과목 명: 시스템프로그래밍

담당 교수 명: 박 운 상

<<Assignment 2>>

**서강대학교 컴퓨터학과**

**[학번] 20151571**

**[이름] 유용혁**

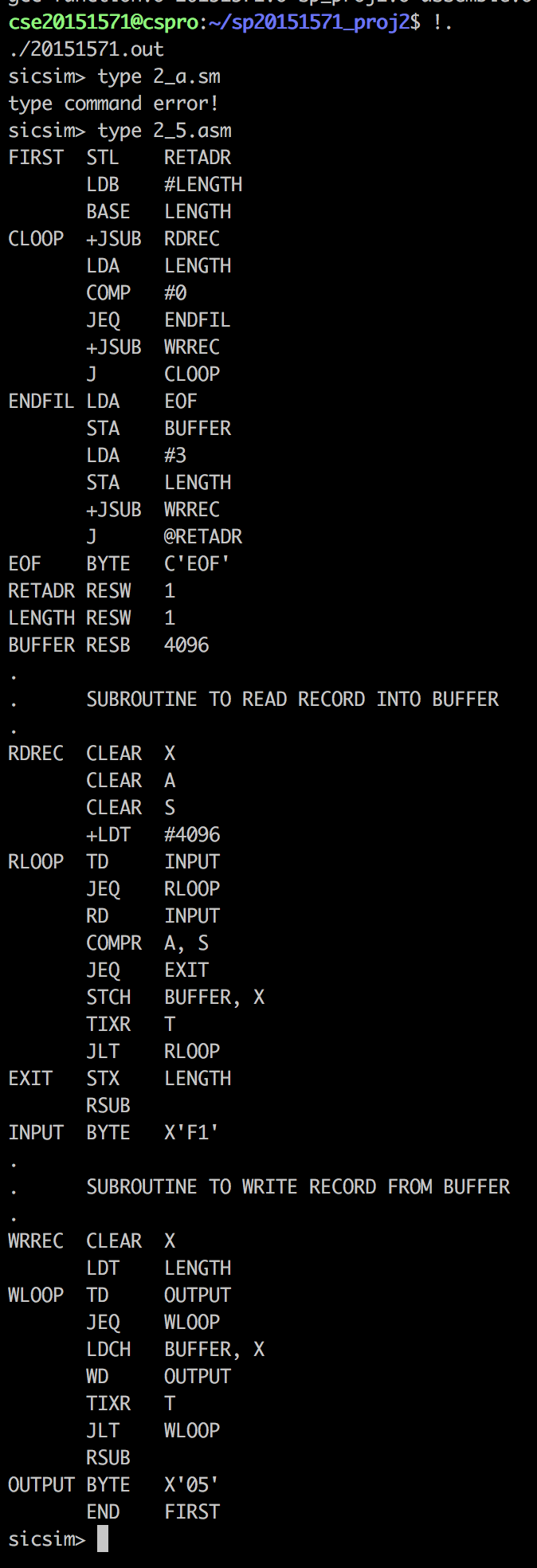
목 차

1. **프로그램 개요** 6 **1.1 assemble 명령어** 6  
    **1.1.1 type 명령어** 6 **1.1.2 assemble 명령어** 6 **1.1.3 symbol 명령어** 7
2. **프로그램 설명** 8 **2.1 프로그램 흐름도** 8  
   **2.2 assemble 프로그램 흐름도** 8
3. **모듈 정의** 9 **3.1 모듈 이름 : symbol\_cmp()** 9 **3.1.1 기능** 9 **3.1.2 사용 변수** 9 **3.2 모듈 이름 : print\_symbol()** 9 **3.2.1 기능** 9 **3.2.2 사용 변수** 9 **3.3 모듈 이름 : delete\_whitespace()** 9 **3.3.1 기능** 9 **3.3.2 사용 변수** 9 **3.4 모듈 이름 : get\_Stablekey()** 9 **3.4.1 기능** 9 **3.4.2 사용 변수** 9 **3.5 모듈 이름 : symbol\_find()** 9 **3.5.1 기능** 9 **3.5.2 사용 변수** 9  
    **3.6 모듈 이름 : symbol\_insert()** 10 **3.6.1 기능** 10 **3.6.2 사용 변수** 10 **3.7 모듈 이름 : opcode\_find()** 10 **3.7.1 기능** 10 **3.7.2 사용 변수** 10 **3.8 모듈 이름 : get\_register()** 10 **3.8.1 기능** 10 **3.8.2 사용 변수** 10 **3.9 모듈 이름 : get\_type()** 10 **3.9.1 기능** 10 **3.9.2 사용 변수** 10 **3.10 모듈 이름 : get\_asmd()** 10 **3.10.1 기능** 10 **3.10.2 사용 변수** 10 **3.11 모듈 이름 : get\_byte()** 10 **3.11.1 기능** 10 **3.11.2 사용 변수** 10 **3.12 모듈 이름 : loc\_count()** 11 **3.12.1 기능** 11 **3.12.2 사용 변수** 11 **3.13 모듈 이름 : get\_objcode()** 11 **3.13.1 기능** 11 **3.13.2 사용 변수** 11 **3.14 모듈 이름 : get\_argu()** 11 **3.14.1 기능** 11 **3.14.2 