/  
/\*리턴값 : OK - 성공, ERR - 실패\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR make\_obj\_code(char \*obj\_code, int PC\_val, char \*MNEMONIC, char \*OP1, char \*OP2, int STARTING\_ADDR) {  
 int format = -1;  
 int n, i, x, b, p, e;  
 int DISP;  
 char m\_record[M\_RECORD\_LEN];  
  
 OP\_NODE \*opcode\_memonic\_map\_node;  
 int is\_digit\_operand = 1;  
 SYM\_node \*sym;  
  
 char mnemonic\_refined[MNEMONIC\_LEN];  
 if (MNEMONIC[0] == '+') { // 4형식  
 strcpy(mnemonic\_refined, MNEMONIC + 1);  
 e = 1;  
 }  
 else {  
 strcpy(mnemonic\_refined, MNEMONIC);  
 e = 0;  
 }  
 opcode\_memonic\_map\_node = get\_opcode\_or\_NULL\_by\_mnemonic(mnemonic\_refined);  
  
 if (e == 1) format = 4;  
 else if (strcmp(opcode\_memonic\_map\_node->format, "3/4") == 0) format = 3;  
 else if (strcmp(opcode\_memonic\_map\_node->format, "2") == 0) format = 2;  
 else if (strcmp(opcode\_memonic\_map\_node->format, "1") == 0) format = 1;  
 PC\_val += format; // 현재 line의 format을 안다면, PC 값을 알 수가 있다.  
  
 REG\_num REG1, REG2;  
 switch (format) {  
 case 1:  
 sprintf(obj\_code, "%02X", opcode\_memonic\_map\_node->opcode);  
 break;  
 case 2:  
 if (\*OP1 == '\0') {  
 printf("\"%s\" need a operand!\n", mnemonic\_refined);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
  
 char reg\_str[OPERAND\_LEN];  
 strcpy(reg\_str, OP1);  
 if (OP1[strlen(OP1) - 1] == ',') reg\_str[strlen(OP1) - 1] = '\0';  
 REG1 = get\_REG\_num(reg\_str);  
 if (REG1 == non\_exist) { // 존재하지 않는 reg에 접근  
 printf("There's no reg \"%s\"\n", reg\_str);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
  
 if (\*OP2 == '\0') { // reg operand가 1개인 경우(OP2가 없다)  
 sprintf(obj\_code, "%02X%X%X", opcode\_memonic\_map\_node->opcode, REG1, 0);  
 break;  
 }  
  
 // reg operand가 2개인 경  
 if (OP1[strlen(OP1) - 1] != ',') { // operand 2개를 입력하는데 comma를 사이에 넣지 않았다면 오류  
 return COMMA\_ERR;  
 }  
 strtok(OP1, " ,");  
 REG2 = get\_REG\_num(OP2);  
 if (REG2 == non\_exist) { // 존재하지 않는 reg에 접근  
 printf("There's no reg \"%s\"\n", OP2);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
 sprintf(obj\_code, "%02X%X%X", opcode\_memonic\_map\_node->opcode, REG1, REG2);  
 break;  
  
 case 3:  
 // n, i를 set  
 if (OP1[0] == '#') n = 0, i = 1; // immediate addressing  
 else if (OP1[0] == '@') n = 1, i = 0; // indirect addressing  
 else n = 1, i = 1; // direct addressing  
  
 // x를 set  
 if (OP2[0] != 'X') x = 0; // index  
 else x = 1;  
  
 char \*ptr;  
 // b, p를 set : PC relative addressing 먼저 시도해보고, 불가능하다면 BASE relative addressing  
 ptr = strtok(OP1, " #@,");  
  
 // RSUB 같은 명령어는 operand가 없는 3형식 명령  
 if (!ptr) {  
 b = 0, p = 0, DISP = 0;  
 sprintf(obj\_code, "%02X%0X%03X", opcode\_memonic\_map\_node->opcode + 2 \* n + i, 8 \* x + 4 \* b + 2 \* p + e,  
 DISP & 0xFFF);  
 break;  
 }  
  
 // digit operand라면 symtab을 탐색할 필요가 없다  
 for (int i = 0; i < (int) strlen(ptr); i++) {  
 if (!isdigit(ptr[i])) {  
 is\_digit\_operand = 0;  
 break;  
 }  
 }  
 if (is\_digit\_operand) {  
 DISP = atoi(OP1 + 1);  
 if (strcmp(MNEMONIC, "LDB") == 0) B\_val = DISP;  
  
 b = 0, p = 0;  
 sprintf(obj\_code, "%02X%0X%03X", opcode\_memonic\_map\_node->opcode + 2 \* n + i, 8 \* x + 4 \* b + 2 \* p + e,  
 DISP & 0xFFF);  
 break;  
 }  
  
 // sym tab에서 symbol을 꺼내어 opcode 생성  
 sym = find\_symbol\_or\_NULL(ptr);  
 if (sym) {  
 // set BASE  
 if (strcmp(MNEMONIC, "LDB") == 0) B\_val = sym->address;  
  
 // PC relative를 최우선으로 시도  
 DISP = sym->address - PC\_val;  
 if (-2048 <= DISP && DISP <= 2047) {  
 b = 0, p = 1;  
 }  
 // BASE relative를 다음으로 시도  
 else if (B\_val > -1 && 0 <= sym->address - B\_val && sym->address - B\_val <= 4095) {  
 DISP = sym->address - B\_val;  
 b = 1, p = 0;  
 }  
 // 3형식인데 relative 접근이 되지 않는다면 에러  
 else return ASSEMBLY\_CODE\_ERR;  
 }  
 else {  
 printf("symbol err: %s\n", ptr);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
 sprintf(obj\_code, "%02X%0X%03X", opcode\_memonic\_map\_node->opcode + 2 \* n + i, 8 \* x + 4 \* b + 2 \* p + e,  
 DISP & 0xFFF);  
 break;  
  
 case 4:  
 // n, i를 set  
 if (OP1[0] == '#') n = 0, i = 1;  
 else if (OP1[0] == '@') n = 1, i = 0;  
 else n = 1, i = 1;  
  
 // x를 set  
 if (OP2[0] == 'X') x = 1;  
 else x = 0;  
  
 // b, p : PC\_val relative 먼저 시도해보고, 불가능하다면 BASE relative  
 ptr = strtok(OP1, " #@,");  
 if (!ptr) b = 0, p = 0, DISP = 0;  
  
 // digit\_operand라면 symtab을 탐색할 필요가 없다.  
 is\_digit\_operand = 1;  
 for (int i = 0; i < (int) strlen(ptr); i++) {  
 if (!isdigit(ptr[i])) {  
 is\_digit\_operand = 0;  
 break;  
 }  
 }  
 if (is\_digit\_operand) {  
 DISP = atoi(OP1 + 1);  
 if (strcmp(MNEMONIC, "LDB") == 0) B\_val = DISP;  
  
 b = 0, p = 0;  
 sprintf(obj\_code, "%02X%0X%05X", opcode\_memonic\_map\_node->opcode + 2 \* n + i, 8 \* x + 4 \* b + 2 \* p + e,  
 DISP & 0xFFFFF);  
 break;  
 }  
  
 // sym tab에서 symbol을 꺼내어 opcode 생성  
 SYM\_node \*sym;  
 sym = find\_symbol\_or\_NULL(ptr);  
 if (sym) {  
 // set BASE  
 if (strcmp(MNEMONIC, "LDB") == 0) B\_val = sym->address;  
 b = 0, p = 0;  
  
 DISP = sym->address;  
 sprintf(obj\_code, "%02X%0X%05X", opcode\_memonic\_map\_node->opcode + 2 \* n + i, 8 \* x + 4 \* b + 2 \* p + e,  
 DISP & 0xFFFFF);  
 sprintf(m\_record, "M%06X%02X", PC\_val - STARTING\_ADDR - 3, FORMAT4\_TA\_LEN);  
 strcpy(M\_RECORDS[M\_RECORD\_NUM++], m\_record);  
 }  
 else {  
 printf("symbol err: %s\n", ptr);  
 return ASSEMBLY\_CODE\_ERR;  
 }  
 break;  
 default:  
 break;  
 }  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : print\_symbols\*/  
/\*목적 : assemble 과정 중에 생성된 symbol table을 화면에 출력한다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void print\_symbols() {  
 SYM\_node \*cur\_node = LATEST\_SYMTAB;  
 if (cur\_node == NULL){  
 printf("There's no symbol table. Assemble first!\n");  
 }  
 while (cur\_node) {  
 printf("\t%-10s %04X\n", cur\_node->symbol, cur\_node->address);  
 cur\_node = cur\_node->nxt;  
 }  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : find\_symbol\_or\_NULL\*/  
/\*목적 : symtab을 탐색하여, 인자로 주어진 symbol string에 해당하는 정보를 찾는다\*/  
/\*리턴값 : symbol string이 symtab에 있다면, 해당 정보를 return, 그렇지 않다면, NULL을 return\*/  
/\*------------------------------------------------------------------------------------\*/  
SYM\_node \*find\_symbol\_or\_NULL(char \*symbol) {  
 SYM\_node \*cur\_node = SYMTAB\_HEAD;  
 for (; cur\_node != NULL; cur\_node = cur\_node->nxt) {  
 if (strcmp(cur\_node->symbol, symbol) == 0) return cur\_node;  
 }  
 return cur\_node;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : push\_to\_symtab\*/  
/\*목적 : symbol과 그에 대응하는 location 값을 symtab에 저장하는 함수이다.\*/  
/\*리턴값 : 없음 \*/  
/\*------------------------------------------------------------------------------------\*/  
void push\_to\_symtab(char \*symbol, int addr) {  
 SYM\_node \*pre\_node = NULL;  
 SYM\_node \*cur\_node = SYMTAB\_HEAD;  
 SYM\_node \*node\_to\_insert = malloc(sizeof(SYM\_node));  
  
 // 삽입할 노드 생성  
 strcpy(node\_to\_insert->symbol, symbol);  
 node\_to\_insert->address = addr;  
 node\_to\_insert->nxt = NULL;  
  
 // symtab이 비어있다면, node를 head에 바로 삽입  
 if (!SYMTAB\_HEAD) {  
 SYMTAB\_HEAD = node\_to\_insert;  
 return;  
 }  
  
 // linked list를 순회하면서 알파벳 순서에 맞는 위치를 찾고 삽입하는 알고리즘  
 for (; cur\_node; pre\_node = cur\_node, cur\_node = cur\_node->nxt) {  
 if (strcmp(cur\_node->symbol, node\_to\_insert->symbol) > 0) {  
 if (!pre\_node) { // 삽입할 노드가 알파벳 가장 첫 순서라면, head에 삽입  
 SYMTAB\_HEAD = node\_to\_insert;  
 node\_to\_insert->nxt = cur\_node;  
 }  
 else { // 알파벳 순서에 맞게 삽입  
 pre\_node->nxt = node\_to\_insert;  
 node\_to\_insert->nxt = cur\_node;  
 }  
 return;  
 }  
 }  
 // 삽입할 노드가 알파벳 마지막 순서라면, tail에 삽  
 pre\_node->nxt = node\_to\_insert;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : free\_SYMTAB\*/  
/\*목적 : SYMTAB에 할당되어 있는 데이터를 해제하는 함수\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void free\_SYMTAB(SYM\_node \*head) {  
 SYM\_node \*cur\_node = head;  
 SYM\_node \*pre\_node = NULL;  
  
 for (; cur\_node; pre\_node = cur\_node, cur\_node = cur\_node->nxt) {  
 free(pre\_node);  
 }  
 head = NULL;  
}

## 5.5 memory\_commands.h

//  
// Created by 김주호 on 2021/03/14.  
//  
  
#ifndef SP\_PROJ1\_MEMORY\_COMMANDS\_H  
#define SP\_PROJ1\_MEMORY\_COMMANDS\_H  
  
/\* 포함되는 파일 \*/  
#include <stdio.h>  
#include <string.h>  
  
#include "utils.h"  
  
/\* 정의되는 상수 \*/  
#define MEM\_SIZE 1048576 // 16\*65536  
#define ONE\_BYTE\_SIZE 256 // FF(hex) == 255(dec)  
  
/\* 전역 변수 \*/  
int MEMORY[MEM\_SIZE];  
int LAST\_ADDR;  
  
/\* 함수 원형 \*/  
OK\_or\_ERR set\_actual\_start\_and\_end(int num\_of\_args, char \*l\_or\_NULL, char \*r\_or\_NULL, int \*start\_dec, int \*end\_dec);  
OK\_or\_ERR dump(int num\_of\_tokens, char \*l\_or\_NULL, char \*r\_or\_NULL);  
void print\_memory(int start, int end);  
OK\_or\_ERR edit(char \*addr\_hexstr, char \*val\_hexstr);  
OK\_or\_ERR fill(char \*start\_hexstr, char \*end\_hexstr, char \*val\_hexstr);  
void reset();  
  
#endif //SP\_PROJ1\_MEMORY\_COMMANDS\_H

## 5.6 memory\_commands.c

//  
// Created by 김주호 on 2021/03/14.  
//  
  
#include "memory\_commands.h"  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : dump\*/  
/\*목적 : 메모리가 가지고 있는 값을 사용자에게 보여준다.\*/  
/\*리턴값 : OK - 성공인 경우, COMMA\_ERR - ','에 문제가 있는 에러인 경우, RANGE\_ERR - 인자에 문제가 있는 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR dump(int num\_of\_tokens, char \*l\_or\_NULL, char \*r\_or\_NULL) {  
 int S\_dec, E\_dec;  
 int num\_of\_args = num\_of\_tokens - 1;  
 int STATE = OK;  
  
 // dump의 명령의 종류는 parameter의 종류에 따라 3가지  
 // 1. dump 2. dump start 3. dump start, end  
 switch (num\_of\_args) {  
 case 0:  
 set\_actual\_start\_and\_end(num\_of\_args, l\_or\_NULL, r\_or\_NULL, &S\_dec, &E\_dec);  
 break;  
 case 1:  
 STATE = set\_actual\_start\_and\_end(num\_of\_args, l\_or\_NULL, r\_or\_NULL, &S\_dec, &E\_dec);  
 break;  
 case 2:  
 STATE = set\_actual\_start\_and\_end(num\_of\_args, l\_or\_NULL, r\_or\_NULL, &S\_dec, &E\_dec);  
 if (STATE == COMMA\_ERR) {  
 printf("comma err! (use ',' between start address and end address)\n");  
 return STATE;  
 }  
 break;  
 default:  
 printf("COMMAND err! too many args!\n");  
 return STATE;  
 }  
 if (STATE == RANGE\_ERR) {  
 printf("range err!\n");  
 return STATE;  
 }  
 print\_memory(S\_dec, E\_dec);  
 LAST\_ADDR = E\_dec;  
 return STATE;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : set\_actual\_start\_and\_end\*/  
/\*목적 : 사용자가 입력한 start address와 end address에 대한 조정이 필요한 경우 이를 수행한다.\*/  
/\*리턴값 : OK - 성공인 경우, COMMA\_ERR - ','에 문제가 있는 에러인 경우, RANGE\_ERR - 인자에 문제가 있는 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR set\_actual\_start\_and\_end(int num\_of\_args, char \*l\_or\_NULL, char \*r\_or\_NULL, int \*start\_dec, int \*end\_dec) {  
 // 1. arg가 0개인 case0 2. arg가 1개인 case1 3. arg가 2개인 case2  
 switch (num\_of\_args) {  
 case 0: // dump  
 \*start\_dec = LAST\_ADDR + 1 < MEM\_SIZE ? LAST\_ADDR + 1 : 0;  
 \*end\_dec = \*start\_dec + 16 \* 10 - 1 < MEM\_SIZE ? \*start\_dec + 16 \* 10 - 1 : MEM\_SIZE - 1;  
 break;  
 case 1: // dump AA  
 \*start\_dec = hexstr\_to\_decint(l\_or\_NULL);  
 \*end\_dec = \*start\_dec + 16 \* 10 - 1 < MEM\_SIZE ? \*start\_dec + 16 \* 10 - 1 : MEM\_SIZE - 1;  
 break;  
 case 2: // dump AA, BB  
 if (l\_or\_NULL[strlen(l\_or\_NULL) - 1] != ',') return COMMA\_ERR;  
 l\_or\_NULL[strlen(l\_or\_NULL) - 1] = '\0';  
 \*start\_dec = hexstr\_to\_decint(l\_or\_NULL);  
 \*end\_dec = hexstr\_to\_decint(r\_or\_NULL);  
 break;  
 default:  
 printf("err! wrong parameter!\n");  
 break;  
 }  
 if (\*start\_dec < 0 || \*start\_dec > MEM\_SIZE || \*end\_dec < 0 || \*end\_dec > MEM\_SIZE) return RANGE\_ERR;  
 if (\*start\_dec > \*end\_dec) return RANGE\_ERR;  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : print\_memory\*/  
/\*목적 : 지정된 범위에 속해있는 MEM 주소와 MEM가 가지고 있는 값을 출력한다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void print\_memory(int start, int end) {  
 int begin\_row = start / 16, end\_row = end / 16;  
 int MAX\_COL\_SIZE = 16;  
  
 for (int r = begin\_row; r <= end\_row; r++) {  
 printf("%05X ", r \* 16);  
 for (int c = 0; c < MAX\_COL\_SIZE; c++) {  
 int cur = r \* MAX\_COL\_SIZE + c;  
 if (cur >= start && cur <= end) printf("%X%X ", (MEMORY[cur] & 0XF0) >> 4, MEMORY[cur] & 0XF);  
 else printf(" ");  
 }  
 printf("; ");  
 for (int c = 0; c < MAX\_COL\_SIZE; c++) {  
 int cur = r \* MAX\_COL\_SIZE + c;  
 char ASCII = MEMORY[cur];  
  
 if (cur >= start && cur <= end && ASCII >= 32 && ASCII <= 126) printf("%c", ASCII);  
 else printf(".");  
 }  
 printf("\n");  
 }  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : edit\*/  
/\*목적 : 지정한 메모리 주소가 갖는 값을 변경한다.\*/  
/\*리턴값 : OK - 성공인 경우, COMMA\_ERR - ','에 문제가 있는 에러인 경우, RANGE\_ERR - 인자에 문제가 있는 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR edit(char \*addr\_hexstr, char \*val\_hexstr) {  
 int addr\_int, val\_int;  
 // hexstr to dec  
 if (addr\_hexstr[strlen(addr\_hexstr) - 1] != ',') {  
 printf("comma err! (use ',' between start address and end address)\n");  
 return COMMA\_ERR;  
 }  
 addr\_hexstr[strlen(addr\_hexstr) - 1] = '\0';  
 addr\_int = hexstr\_to\_decint(addr\_hexstr);  
 val\_int = hexstr\_to\_decint(val\_hexstr);  
  
 // validate  
 if (addr\_int < 0 || addr\_int >= MEM\_SIZE) {  
 printf("range err!\n");  
 return RANGE\_ERR;  
 }  
 if (val\_int < 0 || val\_int >= ONE\_BYTE\_SIZE) {  
 printf("range err!\n");  
 return RANGE\_ERR;  
 }  
 MEMORY[addr\_int] = val\_int;  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : fill\*/  
/\*목적 : 사용자가 지정한 메모리 주소 범위에 대하여 사용자가 원하는 값을 대입하는 함수이다.\*/  
/\*리턴값 : OK - 성공인 경우, COMMA\_ERR - ','에 문제가 있는 에러인 경우, RANGE\_ERR - 인자에 문제가 있는 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR fill(char\* start\_hexstr, char\* end\_hexstr, char\* val\_hexstr){  
 int start\_dec, end\_dec, val\_dec;  
  
 if (start\_hexstr[strlen(start\_hexstr)-1] != ',') {  
 printf("comma err! (use ',' between start address and end address)\n");  
 return COMMA\_ERR;  
 }  
 if (end\_hexstr[strlen(end\_hexstr)-1]!= ',') {  
 printf("comma err! (use ',' between start address and end address)\n");  
 return COMMA\_ERR;  
 }  
 start\_hexstr[strlen(start\_hexstr)-1] = '\0'; end\_hexstr[strlen(end\_hexstr)-1]= '\0';  
 start\_dec = hexstr\_to\_decint(start\_hexstr); end\_dec = hexstr\_to\_decint(end\_hexstr); val\_dec = hexstr\_to\_decint(val\_hexstr);  
  
 if (start\_dec < 0 || start\_dec >= MEM\_SIZE) {  
 printf("range err!\n");  
 return RANGE\_ERR;  
 }  
 if (end\_dec < 0 || end\_dec >= MEM\_SIZE) {  
 printf("range err!\n");  
 return RANGE\_ERR;  
 }  
 if (start\_dec > end\_dec){  
 printf("range err!\n");  
 return RANGE\_ERR;  
 }  
 if (val\_dec < 0 || val\_dec >= ONE\_BYTE\_SIZE){  
 printf("range err!\n");  
 return RANGE\_ERR;  
 }  
  
 for (int i = start\_dec; i<= end\_dec; i++) MEMORY[i] = val\_dec;  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : reset\*/  
/\*목적 : 모든 MEMORY 공간의 값을 0으로 초기화한다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void reset(){  
 for (int i=0; i<MEM\_SIZE; i++){  
 MEMORY[i] = 0;  
 }  
}