사용 변수** 11 **3.15 모듈 이름 : sp2\_init()** 11 **3.15.1 기능** 11 **3.15.2 사용 변수** 11 **3.16 모듈 이름 : obj\_byte()** 11 **3.16.1 기능** 11 **3.16.2 사용 변수** 11 **3.17 모듈 이름 : obj\_opcode()** 12 **3.17.1 기능** 12 **3.17.2 사용 변수** 12 **3.18 모듈 이름 : remove\_char()** 12 **3.18.1 기능** 12 **3.18.2 사용 변수** 12 **3.19 모듈 이름 : asmd\_process()** 12 **3.19.1 기능** 12 **3.19.2 사용 변수** 12 **3.20 모듈 이름 : opcode\_location()** 12 **3.20.1 기능** 12 **3.20.2 사용 변수** 12 **3.21 모듈 이름 : make\_line()** 12 **3.21.1 기능** 12 **3.21.2 사용 변수** 13 **3.22 모듈 이름 : make\_objfile()** 13 **3.22.1 기능** 13 **3.22.2 사용 변수** 13 **3.23 모듈 이름 : assembler\_pass2()** 13 **3.23.1 기능** 13 **3.23.2 사용 변수** 13 **3.24 모듈 이름 : print\_file()** 13 **3.24.1 기능** 13 **3.24.2 사용 변수** 14 **3.25 모듈 이름 : command\_assemble()** 14 **3.25.1 기능** 14 **3.25.2 사용 변수** 14 **3.26 모듈 이름 : command\_symbol()** 14 **3.26.1 기능** 14 **3.26.2 사용 변수** 14 **3.27 모듈 이름 : command\_type()** 14 **3.27.1 기능** 15 **3.27.2 사용 변수** 15
4. **구조체 정의** 16 **4.1 구조체 이름 : Pass1** 16 **4.2 구조체 이름 : Symbol** 16 **4.3 구조체 이름 : Symbol\_table** 16 **4.4 구조체 이름 : symbol\_info** 16 **4.5 구조체 이름 : line\_inform** 16 **4.6 구조체 이름 : object\_inform** 17
5. **~~전역 변수 정의~~**   
   이번 프로젝트에서는 전역 변수를 사용하지 않았다.
6. **코드** 18  
    **6.1 assemble.h** 18 **6.2 structure.h** 20 **6.3 assemble.c** 22