## 5.7 shell\_commands.h

//  
// Created by 김주호 on 2021/03/12.  
//  
  
#ifndef SP\_PROJ1\_SHELL\_COMMANDS\_H  
#define SP\_PROJ1\_SHELL\_COMMANDS\_H  
  
/\* 포함되는 파일 \*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <dirent.h>  
#include <sys/stat.h>  
  
#include "utils.h"  
  
  
/\* 사용자 정의 data type \*/  
typedef struct CMD {  
 char cmd[100];  
 struct CMD \*nxt;  
} CMD;  
  
/\* 전역 변수 \*/  
CMD\* HEAD;  
CMD\* TAIL;  
  
/\* 함수 원형 \*/  
void help();  
void dir();  
void save\_instructions(char\* refined\_cmd);  
void history();  
void free\_log\_of\_instructions();  
OK\_or\_ERR type(char\* filename);  
  
#endif //SP\_PROJ1\_SHELL\_COMMANDS\_H

## 5.8 shell\_commands.c

//  
// Created by 김주호 on 2021/03/12.  
//  
  
#include "shell\_commands.h"  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : help\*/  
/\*목적 : 사용자에게 해당 프로그램에 명령할 수 있는 옵션들이 어떤 것들이 있는지 화면에 print하여 알려준다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void help() {  
 printf("h[elp]\n");  
 printf("d[ir]\n");  
 printf("q[uit]\n");  
 printf("hi[story]\n");  
 printf("du[mp] [start, end]\n");  
 printf("e[dit] address, value\n");  
 printf("f[ill] start, end, value\n");  
 printf("reset\n");  
 printf("opcode mnemonic\n");  
 printf("opcodelist\n");  
 printf("assemble filename\n");  
 printf("type filename\n");  
 printf("symbol\n");  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : dir\*/  
/\*목적 : 현재 directory에 있는 파일들을 출력한다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void dir() {  
 DIR\* pwd = opendir(".");  
 struct dirent\* pwd\_info = NULL;  
 struct stat stat\_info;  
  
 if (pwd != NULL) {  
 pwd\_info = readdir(pwd);  
 while (pwd\_info != NULL) {  
 printf("%s", pwd\_info->d\_name);  
 lstat(pwd\_info->d\_name, &stat\_info);  
 if (stat\_info.st\_mode & S\_IFREG) {  
 if (stat\_info.st\_mode & S\_IXUSR || stat\_info.st\_mode & S\_IXGRP || stat\_info.st\_mode & S\_IXOTH) {  
 printf("\*\n");  
 }  
 else {  
 printf("\n");  
 }  
 }  
 else if (stat\_info.st\_mode & S\_IFDIR) {  
 printf("/\n");  
 }  
 pwd\_info = readdir(pwd);  
 }  
 }  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : save\_instructions\*/  
/\*목적 : 사용자가 입력한 command들을 linked list에 저장한다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void save\_instructions(char \*refined\_cmd) {  
 // create new\_node  
 CMD \*new\_node = malloc(sizeof(CMD));  
 strcpy(new\_node->cmd, refined\_cmd);  
 new\_node->nxt = NULL;  
  
 // connect  
 if (HEAD == NULL) {  
 HEAD = TAIL = new\_node;  
 }   
 else {  
 TAIL->nxt = new\_node;  
 TAIL = new\_node;  
 }  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : history\*/  
/\*목적 : 현재까지 사용자가 입력한 명령어들을 순서대로 번호와 함께 보여준다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void history() {  
 CMD \*cur\_node;  
 int cnt = 1;  
 for (cur\_node = HEAD; cur\_node != NULL; cur\_node = cur\_node->nxt) {  
 printf("%4d %s\n", cnt++, cur\_node->cmd);  
 }  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : free\_log\_of\_instructions\*/  
/\*목적 : 명령어들을 저장하고 있는 linked list의 메모리 할당을 해제한다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void free\_log\_of\_instructions() {  
 CMD \*tmp\_node;  
 if (HEAD == TAIL) {  
 return;  
 }  
 while (HEAD != NULL) {  
 tmp\_node = HEAD->nxt;  
 free(HEAD);  
 HEAD = tmp\_node;  
 }  
 TAIL = NULL;  
}  
  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : type\*/  
/\*목적 : filename에 해당하는 파일을 현재 디렉터리에서 읽어서 화면에 출력한다.\*/  
/\*리턴값 : OK - 성공, FILE\_ERR - 파일 읽기 에러\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR type(char \*filename) {  
 DIR \*dir = opendir(filename);  
 FILE \*fp = fopen(filename, "r");  
 if (dir) return FILE\_ERR;  
 if (!fp) return FILE\_ERR;  
  
 char ch;  
 while (1) {  
 ch = fgetc(fp);  
 if (ch == EOF) break;  
 printf("%c", ch);  
 }  
 printf("\n");  
  
 fclose(fp);  
 return OK;  
}

## 5.9 utils.h

//  
// Created by 김주호 on 2021/03/12.  
//  
  
#ifndef SP\_PROJ1\_UTILS\_H  
#define SP\_PROJ1\_UTILS\_H  
  
/\* 포함되는 파일 \*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <ctype.h>  
  
#define LINE\_LEN 100  
#define NAME\_LEN 100  
#define STR\_ADDR\_LEN 20  
#define MAX\_HASH\_SIZE 20  
#define REG\_NUM 9  
#define CYN "\x1B[36m"  
#define RESET "\x1B[0m"  
  
typedef enum{  
 CMD\_ERR = -6,  
 OBJ\_CODE\_ERR = -5,  
 ASSEMBLY\_CODE\_ERR = -4,  
 FILE\_ERR,  
 RANGE\_ERR,  
 COMMA\_ERR,  
 OK = 1  
} OK\_or\_ERR;  
  
/\* 사용자 정의 data type \*/  
typedef enum {  
 NO\_COMMA = 0,  
 YES\_COMMA  
} func\_option;  
  
typedef enum INSTRUCTION {  
 \_ELSE = -1,  
 \_OPERATION = 0,  
 \_START = 1,  
 \_END = 2,  
 \_COMMENT = 3,  
 \_BASE = 4,  
 \_BYTE = 5,  
 \_WORD = 6,  
 \_RESB = 7,  
 \_RESW = 8  
} INSTRUCTION;  
  
typedef enum REG\_num {  
 non\_exist = -1,  
 regA = 0,  
 regX = 1,  
 regL = 2,  
 regB = 3,  
 regS = 4,  
 regT = 5,  
 regF = 6,  
 regPC = 8,  
 regSW = 9  
} REG\_num;  
  
/\* 전역 변수 \*/  
func\_option IS\_COMMA;  
  
/\* 함수 원형 \*/  
int get\_dx\_to\_nxt\_token(char \*start\_ptr);  
int hexstr\_to\_decint(char \*hexstr);  
int twos\_complement\_str\_to\_decint(char \*hexstr);  
int hash\_func\_by\_mnemonic(char \*string, int max\_hash\_size);  
int hash\_func\_by\_opcode(int opcode);  
INSTRUCTION line\_split(char \*line, char \*LABEL, char \*MNEMONIC, char \*OP1, char \*OP2);  
INSTRUCTION line\_split2(char \*line, int \*LOCCTR, char \*LABEL, char \*MNEMONIC, char \*OP1, char \*OP2);  
INSTRUCTION get\_instruction(char \*mnemonic);  
int get\_byte\_size(char \*BYTE\_operand);  
OK\_or\_ERR file\_open(char\* filename, FILE\*\* fp\_itm, FILE\*\* fp\_lst, FILE\*\* fp\_obj);  
REG\_num get\_REG\_num(char \*REG);  
  
  
#endif //SP\_PROJ1\_UTILS\_H

## 5.10 utils.c

//  
// Created by 김주호 on 2021/03/12.  
//  
  
#include "utils.h"  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : get\_dx\_to\_nxt\_token\*/  
/\*목적 : 현 token의 첫번째 index와 next token의 첫번째 index와의 차이를 계산한다\*/  
/\*리턴값 : dx - 현 token의 첫번째 index와 next token의 첫번째 index와의 차이\*/  
/\*------------------------------------------------------------------------------------\*/  
int get\_dx\_to\_nxt\_token(char \*start\_ptr) {  
 int dx = 0;  
 if (start\_ptr[dx] == '\0') return -1;  
 for (dx = 0;  
 start\_ptr[dx] != ' ' && start\_ptr[dx] != '\t' && start\_ptr[dx] != '\0' && start\_ptr[dx] != ',';  
 dx++);  
  
 if (start\_ptr[dx] == '\0') {  
 IS\_COMMA = NO\_COMMA;  
 return dx;  
 }  
  
 if (start\_ptr[dx] == ',') {  
 IS\_COMMA = YES\_COMMA;  
 }  
 else { // start\_ptr[dx] == ' ' or start\_ptr[dx] == '\t' or start\_ptr[dx] == '\0'  
 IS\_COMMA = NO\_COMMA;  
 }  
  
 start\_ptr[dx++] = '\0'; // token 끝에 NULL을 삽입  
 for (; start\_ptr[dx] == ' ' || start\_ptr[dx] == '\t'; dx++); // 다음 토큰의 첫 ind까지 접근  
 return dx;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : hexstr\_to\_decint\*/  
/\*목적 : 문자열 hexstr을 int형 decimal 형태로 바꾼다\*/  
/\*리턴값 : res - 문자열 hexstr을 int형 decimal 형태로 바꾼 결과\*/  
/\*------------------------------------------------------------------------------------\*/  
int hexstr\_to\_decint(char \*hexstr) {  
 int cur, res = 0;  
 int scale = 1;  
 for (int i = strlen(hexstr) - 1; i >= 0; i--) {  
 if (hexstr[i] >= '0' && hexstr[i] <= '9') cur = hexstr[i] - '0';  
 else if (hexstr[i] >= 'A' && hexstr[i] <= 'F') cur = hexstr[i] - 'A' + 10;  
 else if (hexstr[i] >= 'a' && hexstr[i] <= 'f') cur = hexstr[i] - 'a' + 10;  
 else return RANGE\_ERR; // contain wrong hexstr symbol  
  
 res += cur \* scale;  
 scale \*= 16;  
 }  
 return res;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : twos\_complement\_str\_to\_decint\*/  
/\*목적 : 주어진 hex str에 대한 유효성 검사와 문자열 hex str을 signed 32bit hex 표현으로 간주하고,  
 \* 이에 조응하는 dec int값으로 변환한다.\*/  
/\*리턴값 : res - 문자열 hexstr을 int형 signed 32bit 표현으로 바꾼 결과\*/  
/\*------------------------------------------------------------------------------------\*/  
int twos\_complement\_str\_to\_decint(char \*hexstr) {  
 int res;  
 // 주어진 hexstr을 decimal int 값로 바꾸며, 유효성 검사 실  
 res = hexstr\_to\_decint(hexstr);  
 if (res == RANGE\_ERR) return RANGE\_ERR; // range err!  
  
 // ascii('0') == 48 and ascii('7') == 55  
 // 주어진 hexstr 값이 양수라면, 값을 그대로 return  
 if (hexstr[0] >= 48 && hexstr[0] <= 55) return res;  
  
 // 주어진 hexstr 값이 음수라면, 값을 2의 보수표현으로 읽어 return  
 // 주어진 hexstr bit 앞에 FF를 붙여 음수 bit가 반영된 32bit 수로 변환한다.  
 return res | 0xFF000000;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : hash\_func\_by\_mnemonic\*/  
/\*목적 : string을 입력 받아 이를 0부터 19까지 20개의 숫자에 mapping한다. 입력 받은 명령어를 구성하는 모든 문자의  
 \* ASCII 값을 더하여, 20으로 나눈다.\*/  
/\*리턴값 : total % max\_hash\_size\*/  
/\*------------------------------------------------------------------------------------\*/  
int hash\_func\_by\_mnemonic(char \*string, int max\_hash\_size) {  
 int total = 0;  
 for (int i = 0; i < (int)strlen(string); i++) total += string[i];  
 return total % max\_hash\_size;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : hash\_func\_by\_opcode\*/  
/\*목적 : int형 opcode를 이를 0부터 19까지 20개의 숫자에 mapping한다.\*/  
/\*리턴값 : total % max\_hash\_size\*/  
/\*------------------------------------------------------------------------------------\*/  
int hash\_func\_by\_opcode(int opcode) {  
 return opcode % MAX\_HASH\_SIZE;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : line\_split\*/  
/\*목적 : line을 char 포인터로 입력받아 parsing하여 line에서 LABEL, MNEMONIC, OP1, OP2를 추출하고 정하여 caller에게 알려준다.\*/  
/\*리턴값 : type(어떤 유형의 line인지) \*/  
/\*------------------------------------------------------------------------------------\*/  
INSTRUCTION line\_split(char \*line, char \*LABEL, char \*MNEMONIC, char \*OP1, char \*OP2) {  
 char buf[LINE\_LEN];  
 char \*ptr;  
 INSTRUCTION type;  
  
 // buf, LABEL, MNEMONIC, OP1, OP2 초기화  
 strcpy(buf, "\0"); strcpy(buf, line);  
 strcpy(LABEL, "\0"); strcpy(MNEMONIC, "\0"); strcpy(OP1, "\0"); strcpy(OP2, "\0");  
  
 ptr = strtok(buf, " \t\r");  
 if (!ptr) return \_ELSE;  
 if (!(\*ptr)) return \_ELSE;  
  
 // line이 주석이라면 바로 return.  
 if (buf[0] == '.') {  
 strcpy(LABEL, ptr);  
 ptr = strtok(NULL, " \t\r");  
 if (!ptr) return \_COMMENT;  
  
 \*(ptr + strlen(ptr)) = ' ';  
 strcpy(MNEMONIC, ptr);  
 return \_COMMENT;  
 }  
  
 // line에서 LABEL 추출  
 if (isalpha(buf[0])) {  
 strcpy(LABEL, ptr);  
 ptr = strtok(NULL, " \t\r");  
 }  
 // line에서 MNEMONIC 추출  
 strcpy(MNEMONIC, ptr);  
 type = get\_instruction(MNEMONIC);  
  
 // line에서 OP1 추출  
 ptr = strtok(NULL, " \t\r");  
 if (!ptr) return type;  
 strcpy(OP1, ptr);  
  
 // line에서 OP2 추출  
 ptr = strtok(NULL, " \t\r");  
 if (!ptr) return type;  
 strcpy(OP2, ptr);  
  
 return type;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : line\_split2\*/  
/\*목적 : line\_split 함수의 parsing 기능을 수행하면서 추가로 LOCCTR 값을 decimal int 값으로 저장한다\*/  
/\*리턴값 : line의 종류\*/  
/\*------------------------------------------------------------------------------------\*/  
INSTRUCTION line\_split2(char \*line, int \*LOCCTR, char \*LABEL, char \*MNEMONIC, char \*OP1, char \*OP2) {  
 char tmp[LINE\_LEN];  
 char \*ptr;  
  
 strcpy(tmp, line);  
 ptr = strtok(tmp, " \t");  
 if (!ptr) return \_ELSE;  
 if (!(\*ptr)) return \_ELSE;  
 \*LOCCTR = hexstr\_to\_decint(ptr);  
  
 return line\_split(ptr + strlen(ptr) + 1, LABEL, MNEMONIC, OP1, OP2);  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : get\_instruction\*/  
/\*목적 : mnemonic에 해당하는 line type을 알려주는 함수\*/  
/\*리턴값 : mnemonic의 type\*/  
/\*------------------------------------------------------------------------------------\*/  
INSTRUCTION get\_instruction(char \*mnemonic) {  
 INSTRUCTION type;  
 if (strcmp(mnemonic, "START") == 0) type = \_START;  
 else if (strcmp(mnemonic, "END") == 0) type = \_END;  
 else if (strcmp(mnemonic, "COMMENT") == 0) type = \_COMMENT;  
 else if (strcmp(mnemonic, "BASE") == 0) type = \_BASE;  
 else if (strcmp(mnemonic, "BYTE") == 0) type = \_BYTE;  
 else if (strcmp(mnemonic, "WORD") == 0) type = \_WORD;  
 else if (strcmp(mnemonic, "RESB") == 0) type = \_RESB;  
 else if (strcmp(mnemonic, "RESW") == 0) type = \_RESW;  
 else type = \_OPERATION;  
  
 return type;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : get\_byte\_size\*/  
/\*목적 : BYTE 명령의 operand에 있는 상수가 몇 byte인지 계산하는 함수 \*/  
/\*리턴값 : byte size 정수값\*/  
/\*------------------------------------------------------------------------------------\*/  
int get\_byte\_size(char \*BYTE\_operand) {  
 int byte\_size = 0;  
 char \*ptr;  
 if (BYTE\_operand[0] == 'C') {  
 ptr = strtok(BYTE\_operand, "C`'");  
 byte\_size = (int) strlen(ptr);  
 }  
 else if (BYTE\_operand[0] == 'X') {  
 ptr = strtok(BYTE\_operand, "X`'");  
 byte\_size = (int) strlen(ptr) / 2;  
 }  
 return byte\_size;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : file\_open\*/  
/\*목적 : fp\_itm은 읽기모드로, fp\_lst, fp\_obj는 쓰기모드로 파일을 여는 함 \*/  
/\*리턴값 : OK - 파일 열기 성공, FILE\_ERR - 파일 열기 실패\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR file\_open(char\* filename, FILE\*\* fp\_itm, FILE\*\* fp\_lst, FILE\*\* fp\_obj){  
 char target\_file[NAME\_LEN];  
  
 // intermediate 파일을 읽기모드로 open  
 strcpy(target\_file, filename);  
 strcat(target\_file, ".itm");  
 \*fp\_itm = fopen(target\_file, "r");  
 if (!(\*fp\_itm)) return FILE\_ERR;  
  
 // lst 파일을 쓰기모드로 open  
 strcpy(target\_file, filename);  
 \*fp\_lst = fopen(strcat(target\_file, ".lst"), "w");  
 if (!(\*fp\_lst)) return FILE\_ERR;  
  
 // obj 파일을 쓰기모드로 open  
 strcpy(target\_file, filename);  
 \*fp\_obj = fopen(strcat(target\_file, ".obj"), "w");  
 if (!(\*fp\_obj)) return FILE\_ERR;  
  
 return OK;  
}  
  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : get\_REG\_num\*/  
/\*목적 : string으로 주어지는 register에 대하여 해당하는 register number를 알려준다\*/  
/\*리턴값 : register number\*/  
/\*------------------------------------------------------------------------------------\*/  
REG\_num get\_REG\_num(char \*REG) {  
 switch (hash\_func\_by\_mnemonic(REG, MAX\_HASH\_SIZE)) {  
 case 5:  
 return regA;  
 case 8:  
 return regX;  
 case 16:  
 return regL;  
 case 6:  
 return regB;  
 case 3:  
 return regS;  
 case 4:  
 return regT;  
 case 10:  
 if (strcmp(REG, "F") == 0) return regF;  
 else if (strcmp(REG, "SW") == 0) return regSW;  
 else break;  
 case 7:  
 return regPC;  
 default:  
 return non\_exist;  
 }  
 return non\_exist;  
}

## 5.11 opcode\_table\_commands.h

//  
// Created by 김주호 on 2021/03/17.  
//  
  
#ifndef SP\_PROJ1\_OPCODE\_TABLE\_COMMANDS\_H  
#define SP\_PROJ1\_OPCODE\_TABLE\_COMMANDS\_H  
  
/\* 포함되는 파일 \*/  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
  
#include "utils.h"  
  
// #define MAX\_OPCODE\_SIZE 3  
#define MAX\_INSTRUCTION\_SIZE 10  
#define MAX\_FORMAT\_SIZE 10  
#define MAX\_HASHTABLE\_SIZE 20  
  
/\* 사용자 정의 data type\*/  
typedef struct OP\_NODE {  
 int opcode;  
 char mnemonic[MAX\_INSTRUCTION\_SIZE];  
 char format[MAX\_FORMAT\_SIZE];  
 struct OP\_NODE \*nxt\_by\_mnemonic;  
 struct OP\_NODE \*nxt\_by\_opcode;  
} OP\_NODE;  
  
/\* 전역 변수 \*/  
OP\_NODE\* OPTAB\_by\_mnemonic[MAX\_HASHTABLE\_SIZE];  
OP\_NODE\* OPTAB\_by\_opcode[MAX\_HASHTABLE\_SIZE];  
  
/\* 함수 원형 \*/  
OK\_or\_ERR print\_opcode\_by\_mnemonic(char \*mnemonic);  
OP\_NODE \*get\_opcode\_or\_NULL\_by\_mnemonic(char \*mnemonic);  
OP\_NODE \*get\_opcode\_or\_NULL\_by\_opcode(int opcode);  
void init\_OPTAB(char \*filename);  
void opcodelist();  
void free\_OPTAB();  
  
  
#endif //SP\_PROJ1\_OPCODE\_TABLE\_COMMANDS\_H

## 5.12 opcode\_table\_commands.c

//  
// Created by 김주호 on 2021/03/17.  
//  
  
#include "opcode\_table\_commands.h"  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : init\_OPTAB\*/  
/\*목적 : opcode 정보들이 적힌 file을 읽어서 해당파일을 열고, 이를 프로그램에서 선언한 hash table에 저장한다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void init\_OPTAB(char \*filename) {  
 int ind\_by\_mnemonic\_hash, ind\_by\_opcode\_hash;  
 FILE \*fp = fopen(filename, "r");  
 OP\_NODE \*new\_node = malloc(sizeof(OP\_NODE));  
  
 for (int i = 0; i < MAX\_HASHTABLE\_SIZE; i++) {  
 OPTAB\_by\_mnemonic[i] = NULL;  
 OPTAB\_by\_opcode[i] = NULL;  
 }  
 while (1) {  
 if (fscanf(fp, "%X%s%s", &(new\_node->opcode), new\_node->mnemonic, new\_node->format) == EOF) break;  
 ind\_by\_mnemonic\_hash = hash\_func\_by\_mnemonic(new\_node->mnemonic, MAX\_HASHTABLE\_SIZE);  
 ind\_by\_opcode\_hash = hash\_func\_by\_opcode(new\_node->opcode);  
  