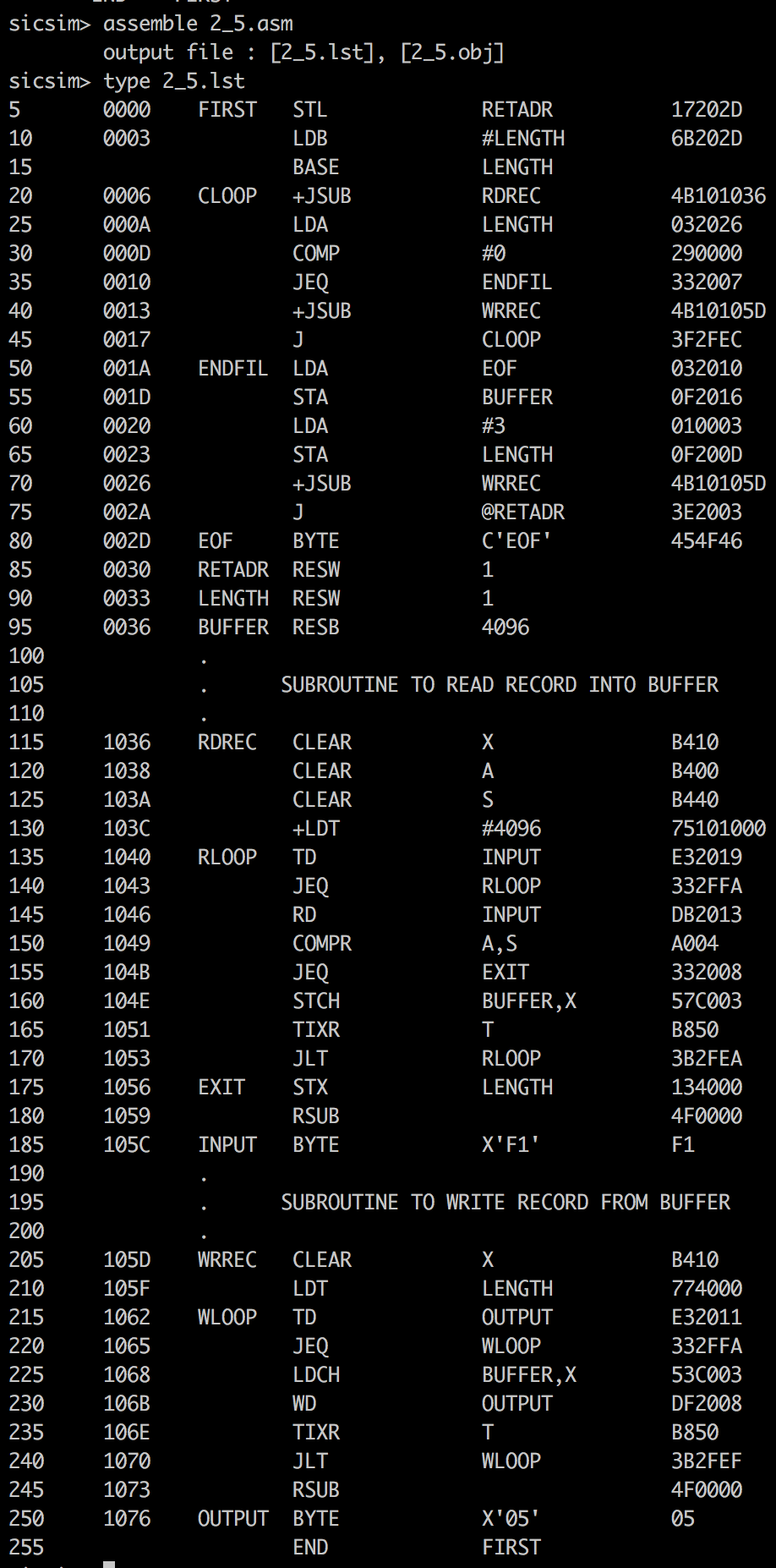
1. **프로그램 개요** 프로젝트 #1에서 구현한 셀(shell)에서 assemble 기능을 추가한 프로그램이다. SIC/XE의 assembly program source 파일을 입력 받아서 object 파일을 생성하고, 어셈블리 과정 중 생성된 symbol table과 결과물인 object 파일을 볼 수 있는 기능을 가지고 있다.

**1.1 assemble 관련 명령어  
1.1.1 type filename**

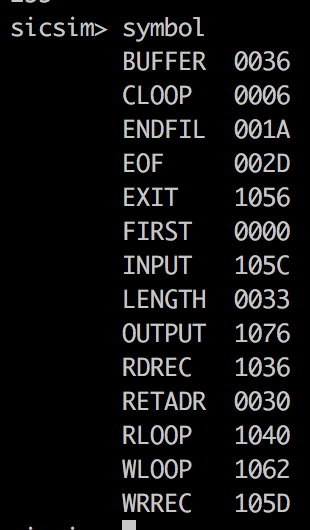
filename에 해당하는 파일을 현재 디렉터리에서 읽어서 화면에 출력한다. 현재 디렉터리에 해당 파일이 존재하지 않으면 에러 메시지를 출력한다. filename이 디렉토리인 경우는 고려하지 않는다



**1.1.2 assemble filename** filename에 해당하는 파일을 asssemble한다. filename이 확장자가 asm가 아닌 경우 에러를 출력한다. filename에 해당하는 파일이 없으면 에러를 출력한다.