 // connect  
 new\_node->nxt\_by\_mnemonic = OPTAB\_by\_mnemonic[ind\_by\_mnemonic\_hash];  
 new\_node->nxt\_by\_opcode = OPTAB\_by\_opcode[ind\_by\_opcode\_hash];  
  
 OPTAB\_by\_mnemonic[ind\_by\_mnemonic\_hash] = new\_node;  
 OPTAB\_by\_opcode[ind\_by\_opcode\_hash] = new\_node;  
 // malloc  
 new\_node = malloc(sizeof(OP\_NODE));  
 }  
 free(new\_node);  
 fclose(fp);  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : print\_opcode\_by\_mnemonic\*/  
/\*목적 : mnemonic을 입력 받아 해당 mnemonic에 맞는 opcode를 알려준다.\*/  
/\*리턴값 : OK - 성공인 경우, ERR - 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR print\_opcode\_by\_mnemonic(char \*mnemonic) {  
 const int OK = 1;  
 const int ERR = 0;  
 OP\_NODE \*cur\_node = OPTAB\_by\_mnemonic[hash\_func\_by\_mnemonic(mnemonic, MAX\_HASHTABLE\_SIZE)];  
 while (cur\_node != NULL) {  
 if (strcmp(cur\_node->mnemonic, mnemonic) == 0) {  
 printf("opcode is %02X\n", cur\_node->opcode);  
 return OK;  
 }  
 cur\_node = cur\_node->nxt\_by\_mnemonic;  
 }  
 printf("err: no opcode for the mnemonic %s\n", mnemonic);  
 return ERR;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : get\_opcode\_or\_NULL\_by\_opcode\*/  
/\*목적 : opcode를 입력 받아 해당 opcode에 맞는 OP\_node를 return 한다.\*/  
/\*리턴값 : OK - 성공인 경우, ERR - 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OP\_NODE \*get\_opcode\_or\_NULL\_by\_opcode(int opcode) {  
 int ind = hash\_func\_by\_opcode(opcode);  
 for(OP\_NODE \*cur\_node = OPTAB\_by\_opcode[ind]; cur\_node; cur\_node = cur\_node->nxt\_by\_opcode){  
 if(cur\_node->opcode == opcode){  
 return cur\_node;  
 }  
 }  
 return NULL;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : get\_opcode\_or\_NULL\_by\_mnemonic\*/  
/\*목적 : 주어진 mnemonic에 맞는 opcode를 return하는 함수이다\*/  
/\*리턴값 : 주어진 mnemonic이 opcode table에 있다면 해당 node를 리턴, 없다면 NULL을 리턴\*/  
/\*------------------------------------------------------------------------------------\*/  
OP\_NODE \*get\_opcode\_or\_NULL\_by\_mnemonic(char \*mnemonic) {  
 OP\_NODE \*cur\_node = OPTAB\_by\_mnemonic[hash\_func\_by\_mnemonic(mnemonic, MAX\_HASHTABLE\_SIZE)];  
 while (cur\_node != NULL) {  
 if (strcmp(cur\_node->mnemonic, mnemonic) == 0) {  
 return cur\_node;  
 }  
 cur\_node = cur\_node->nxt\_by\_mnemonic;  
 }  
 return NULL;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : opcodelist\*/  
/\*목적 : hash table의 모든 index를 방문하면서, 해당 index에 저장되어있는 모든 node들을 형식에 맞게 출력한다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void opcodelist() {  
 OP\_NODE \*cur\_node;  
 for (int i = 0; i < MAX\_HASHTABLE\_SIZE; i++) {  
 printf("%d : ", i);  
 cur\_node = OPTAB\_by\_mnemonic[i];  
 if (cur\_node == NULL) {  
 printf("\n");  
 continue;  
 }  
 printf("[%s,%02X]", cur\_node->mnemonic, cur\_node->opcode);  
 cur\_node = cur\_node->nxt\_by\_mnemonic;  
  
 while (cur\_node != NULL){  
 printf(" -> ");  
 printf("[%s,%02X]", cur\_node->mnemonic, cur\_node->opcode);  
 cur\_node = cur\_node->nxt\_by\_mnemonic;  
 }  
 printf("\n");  
 }  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : free\_OPTAB\*/  
/\*목적 : hash table에 저장된 모든 node들의 할당을 해제한다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void free\_OPTAB(){  
 OP\_NODE \*pre\_node;  
 for (int i=0; i<MAX\_HASHTABLE\_SIZE; i++){  
 for (OP\_NODE \*cur\_node = OPTAB\_by\_mnemonic[i]; cur\_node;){  
 pre\_node = cur\_node;  
 cur\_node = cur\_node->nxt\_by\_mnemonic;  
 free(pre\_node);  
 }  
 OPTAB\_by\_mnemonic[i] = NULL;  
 }  
 for (int i = 0; i < MAX\_HASH\_SIZE; i++) OPTAB\_by\_opcode[i] = NULL;  
}

## 5.13 loader\_commands.h

//  
// Created by 김주호 on 2021/04/17.  
//  
  
#ifndef ASSEMBLER\_COMMANDS\_C\_LOADER\_H  
#define ASSEMBLER\_COMMANDS\_C\_LOADER\_H  
  
#include "memory\_commands.h"  
#include "utils.h"  
#include "assembler\_commands.h"  
  
#define MAX\_FILES\_NUM 5  
#define MAX\_FILENAME\_LEN 15  
#define ESTAB\_HASH\_SIZE 20  
#define MAX\_RF\_NUM 100  
  
typedef struct ES\_NODE{  
 char name[SYMBOL\_LEN];  
 int addr;  
 struct ES\_NODE\* nxt;  
}ES\_NODE;  
ES\_NODE \*ESTAB[ESTAB\_HASH\_SIZE];  
  
// bp가 걸려있는 memory cell의 주소에 true(1) 값을 저장시켜 bp가 걸려있음을 표시한다.  
int BP\_CHK[MEM\_SIZE];  
int BP\_ADDR[MEM\_SIZE];  
int PRINT\_FLAG;  
int NUM\_OF\_BP;  
  
// 레지스터와 프로그램 address와 길이 관리 변수  
int REG[REG\_NUM + 1];  
char CC; // Condition Code  
int PROGRAM\_ADDR;  
int PROGRAM\_LEN;  
int CS\_ADDR;  
int CS\_LEN;  
int FIRST\_INSTRUCTION\_ADDR;  
  
// func about loader instruction  
OK\_or\_ERR set\_PROGADDR(char\* addr\_hexstr);  
OK\_or\_ERR loader(char filenames[MAX\_FILES\_NUM][MAX\_FILENAME\_LEN]);  
OK\_or\_ERR load\_pass1(FILE \*fp);  
OK\_or\_ERR load\_pass2(FILE \*fp);  
  
// func about ESTAB instruction  
OK\_or\_ERR push\_to\_ESTAB(char \*es\_name, int es\_addr);  
ES\_NODE \*find\_ESNODE\_or\_NULL(char \*es\_name);  
void free\_ESTAB();  
  
// func about BP and run instruction  
OK\_or\_ERR bp\_command(int num\_of\_tokens, char \*addr\_hexstr\_or\_claer\_str);  
OK\_or\_ERR run();  
OK\_or\_ERR execute\_instructions();  
int LD\_related\_instruction(int ni, int TA, int format, int num\_of\_bytes);  
int J\_related\_instruction(int ni, int TA, int format);  
void ST\_related\_instruction(int ni, int TA, int val\_to\_push, int num\_of\_bytes);  
  
#endif //ASSEMBLER\_COMMANDS\_C\_LOADER\_H

## 5.14 loader\_commands.c

//  
// Created by 김주호 on 2021/04/17.  
//  
  
#include "loader\_commands.h"  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : set\_PROGADDR\*/  
/\*목적 : 사용자로부터 입력받은 address로 program이 load될 메모리 시작주소 위치를 set한다.\*/  
/\*리턴값 : OK - 성공인 경우, ERR - 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR set\_PROGADDR(char \*addr\_hexstr) {  
 int addr\_int;  
  
 addr\_int = hexstr\_to\_decint(addr\_hexstr);  
  
 // address에 대한 유효성 검사  
 if (addr\_int == RANGE\_ERR) { // 적절하지 않은 형식의 address가 입력된 경우  
 printf("range err!\n");  
 return RANGE\_ERR;  
 }  
 if (addr\_int < 0 || addr\_int >= MEM\_SIZE) { // 입력된 address가 메모리 주소 범위를 초과한 경우  
 printf("range err!\n");  
 return RANGE\_ERR;  
 }  
  
 // 사용자로부터 입력 받은 address를  
 // 1. program의 address로 설정  
 // 2. PC 초기값으로 설정  
 PROGRAM\_ADDR = REG[regPC] = addr\_int;  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : loader\*/  
/\*목적 : pass1과 pass2 알고리즘을 call 하여, ES table을 생성(pass1)하고, 이를 이용한 load 작업을  
 \* 수행(pass2)한다.\*/  
/\*리턴값 : OK - 성공인 경우, ERR - 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR loader(char filenames[MAX\_FILES\_NUM][MAX\_FILENAME\_LEN]) {  
 FILE \*fp;  
  
 // loader()를 호출하면, 먼저 이전에 저장된 ESTAB을 할당 해제한다.  
 free\_ESTAB();  
  
 // \*\*\*\*\*\*\*\*\*\* pass1 \*\*\*\*\*\*\*\*\*\*\*  
 printf("%-7s %-7s %-7s %-7s\n", "control", "symbol", "address", "length");  
 printf("%-7s %-7s\n", "section", "name");  
 printf("-----------------------------------\n");  
 // pass1 진입 이전에 초기조건 설정  
 PROGRAM\_LEN = 0;  
 CS\_ADDR = FIRST\_INSTRUCTION\_ADDR = PROGRAM\_ADDR;  
  
 // 사용자로부터 입력받은 모든 obj 파일에 대하여 pass1 알고리즘을 수행  
 char \*obj\_file = filenames[1];  
 for (int i = 2; obj\_file != NULL && \*obj\_file != '\0'; i++) {  
 fp = fopen(obj\_file, "r");  
 if (!fp) {  
 printf("err! no file to open\n");  
 return FILE\_ERR;  
 }  
  
 load\_pass1(fp);  
 fclose(fp);  
 obj\_file = filenames[i];  
 }  
 printf("-----------------------------------\n");  
 printf("%-7s %-7s %-7s %04X \n", " ", " total", "length", PROGRAM\_LEN);  
  