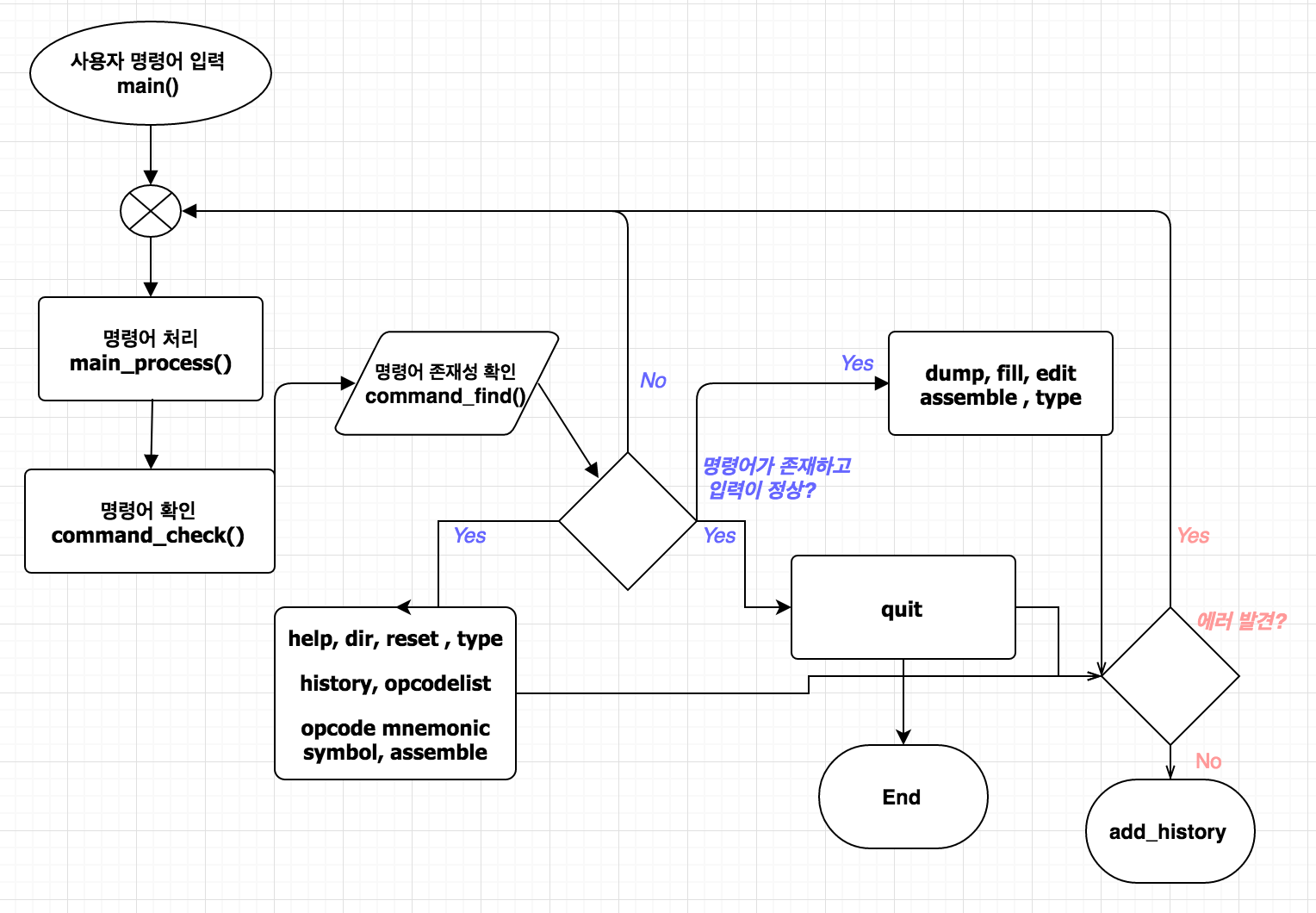


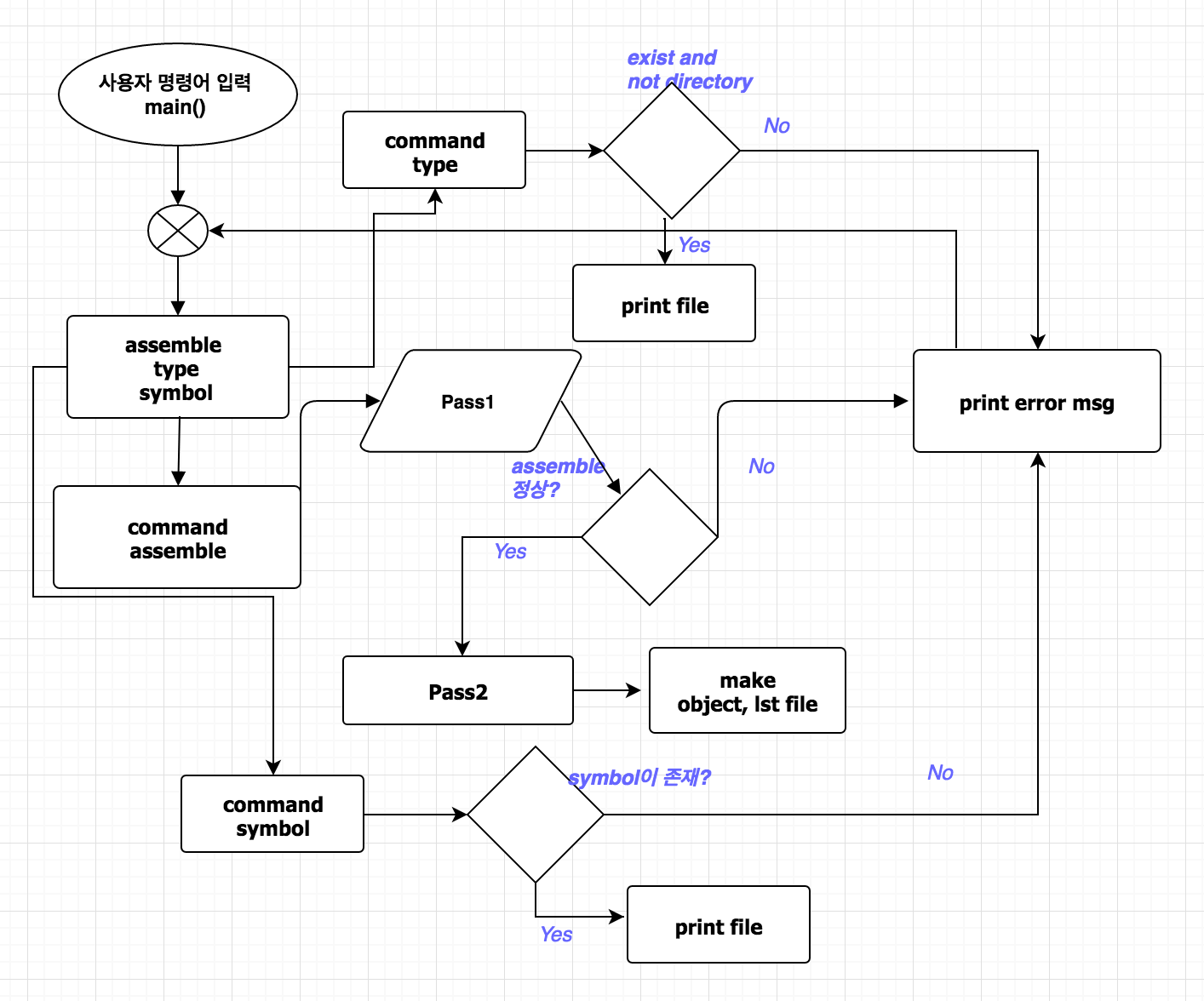
**1.1.3 symbol**

가장 최근에 assemble 성공한 file의 symbol들을 출력한다. 

1. **프로그램 설명**

## 프로그램 전체 흐름도



**2.2 assemble 프로그램 흐름도**

**3. 모듈 정의**

**3.1 모듈 이름 : symbol\_cmp()  
 3.1.1 기능**symbol 끼리 symbol string을 기준으로 크기를 비교해주는 함수이다. **3.1.2 사용 변수**symbolPtr A // 비교 할 구조체  
symbolPtr B // 비교 할 구조체 **3.2 모듈 이름 : print\_symbol()  
 3.2.1 기능**command symbol 기능을 수행하는 함수이다. symbol을 내림차순으로 출력하기 위해서 selection sort를 한후 출력한다. **3.2.2 사용 변수**int idx // min index  
char copy[256] // copy용 문자열   
int address // copy용 정수  
 **3.3 모듈 이름 : delete\_whitespace()  
 3.3.1 기능**‘ ‘, ‘\t’ 과 같은 whitespace를 지워주는 함수이다. 정확히는 ‘ ‘하나로 바꿔준다. **3.3.2 사용 변수**int white\_flag // whitespace flag  