  
 // \*\*\*\*\*\*\*\*\*\* pass2 \*\*\*\*\*\*\*\*\*\*\*  
 // pass2 진입 이전에 초기조건 설정  
 CS\_ADDR = FIRST\_INSTRUCTION\_ADDR = PROGRAM\_ADDR;  
  
 // 입력받은 모든 obj 파일에 대하여 pass2 알고리즘을 수행  
 obj\_file = filenames[1];  
 for (int i = 2; obj\_file != NULL && \*obj\_file != '\0'; i++) {  
 fp = fopen(obj\_file, "r");  
 if (!fp) {  
 printf("wrong object code!\n");  
 return FILE\_ERR;  
 }  
  
 if (load\_pass2(fp) != OK) {  
 fclose(fp);  
 return FILE\_ERR;  
 }  
 fclose(fp);  
 obj\_file = filenames[i];  
 }  
  
 // reset REG  
 for (int i = 0; i < REG\_NUM; i++) REG[i] = 0;  
  
 // 프로그램의 길이를 return address로 한다. (0번지에 sub program을 올린다는 가정)  
 REG[regL] = PROGRAM\_LEN;  
 REG[regPC] = FIRST\_INSTRUCTION\_ADDR;  
 PRINT\_FLAG = 0;  
  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : load\_pass1\*/  
/\*목적 : object file을 열어 H 레코드와 D 레코드를 참조하여, ES table을 완성한다.\*/  
/\*리턴값 : OK - 성공인 경우, ERR - 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR load\_pass1(FILE \*fp) {  
 char \*ptr;  
 char line[LINE\_LEN];  
  
 while (!feof(fp)) {  
 fgets(line, LINE\_LEN, fp);  
 if (line[0] == 'H') {  
 char CS\_name[SYMBOL\_LEN];  
 char CS\_start\_addr\_hexstr[STR\_ADDR\_LEN];  
 char CS\_len\_hexstr[STR\_ADDR\_LEN];  
 int CS\_start\_addr\_int;  
  
 line[strlen(line) - 1] = '\0';  
 ptr = strtok(line, " \r");  
 ptr++;  
  
 // get CS\_name  
 strncpy(CS\_name, ptr, 6);  
 CS\_name[6] = '\0';  
 strtok(CS\_name, " ");  
 ptr += 6;  
  
 // get starting address  
 strncpy(CS\_start\_addr\_hexstr, ptr, 6);  
 CS\_start\_addr\_hexstr[6] = '\0';  
 CS\_start\_addr\_int = hexstr\_to\_decint(CS\_start\_addr\_hexstr);  
 if (push\_to\_ESTAB(CS\_name, CS\_ADDR + CS\_start\_addr\_int) == OBJ\_CODE\_ERR) {  
 free\_ESTAB();  
 printf("CS name duplicate!\n");  
 return OBJ\_CODE\_ERR;  
 }  
  
 // get Control Section 길이  
 strncpy(CS\_len\_hexstr, ptr + 6, 6);  
 CS\_len\_hexstr[6] = '\0';  
 CS\_LEN = hexstr\_to\_decint(CS\_len\_hexstr);  
 // PROGRAM 길이에 CS 길이를 누적시킴  
 PROGRAM\_LEN += CS\_LEN;  
  
 printf("%-7s %-7s %04X %04X \n", CS\_name, " ", CS\_ADDR + CS\_start\_addr\_int, CS\_LEN);  
 }  
 else if (line[0] == 'D') {  
 char SYM\_name[SYMBOL\_LEN];  
 char sym\_addr\_hexstr[STR\_ADDR\_LEN];  
 int sym\_addr\_int;  
  
 line[strlen(line) - 1] = '\0';  
 ptr = strtok(line, " \r");  
 ptr++;  
 while (1) {  
 // get name of symbol  
 strncpy(SYM\_name, ptr, 6);  
 SYM\_name[6] = '\0';  
 strtok(SYM\_name, " ");  
 ptr += 6;  
  
 // get address of symbol and push to ESTAB  
 strncpy(sym\_addr\_hexstr, ptr, 6);  
 sym\_addr\_hexstr[6] = '\0';  
 sym\_addr\_int = hexstr\_to\_decint(sym\_addr\_hexstr);  
 if (push\_to\_ESTAB(SYM\_name, CS\_ADDR + sym\_addr\_int) == OBJ\_CODE\_ERR) {  
 free\_ESTAB();  
 printf("symbol name duplicate!\n");  
 return OBJ\_CODE\_ERR;  
 }  
  
 printf("%-7s %-7s %04X \n", " ", SYM\_name, CS\_ADDR + sym\_addr\_int);  
 ptr += 6;  
 if (\*(ptr) == '\0') break;  
 }  
 }  
 else if (line[0] == 'E') {  
 // CS address를 다음 program이 load될 주소로 조정해둔다.  
 CS\_ADDR += CS\_LEN;  
 break;  
 }  
 }  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : load\_pass2\*/  
/\*목적 : pass1에서 완성한 ES table과 R, T, M 레코드를 참조하여 memory에 object 코드들을 적재한다\*/  
/\*리턴값 : OK - 성공인 경우, ERR - 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR load\_pass2(FILE \*fp) {  
 char \*ptr;  
 char line[MAX\_LINE\_NUM];  
 int RFTAB[MAX\_RF\_NUM];  
  
 while (!feof(fp)) {  
 fgets(line, LINE\_LEN, fp);  
 if (line[0] == 'H') {  
 char CS\_name[NAME\_LEN];  
  
 line[strlen(line) - 1] = '\0';  
 ptr = strtok(line, " \r");  
 ptr++;  
  
 // get CS\_name  
 strncpy(CS\_name, ptr, 6);  
 CS\_name[6] = '\0';  
 strtok(CS\_name, " ");  
  
 // CS\_name의 address를 RFTAB[1]에 저장한다.  
 ES\_NODE \*cs\_node = find\_ESNODE\_or\_NULL(CS\_name);  
 CS\_ADDR = RFTAB[1] = cs\_node->addr;  
 }  
 else if (line[0] == 'R') {  
 char rf\_ind\_hexstr[NAME\_LEN];  
 int rf\_ind;  
 char ES\_name[NAME\_LEN];  
  
 line[strlen(line) - 1] = '\0';  
 ptr = strtok(line, " \r");  
 ptr++;  
  
 // RFTAB[rf\_ind]에 해당 symbol의 address를 저장하는 작업  
 for (; ptr; ptr = strtok(NULL, " \r")) {  
 strncpy(rf\_ind\_hexstr, ptr, 2);  
 rf\_ind\_hexstr[2] = '\0';  
  
 rf\_ind = hexstr\_to\_decint(rf\_ind\_hexstr);  
 if (rf\_ind != RANGE\_ERR) {  
 strncpy(ES\_name, ptr + 2, 6);  
 ES\_name[6] = '\0';  
  
 ES\_NODE \*es\_node = find\_ESNODE\_or\_NULL(ES\_name);  
 if (!es\_node) {  
 free\_ESTAB();  
 printf("err! check ES table\n");  
 return OBJ\_CODE\_ERR;  
 }  
 RFTAB[rf\_ind] = es\_node->addr;  
 }  
 else {  
 free\_ESTAB();  
 printf("err! wrong reference number\n");  
 return OBJ\_CODE\_ERR;  
 }  
 }  
 }  
 else if (line[0] == 'T') {  
 char starting\_addr\_hexstr[STR\_ADDR\_LEN];  
 char one\_line\_T\_record\_len\_hexstr[STR\_ADDR\_LEN];  
 char byte\_hexstr[STR\_ADDR\_LEN];  
 int one\_line\_T\_record\_len\_int;  
 int byte\_int;  
 int LOC\_to\_push;  
  
 line[strlen(line) - 1] = '\0';  
 ptr = strtok(line, " \r");  
 ptr++;  
  
 // get "obj\_code가 올라갈 MEM의 시작주소"  
 strncpy(starting\_addr\_hexstr, ptr, 6);  
 starting\_addr\_hexstr[6] = '\0';  
  
 LOC\_to\_push = CS\_ADDR + hexstr\_to\_decint(starting\_addr\_hexstr);  
 ptr += 6;  
  
 // get "LEN of one line T record"  
 strncpy(one\_line\_T\_record\_len\_hexstr, ptr, 2);  
 one\_line\_T\_record\_len\_hexstr[2] = '\0';  
 one\_line\_T\_record\_len\_int = LOC\_to\_push + hexstr\_to\_decint(one\_line\_T\_record\_len\_hexstr);  
 ptr += 2;  
  
 // T record 1byte씩 접근하여 MEM에 적재  
 for (; LOC\_to\_push <= one\_line\_T\_record\_len\_int; MEMORY[LOC\_to\_push++] = byte\_int) {  
 strncpy(byte\_hexstr, ptr, 2);  
 byte\_hexstr[2] = '\0';  
 ptr += 2;  
 byte\_int = hexstr\_to\_decint(byte\_hexstr);  
 if (byte\_int == RANGE\_ERR) {  
 printf("err! : wrong byte value!\n");  
 return RANGE\_ERR;  
 }  
 }  
 }  
 else if (line[0] == 'M') {  
 char LOC\_to\_be\_modified\_hexstr[STR\_ADDR\_LEN];  
 char halfbytes\_len\_to\_be\_modified\_hexstr[STR\_ADDR\_LEN];  
 char half\_bytes\_to\_be\_modified\_str[STR\_ADDR\_LEN];  
 int LOC\_to\_be\_modified\_int;  
 int halfbytes\_len\_to\_be\_modified\_int;  
 int rf\_ind\_int;  
 int dx; // modify를 적용하는 증분(음수일 수 있음). target object code에 dx만큼의 조정을 가하게 된다.  
  