int i, len // index  
int first\_char // 처음 char 저장  
int sep\_flag // ‘\’ 확인하는 flag **3.4 모듈 이름 : get\_Stablekey()  
 3.4.1 기능**symbol table key를 구하는 함수이다. **3.4.2 사용 변수**int key // key 저장하는 변수  
 **3.5 모듈 이름 : symbol\_find()  
 3.5.1 기능**해당하는 문자열의 symbol이 table에 있는지 확인하고 있으면 그 table의 pointer를 반환한다. 없으면 NULL을 반환한다. **3.5.2 사용 변수**  
int key // string의 key를 확인  
 **3.6 모듈 이름 : symbol\_insert()  
 3.6.1 기능**symbol table에 해당하는 symbol을 넣어주는 함수이다. **3.6.2 사용 변수**int key // key 저장 변수  
 **3.7 모듈 이름 : opcode\_find()  
 3.7.1 기능**opcode가 있는지 확인하는 함수이다. 만약에 존재하면 해당하는 pointer의 주소를 없으면 NULL을 반환한다. **3.7.2 사용 변수**int i // idx  
Hnode ptr // temp pointer 변수 **3.8 모듈 이름 : get\_register()  
 3.8.1 기능**해당하는 문자열이 어떤 register인지 확인해주는 함수이다. **3.8.2 사용 변수**

없다. **3.9 모듈 이름 : get\_type()  
 3.9.1 기능**해당하는 라인이 주석인지 opcode인지 assemble directives인지 label인지 구분해주는 함수이다. **3.9.2 사용 변수**없다. **3.10 모듈 이름 : get\_asmd()  
 3.10.1 기능**해당하는 문자열이 assemble directives인지 찾고 어떤 assemble directives인지 찾아주는 함수이다. **3.10.2 사용 변수**없다. **3.11 모듈 이름 : get\_byte()  
 3.11.1 기능**해당하는 문자열이 byte인 경우 제대로 operand가 입력되었는지 확인해주는 함수이다. **3.11.2 사용 변수**char \*string // ‘C’나 ‘X’를 제거한 문자열  
char \*error // error 체크 pointer  
int len // string length **3.12 모듈 이름 : loc\_count()  
 3.12.1 기능**location counter 함수이다. 더해야할 location을 구해서 counter를 반환한다. **3.12.2 사용 변수**int len // string length  
char \*token // tokenize pointer  
char \*error // error check pointer **3.13 모듈 이름 : get\_objcode()  
 3.13.1 기능**object 코드를 구해주는 함수이다. e = 0인 경우 3byte로 e = 1인 경우 4byte로 계산해준다. **3.13.2 사용 변수**int code // binary to decimal code  
int bit[34] // bit array  
int len, idx // string length, index **3.14 모듈 이름 : get\_argu()   
 3.14.1 기능**해당하는 line을 tokenize하여서 각 문자열의 위치를 argu 배열에 저장한다. 그리고 첫 번째 문자열에 대해서 주석인지 opcode인지 assemble directives인지 label인지 찾는다. **3.14.2 사용 변수**int len, num // string number, assemble directive number  
char sep[] // seperate array  
char \*token // seperate pointer  
int type[5] // 해당하는 문자열이 어떤 type인지 저장 **3.15 모듈 이름 : sp2\_init()  
 3.15.1 기능**project #2에서 사용할 변수들을 초기화 해준다. **3.15.2 사용 변수**없다.  
 **3.16 모듈 이름 : obj\_byte()   
 3.16.1 기능**byte인 경우 object 코드를 생성해주는 함수이다. error가 발견되면 return 1, 없으면 return 0을 반환한다. **3.16.2 사용 변수**char c // byte이므로 ‘C’나 ‘X’가 존재하는지 확인해주기 위해 c에 첫글자 저장  
char copy[256] // copy용 string  
char tmp[5] // objcode string 저장