 // get "modified될 명령의 시작 위치"  
 line[strlen(line) - 1] = '\0';  
 ptr = line + 1;  
 strncpy(LOC\_to\_be\_modified\_hexstr, ptr, 6);  
 LOC\_to\_be\_modified\_hexstr[6] = '\0';  
 LOC\_to\_be\_modified\_int = CS\_ADDR + hexstr\_to\_decint(LOC\_to\_be\_modified\_hexstr);  
 ptr += 6;  
  
 // get "수정될 half byte의 길이"  
 strncpy(halfbytes\_len\_to\_be\_modified\_hexstr, ptr, 2);  
 halfbytes\_len\_to\_be\_modified\_hexstr[2] = '\0';  
 halfbytes\_len\_to\_be\_modified\_int = hexstr\_to\_decint(halfbytes\_len\_to\_be\_modified\_hexstr);  
 ptr += 2;  
  
 // modify를 적용하는 증분을 RFTAB에서 꺼내온다.  
 dx = 0;  
 if (ptr) {  
 char rf\_ind\_or\_rf\_name[STR\_ADDR\_LEN];  
 strcpy(rf\_ind\_or\_rf\_name, ptr + 1);  
 strtok(rf\_ind\_or\_rf\_name, " \r");  
 rf\_ind\_int = hexstr\_to\_decint(rf\_ind\_or\_rf\_name);  
  
 // rf number를 이용한 M레코드가 적여있다고 가정 (ex> +02)  
 dx = RFTAB[rf\_ind\_int];  
 if (\*ptr == '-') dx = -dx;  
 }  
  
 // 수정할 obj code 3byte(=6 halfbyte)를 가져온다.  
 char tmp[STR\_ADDR\_LEN];  
 strcpy(half\_bytes\_to\_be\_modified\_str, "");  
 for (int i = 0; i < 3; i++) {  
 sprintf(tmp, "%02X", MEMORY[LOC\_to\_be\_modified\_int + i]);  
 strcat(half\_bytes\_to\_be\_modified\_str, tmp);  
 }  
  
 // dx 만큼 modified된 objcode 생성  
 int modified\_val;  
 // modify 해야할 object code가 5 half byte인 경우, 해당 부분은 주솟값이기 때문에 음의 값은 올 수 없고,  
 // hexstr -> int 작업만 수행하면 된다.  
 if (halfbytes\_len\_to\_be\_modified\_int == 5) {  
 modified\_val = hexstr\_to\_decint(half\_bytes\_to\_be\_modified\_str + 1) + dx;  
 sprintf(tmp, "%02X", MEMORY[LOC\_to\_be\_modified\_int]);  
 sprintf(half\_bytes\_to\_be\_modified\_str, "%c%05X", tmp[0], modified\_val);  
 }  
 // modify 해야할 object code가 6 half byte인 경우 해당 부분은 2의 보수표현 음수가 올 수 있어서  
 // 혹 음수가 온 경우 이를 음숫값으로 계산하여 modified value를 계산해야한다.  
 // ex) 0xFFFFFA를 16777210로 계산하면 안되고, -10이라고 계산해야한다.  
 else if (halfbytes\_len\_to\_be\_modified\_int == 6) {  
 modified\_val = twos\_complement\_str\_to\_decint(half\_bytes\_to\_be\_modified\_str) + dx;  
 sprintf(half\_bytes\_to\_be\_modified\_str, "%06X", modified\_val);  
 if (modified\_val < 0) {  
 for (int i = 0; i <= 6; i++)  
 half\_bytes\_to\_be\_modified\_str[i] = half\_bytes\_to\_be\_modified\_str[i + 2];  
 }  
 }  
  
 // 생성한 modified val를 한 byte씩 메모리에 심는다.  
 for (int i = 0; i < 3; i++) {  
 int byte\_val;  
 strncpy(tmp, half\_bytes\_to\_be\_modified\_str + 2 \* i, 2);  
 tmp[2] = '\0';  
 byte\_val = hexstr\_to\_decint(tmp);  
 MEMORY[LOC\_to\_be\_modified\_int + i] = byte\_val;  
 }  
 }  
 else if (line[0] == 'E') {  
 if (line[1] != '\0') {  
 char first\_instrcution\_addr\_str[STR\_ADDR\_LEN];  
 strncpy(first\_instrcution\_addr\_str, line + 1, 6);  
 FIRST\_INSTRUCTION\_ADDR = CS\_ADDR + hexstr\_to\_decint(first\_instrcution\_addr\_str);  
 }  
 break;  
 }  
 }  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : push\_to\_ESTAB\*/  
/\*목적 : ES table에 es node를 push하는 작업을 수행한다. ES table은 hash size 20인 hash map으로 구현되어있다.\*/  
/\*리턴값 : OK - 성공인 경우, ERR - 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR push\_to\_ESTAB(char \*es\_name, int es\_addr) {  
 int ind = hash\_func\_by\_mnemonic(es\_name, ESTAB\_HASH\_SIZE);  
 ES\_NODE \*cur\_node = ESTAB[ind];  
 ES\_NODE \*new\_node = malloc(sizeof(ES\_NODE));  
  
 // create new node  
 strcpy(new\_node->name, es\_name);  
 new\_node->addr = es\_addr;  
 new\_node->nxt = NULL;  
  
 // ES table의 해당 index 자리가 비어있는 경우 head에 삽입  
 if (cur\_node == NULL) {  
 ESTAB[ind] = new\_node;  
 return OK;  
 }  
 // 비어있지 않다면, tail 에 삽입  
 for (; cur\_node->nxt; cur\_node = cur\_node->nxt) {  
 // ES table에 이미 존재한다면, 오류  
 if (strcmp(cur\_node->name, es\_name) == 0) {  
 free(new\_node);  
 return OBJ\_CODE\_ERR;  
 }  
 }  
 cur\_node->nxt = new\_node;  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : find\_ESNODE\_or\_NULL\*/  
/\*목적 : es name을 입력받아 ES table을 탐색하여 이에 해당하는 node를 돌려준다. 해당 es name이 없다면,  
 \* NULL 값을 return한다.\*/  
/\*리턴값 : 해당 es\_node - ES table에 존재하는 경우, NULL - ES table에 es name이 존재하지 않는 경\*/  
/\*------------------------------------------------------------------------------------\*/  
ES\_NODE \*find\_ESNODE\_or\_NULL(char \*es\_name) {  
 int ind = hash\_func\_by\_mnemonic(es\_name, ESTAB\_HASH\_SIZE);  
 ES\_NODE \*cur\_node = ESTAB[ind];  
 for (; cur\_node; cur\_node = cur\_node->nxt) {  
 if (strcmp(cur\_node->name, es\_name) == 0) {  
 return cur\_node;  
 }  
 }  
 return NULL;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : free\_ESTAB\*/  
/\*목적 : ESTAB에 할당된 메모리를 해제하는 함수이다.\*/  
/\*리턴값 : 없음\*/  
/\*------------------------------------------------------------------------------------\*/  
void free\_ESTAB() {  
 ES\_NODE \*cur\_node, \*pre\_node;  
 for (int i = 0; i < ESTAB\_HASH\_SIZE; i++) {  
 cur\_node = ESTAB[i];  
 while (cur\_node) {  
 pre\_node = cur\_node;  
 cur\_node = cur\_node->nxt;  
 free(pre\_node);  
 }  
 ESTAB[i] = NULL;  
 }  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : bp\_command\*/  
/\*목적 : bp 관련한 사용자의 명령에 따라 관련 작업을 수행하고, 그 내용은 아래와 같다.  
 \* 1. bp : bp가 걸린 메모리 주소를 보여준다.  
 \* 2. bp clear : set 되어 있는 bp값들을 모두 해제한다.  
 \* 3. bp [address] : 해당 [address]에 bp를 설정한다.\*/  
/\*리턴값 : OK - 성공인 경우, ERR - 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR bp\_command(int num\_of\_tokens, char \*addr\_hexstr\_or\_claer\_str) {  
 // bp 명령  
 if (num\_of\_tokens == 1) {  
 printf("\t\tbreakpoint\n");  
 printf("\t\t----------\n");  
 for (int i = 0; i < NUM\_OF\_BP; i++) printf("\t\t%X\n", BP\_ADDR[i]);  
 }  
  
 // bp clear 명령  
 else if (num\_of\_tokens == 2 && strcmp(addr\_hexstr\_or\_claer\_str, "clear") == 0) {  
 for (int i = 0; i < NUM\_OF\_BP; i++){  
 BP\_CHK[BP\_ADDR[i]] = 0;  
 }  
 NUM\_OF\_BP = 0;  
  
 printf("\t\t[" CYN"ok"RESET "] clear all breakpoints\n");  
 }  
  
 // bp [address] 명령  
 else if (num\_of\_tokens == 2) {  
 int addr\_int = hexstr\_to\_decint(addr\_hexstr\_or\_claer\_str);  
 if (addr\_int < 0) {  
 printf("err! wrong address\n");  
 return RANGE\_ERR;  
 }  
 if (addr\_int > MEM\_SIZE) {  
 printf("err! wrong address\n");  
 return RANGE\_ERR;  
 }  
  
 // bp가 걸린 address를 자료구조에 표시해둔다.  
 // 1. MEMORY와 동일한 크기의 배열(BP\_CHK)을 정의하고 해당 메모리 주소에 true(1)를 저장시킴  
 // 2. true를 저장하고 있는 MEM 주소를 BP\_ADDR에 저장한다.  
 BP\_CHK[addr\_int] = 1;  
 BP\_ADDR[NUM\_OF\_BP++] = addr\_int;  
 printf("\t\t[" CYN"ok"RESET "] create breakpoint %s\n", addr\_hexstr\_or\_claer\_str);  
 }  
 else {  
 printf("wrong command!\n");  
 return CMD\_ERR;  
 }  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : run\*/  
/\*목적 : machine instruction을 수행시키며, break이 걸린 point들을 잡아내어 그 때마다 register의 내용을  
 \* print한다.\*/  
/\*리턴값 : OK - 성공인 경우, ERR - 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR run() {  
 int END = PROGRAM\_ADDR + CS\_LEN;  
 while (REG[regPC] < END) {  
 if (BP\_CHK[REG[regPC]] && PRINT\_FLAG) {  
 // print register  
 printf("A : %06X X : %06X\n", REG[regA], REG[regX]);  
 printf("L : %06X PC : %06X\n", REG[regL], REG[regPC]);  
 printf("B : %06X S : %06X\n", REG[regB], REG[regS]);  
 printf("T : %06X\n", REG[regT]);  
 printf("\t\tStop at checkpoint[%X]\n", REG[regPC]);  
  
 // bp 지점에서 다시 run을 수행시, 해당위치에서 register 값을 또 출력하지 않도록 조정  
 PRINT\_FLAG = 0;  
 return OK;  
 }  
 execute\_instructions();  
 PRINT\_FLAG = 1;  
 }  
 // 프로그램 run 수행이 끝나면 print register  
 printf("A : %06X X : %06X\n", REG[regA], REG[regX]);  
 printf("L : %06X PC : %06X\n", REG[regL], REG[regPC]);  
 printf("B : %06X S : %06X\n", REG[regB], REG[regS]);  
 printf("T : %06X\n", REG[regT]);  
 printf("\t\tEnd Program\n");  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : execute\_instructions\*/  
/\*목적 : 메모리에 적재된 object code들을 메모리에서 읽어들여 machine instruction을 수행시킨다.\*/  
/\*리턴값 : OK - 성공인 경우, ERR - 에러인 경우\*/  
/\*------------------------------------------------------------------------------------\*/  
OK\_or\_ERR execute\_instructions() {  
 int opcode, format;  
 int reg1, reg2;  
 int disp = 0;  
 int ni, x, b, p, e;  
 ni = x = b = p = e = 0;  
  
 int TA = 0;  
 int start\_loc = REG[regPC];  
 int operand;  
  
 OP\_NODE \*op\_node;  
  
 // 메모리에서 1 byte 읽어옴.  
 opcode = MEMORY[start\_loc];  
 // get opcode and ni  
 ni = opcode & 0b00000011;  
 opcode -= ni;  
 op\_node = get\_opcode\_or\_NULL\_by\_opcode(opcode);  
 if (!op\_node) {  
 printf("wrong object code!\n");  
 return OBJ\_CODE\_ERR;  
 }  
  
 // op\_node에 기록된 format을 읽어옴.  
 if (\*(op\_node->format) == '1') format = 1;  
 else if (\*(op\_node->format) == '2') format = 2;  
 else { // format == 3 또는 format == 4  
 e = (MEMORY[start\_loc + 1] & 0b10000) >> 4;  
 if (e == 1) format = 4;  
 else format = 3;  
 }  
 REG[regPC] += format;  
  
 // get operands  
 switch (format) {  
 case 2:  
 reg1 = (MEMORY[start\_loc + 1] & 0xF0) >> 4;  
 reg2 = MEMORY[start\_loc + 1] & 0x0F;  
 break;  
 case 3:  
 x = (MEMORY[start\_loc + 1] & 0b10000000) >> 7;  
 b = (MEMORY[start\_loc + 1] & 0b1000000) >> 6;  
 p = (MEMORY[start\_loc + 1] & 0b100000) >> 5;  
 // MEMORY[start\_loc + 1]의 오른쪽 half byte와 MEMORY[start\_loc + 2]를 결합해 disp 만듦  
 disp = ((MEMORY[start\_loc + 1] & 0x0F) << 8) + MEMORY[start\_loc + 2];  
 break;  
 case 4:  
 x = (MEMORY[start\_loc + 1] & 0b10000000) >> 7;  
 b = (MEMORY[start\_loc + 1] & 0b1000000) >> 6;  
 p = (MEMORY[start\_loc + 1] & 0b100000) >> 5;  
 // MEMORY[start\_loc + 1]의 오른쪽 half byte와 MEMORY[start\_loc + 2]와 MEMORY[start\_loc + 3]를 결합해 disp 만듦  
 disp = ((MEMORY[start\_loc + 1] & 0x0F) << 16) + (MEMORY[start\_loc + 2] << 8) + MEMORY[start\_loc + 3];  
 break;  
 default:  
 break;  
 }  
  
 // disp가 음수인 경우, 2의 보수표현 -> int 값으로 가져온다  
 // "| 0xFFFFF000"은 disp값(3 halfbyte)에 앞에 FFFFF를 붙여서, 이를 32bit 2의 보수표현으로 읽히게 하는 작업이다  
 if (format == 3 && (disp & 0x800)) disp = (int)(disp | 0xFFFFF000);  
  
 // operand 값 정하기  
 if (format == 3 || format == 4) {  
 if (b == 1 && p == 0) TA = REG[regB] + disp; // b-relative addressing  
 else if (b == 0 && p == 1) TA = REG[regPC] + disp; // p-relative addressing  
 else if (b == 0 && p == 0) TA = disp; // direct addressing  
 }  
 if (x == 1) TA += REG[regX];  
  
 // execute instructions  
 switch (opcode) {  
 case 0XB4: // CLEAR  
 REG[reg1] = 0;  
 break;  
 case 0x28: // COMP  
 operand = LD\_related\_instruction(ni, TA, format, 3);  
 if (REG[regA] > operand) CC = '>';  
 else if (REG[regA] < operand) CC = '<';  
 else CC = '=';  
 break;  
 case 0xA0: // COMPR  
 if (REG[reg1] > REG[reg2]) CC = '>';  
 else if (REG[reg1] < REG[reg2]) CC = '<';  
 else CC = '=';  
 break;  
 case 0x3C: // J  
 REG[regPC] = J\_related\_instruction(ni, TA, format);  
 break;  
 case 0x30: // JEQ  
 if (CC == '=') REG[regPC] = J\_related\_instruction(ni, TA, format);  
 break;  
 case 0x38: // JLT  
 if (CC == '<') REG[regPC] = J\_related\_instruction(ni, TA, format);  
 break;  
 case 0x48: // JSUB  
 REG[regL] = REG[regPC];  
 REG[regPC] = J\_related\_instruction(ni, TA, format);  
 break;  
 case 0x00: // LDA  
 REG[regA] = LD\_related\_instruction(ni, TA, format, 3);  
 break;  
 case 0x68: // LDB  
 REG[regB] = LD\_related\_instruction(ni, TA, format, 3);  
 break;  
 case 0x50: // LDCH  
 operand = LD\_related\_instruction(ni, TA, format, 1);  
 REG[regA] = (int)((REG[regA] & 0xFFFFFF00) | (0x000000FF & operand)); // rightmost byte  
 break;  
 case 0x70: // LDF  
 REG[regF] = LD\_related\_instruction(ni, TA, format, 6);  
 break;  
 case 0x08: // LDL  
 REG[regL] = LD\_related\_instruction(ni, TA, format, 3);  
 break;  
 case 0x6C: // LDS  
 REG[regS] = LD\_related\_instruction(ni, TA, format, 3);  
 break;  
 case 0x74: // LDT  
 REG[regT] = LD\_related\_instruction(ni, TA, format, 3);  
 break;  
 case 0x04: // LDX  
 REG[regX] = LD\_related\_instruction(ni, TA, format, 3);  
 break;  
 case 0xD8: // RD  
 // input device로부터 아무것도 받지 못했다는 가정. input device로부터 NULL(ascii==0)을 읽어온다  
 REG[regA] = (int)(REG[regA] & 0xFFFFFF00);  
 break;  
 case 0x4C: // RSUB  
 REG[regPC] = REG[regL];  
 break;  
 case 0x0C: // STA  
 ST\_related\_instruction(ni, TA, REG[regA], 3);  
 break;  
 case 0x78: // STB  
 ST\_related\_instruction(ni, TA, REG[regB], 3);  
 break;  
 case 0x54: // STCH  
 ST\_related\_instruction(ni, TA, REG[regA], 1);  
 break;  
 case 0x80: // STF  
 ST\_related\_instruction(ni, TA, REG[regF], 6);  
 break;  
 case 0x14: // STL  
 ST\_related\_instruction(ni, TA, REG[regL], 3);  
 break;  
 case 0x7C: // STS  
 ST\_related\_instruction(ni, TA, REG[regS], 3);  
 break;  
 case 0xE8: // STSW  
 ST\_related\_instruction(ni, TA, REG[regSW], 3);  
 break;  
 case 0x84: // STT  
 ST\_related\_instruction(ni, TA, REG[regT], 3);  
 break;  
 case 0x10: // STX  
 ST\_related\_instruction(ni, TA, REG[regX], 3);  
 break;  
 case 0xE0: // TD  
 CC = '<';  
 break;  
 case 0xB8: // TIXR  
 REG[regX] += 1;  
 if (REG[regX] > REG[reg1]) CC = '>';  
 else if (REG[regX] < REG[reg1]) CC = '<';  
 else CC = '=';  
 break;  
 case 0xDC: // WD  
 break;  
 default:  
 printf("wrong access! : 0x%02X\n", REG[regPC]);  
 return OBJ\_CODE\_ERR;  
 break;  
 }  
 return OK;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : LD\_related\_instruction\*/  
/\*목적 : operand로 (m..m+2)가 오는 즉, operand로 'MEM가 가지고 있는 value'가 오는 경우의 instruction을  
 \* 처리한다.\*/  
/\*리턴값 : tar\_val\*/  
/\*------------------------------------------------------------------------------------\*/  
int LD\_related\_instruction(int ni, int TA, int format, int num\_of\_bytes) {  
 int Val = 0;  
  
 if (format == 4) return TA;  
 switch (ni) {  
 case 0: // ni == 00 : considered as standard SIC instruction  
 // COPY 프로그램에 해당 유형의 instruction은 존재하지 않는다고 가정.  
 break;  
 case 1 : // ni == 01 : immediate addressing  
 Val = TA;  
 break;  
 case 2: // ni == 10 : indirect addressing  
 TA = MEMORY[TA];  
 for (int i = 0; i < num\_of\_bytes; i++) Val += MEMORY[TA + num\_of\_bytes - 1 - i] << (i \* 8);  
 break;  
 case 3: // ni == 11 : simple addressing  
 for (int i = 0; i < num\_of\_bytes; i++) Val += MEMORY[TA + num\_of\_bytes - 1 - i] << (i \* 8);  
 break;  
 default:  
 break;  
 }  
 return Val;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : J\_related\_instruction\*/  
/\*목적 : operand로 m 가 오는 즉, operand로 MEM의 주소가 오는 경우의 mnemonic을 처리한다.\*/  
/\*리턴값 : tar\_addr\*/  
/\*------------------------------------------------------------------------------------\*/  
int J\_related\_instruction(int ni, int TA, int format) {  
 int Addr = 0;  
 if (format == 4) return TA;  
  
 switch (ni) {  
 case 2: // ni == 10 : indirect addressing  
 for (int i = 0; i < 3; i++) Addr += MEMORY[TA + 2 - i] << (i \* 8);  
 break;  
 case 3: // ni == 11 : simple addressing  
 Addr = TA;  
 break;  
 default:  
 break;  
 }  
 return Addr;  
}  
  
/\*------------------------------------------------------------------------------------\*/  
/\*함수 : ST\_related\_instruction\*/  
/\*목적 : operand로 m..m+2가 오며, 해당 메모리 주소에 val\_to\_push를 적재한다. \*/  
/\*리턴값 : 없음 \*/  
/\*------------------------------------------------------------------------------------\*/  
void ST\_related\_instruction(int ni, int TA, int val\_to\_push, int num\_of\_bytes) {  
 if (ni == 2) TA = MEMORY[TA]; // ni == 10 : indirect addressing  
  
 switch (num\_of\_bytes) {  
 case 1:  
 MEMORY[TA] = val\_to\_push & 0x000000FF;  
 break;  
 case 3:  
 // binary로 표현된 tar\_val를 오른쪽에서부터 8bit(=1byte)씩 접근하면서, 메모리에 적재  
 for (int i = 0; i < num\_of\_bytes; i++)  
 MEMORY[TA + 2 - i] = (val\_to\_push & (0x000000FF << (8 \* i))) >> (8 \* i);  
 break;  
 case 6:  
 // binary로 표현된 tar\_val를 오른쪽에서부터 8bit(=1byte)씩 접근하면서, 메모리에 적재  
 for (int i = 0; i < num\_of\_bytes; i++)  
 MEMORY[TA + 5 - i] = (val\_to\_push & (0x0000000000FF << (8 \* i))) >> (8 \* i);  
 break;  
 default:  
 break;  
 }  
}