char \*error // error check pointer  
int len, s // string length  
 **3.17 모듈 이름 : obj\_opcode()  
 3.17.1 기능**opcode인 경우 object code를 구해주는 함수이다. error가 발견되면 return 1, 없으면 return 0을 반환한다. **3.17.2 사용 변수**Hnode hashptr // hashpointer  
symbolPtr sptr // symbolpointer  
char tmp1[], tmp2[], tmp3[] // copy용 문자열  
char \*error // error check pointer  
int regi\_num // register number  
int symbol\_num // symbol number  
int immediate // immediate value  
int format // format number **3.18 모듈 이름 : remove\_char()  
 3.18.1 기능**immediate이나 indirect 경우 붙는 ‘#’ , ‘$’를 제거해주고 string에 오류가 있는지 확인하는 함수이다. 에러가 발견되면 return -1 , 발견이 안되면 return 0을 한다. **3.18.2 사용 변수**char \*error // error pointer  
 **3.19 모듈 이름 : asmd\_process()   
 3.19.1 기능**assemble directives인 경우 line location을 처리해주는 함수이다. 에러가 발견되면 return 1 아니면 return location counter가 반환된다. **3.19.2 사용 변수**int asmd // assemble directive **3.20 모듈 이름 : opcode\_location()   
 3.20.1 기능**opcode인 경우 location counter함수이다. 해당하는 opcode 형식에 따라서 format을 저장하고 그 만큼 location을 더한다. **3.20.2 사용 변수**int format // format **3.21 모듈 이름 : make\_line()   
 3.21.1 기능**입력 받은 assemble file 라인을 처리해주는 함수이다. 해당하는 line이 어떤 type인지 구해주고 type에 따라 처리를 해준다. 어떤 format인지, 어떤 opcode인지 symbol을 가졋는지 등등 pass2 algorithm에 필요한 정보들을 저장한다. **3.21.2 사용 변수**char \*error // error pointer  
int label\_type // label 있는 경우 type을 한번 더 구함. **3.22 모듈 이름 : make\_objfile()  
 3.22.1 기능**object file을 생성하는 함수이다. pass2에서 저장한 object information을 이용하여 line by line으로 작성한다, **3.22.2 사용 변수**int i, idx // i = index , idx = 주석을 제외한 라인부터 시작 int end\_flag // end flag **3.23 모듈 이름 : assembler\_pass2()  
 3.23.1 기능**pass2 algorithm을 구현한 함수이다. pass2 algorithm이 성공적으로 종료되면 lst와 obj file을 생성하고 종료한다. **3.23.2 사용 변수**FILE \*fp, \*lstfp, \*objfp // filepointer  
char lstname[30] // lst file name  
char objname[30] // object file name  
char buffer[256] // string buffer  
char bef[256] // string copy buffer   
char copy[256] // 파일명만 저장  
char \*errorp // error pointer  
int arr[7] // n , i , x, b, p , e  
int linenum, type, num // line number, type int  
size\_t idx, len // index  
int end\_flag, error , obj\_flag, obj\_idx // flag  
int n, i , x, b, p , e   
int asmd, start\_line  
Hash \*hashTable  
symbolPtr sptr  
object\_inform obj\_info[300] // object information **3.24 모듈 이름 : print\_file()   
 3.24.1 기능**command type 명령에서 file 내용을 출력하는 함수이다. **3.24.2 사용 변수**FILE \*fp // file pointer  
char buffer[500]// string **3.25 모듈 이름 : command\_assemble()   
 3.25.1 기능**assemble command를 수행하는 함수이다.해당하는 파일 확장자가 .asm이 아니거나 해당하는 파일이 존재하지 않으면 에러처리를 해준다. 그런 문제가 없는 경우 pass1 algoirthm을 시작한다.처음에는 pass1 algorithm을 수행하기 위해서 line by line으로 입력을 받는다. 그리고 각 line 마다 해당하는 line의 정보를 저장하여서 label 유무, opcode인지 asseble directives인지 구분하여 준다. 그리고 각 symbol의 주소와 line의 location을 각각 저장하고 program length를 저장한다.  
 도중에 에러가 발견되지 않으면 pass2 algorithm으로 넘어간다. pass2는 lst파일과 obj 파일을 만드는 과정이다. pass1에서 저장한 정보를 통해서 lst 파일을 작성한다.  
pass1 algorithm -> make\_line  
pass2 algorithm -> ( assembler\_pass2, make\_objfile )  
 **3.25.2 사용 변수**FILE \*fp // filepointer  
char buffer[256] // string  
char copy[256] // copy string  
char \*argu[10] // argument 저장  
char \*filename // filename 저장  
char \*ptr // filename pointer 주소 저장  
int type, len , flag // type   
size\_t idx // index  
line\_inform line\_info[200] // line information  
Pass1 Pinfo // pass1 information  
 **3.26 모듈 이름 : command\_symbol()   
 3.26.1 기능**symbol 명령어를 수행하는 함수이다. 가장 최근에 assemble 성공한 파일의 symbol들을 출력한다.  **3.26.2 사용 변수**int size // symbolTable size

**3.27 모듈 이름 : command\_type()   
 3.27.1 기능**  
type 명령어를 수행하는 함수이다. 현재 디렉토리에 있는 filename에 해당하는 파일을 읽어서 화면에 출력한다. 디렉토리이거나 존재하지 않으면 에러 메세지를 출력한다. **3.27.2 사용 변수**DIR \*dirp; // DIR 변수  
struct dirent \*direntp // dirent 구조체 변수  
struct stat buf // 어떤 종류인지 구분해주는 변수  
int flag // flag  
char \*filename // filename

**4. 구조체 정의   
  
 4.1 구조체 이름 : Pass1**typedef struct{

int base\_address; // base address를 저장

int location; // location 저장

int line; // line

int address; //

int program\_len; // program length 저장

char \*\*argu; // argu

}Pass1;   
 **4.2 구조체 이름 : Symbol**typedef struct \_\_SYMBOL\_\_{

char symbol[50]; // symbo string

int address; // symbol address

symbolPtr next; // pointer

}Symbol; **4.3 구조체 이름 : Symbol\_table**typedef struct{

const int size; // 37

Symbol \*table[37]; // table pointer

Hash \*hashTable; // opcode hashtable

}Symbol\_table;

**4.4 구조체 이름 : symbol\_info**typedef struct{

char symbol[5]; // symbol string

int address; // symbol address

}symbol\_info;  
 **4.5 구조체 이름 : line\_inform**typedef struct{

int location;

int format; // format -1 : 주석

int label\_flag; // label flag

int base\_flag; // base flag

int n\_flag; //

int modi\_flag; // modification flag

int obj\_code; // object code flag

char asmd[50]; // assembly directive string

char opcode[50]; // opcode string

char symbol[50]; // symbol string

char operhand[50]; // operhand string

char obj\_strcode[50]; // object code string

char comment[100]; // comment string

}line\_inform; **4.6 구조체 이름 : object\_inform**

typedef struct{

int start; // start address

int size; // line size

}object\_inform;

**5. 코드**

**5.1 assemble.h**

#ifndef \_\_sp\_proj2\_\_

#define \_\_sp\_proj2\_\_

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include <sys/types.h>

#include "structure.h"

// assembler directives(pseudo-instructions)

typedef enum{

start, end, base, byte,

word, resb, resw

}assem\_directives;

//register

typedef enum{

A, X, L, B , S ,

T , F , PC, SW

}Register;

typedef enum{

comment, opcode, pass1\_start, asmd, label, pass1\_end

}pass1\_info;

int symbol\_cmp(void \*, void \*); // symbol compare function

void print\_symbol(symbol\_info \*, int ); // symbol print function

void delete\_whitespace(char \*); // whitespace 제거해주는 함수

int get\_Stablekey(Symbol\_table \*, char \*); // symbol\_table에 넣을 string의 key를 구해주는 함수

symbolPtr symbol\_find(Symbol\_table \*, char \*string); // symbol find funcition

void symbol\_insert(Symbol\_table \*, char \*, int); // symbol\_table insert fucntin

void symbol\_init(Symbol\_table \*); // symbol init

Hnode opcode\_find(Hash \*, char \*mnemonic); // opcode 찾는 함

int get\_register(char \*); // 어떤 register인지 구해주는 함수

int get\_type(char \*, Hash \*); // string이 label인지 address인지 등등 구해주는 함수

int get\_asmd(char \*); // 어떤 asmd인지 구해주는 함수.

int get\_byte(char \*\*, line\_inform \*line\_info); // byte인 경우 값을 구해주는 함

int loc\_count(char \*, int asmd, int location ); // location counter

int get\_objcode(int opcode, int n, int i, int x,

int b, int p, int e, int address); // object code 생성함수

int get\_operhand(char \*); // operhand를 구해주는 함수

int get\_argu(char \*, char \*\*, Hash \*); // 인자 개수를 확인해주는 함수

int obj\_byte(FILE \*, Symbol\_table\*, line\_inform \*line\_info,

object\_inform \*,int \*, int \*); // byte 인경우 object code 생성

int obj\_opcode(FILE \*, Hash \*, Symbol\_table \*, line\_inform \*,

object\_inform \*,int \*, int \*, int \*, int base\_address); // opcode인 경우 object code 생성하는 함수

int remove\_char(char \*, Symbol\_table \*, int \*, int \*); // # 이나 @ 없애주는 함수

int asmd\_process(char \*\*, line\_inform \*line\_info, int location); // assemble directives 처리해주는 함수

int opcode\_location(Pass1 \*, Hash \*, char \*\*, line\_inform \*line\_info); // locationo 계산해주는 함수

int make\_line(char \*string, int type, size\_t idx, int \*flag,

Symbol\_table \*, Pass1 \*Pinfo, line\_inform \*line\_info); // pass1에서 각 라인 마다 처리해주는 함수

void make\_objfile(FILE \*fp, int, int ,line\_inform \*, object\_inform \*); // obj file 만드는 함수

int assembler\_pass2(char \*filename, Symbol\_table \*, int length,

Pass1 \*Pinfo, line\_inform \*line\_info); // pass2 algorithm 함수

void print\_file(char \*); // file 내용을 출력하는 함수

#endif

**5.2 structures.h**

**#ifndef \_\_structure\_\_**

**#define \_\_structure\_\_**

**typedef struct LINKED\_LIST \* History;**

**typedef struct LINKED\_LIST \* Lnode;**

**typedef struct HASH\_Linked\_List \* Hnode;**

**typedef struct LINKED\_LIST{**

**char command[50];**

**Lnode next;**

**}Linked\_list;**

**#ifndef HASH\_SIZE**

**#define HASH\_SIZE 20**

**typedef struct \_HASH\_{**

**int size;**

**Hnode Table[HASH\_SIZE];**

**}Hash;**

**#endif**

**typedef struct HASH\_Linked\_List{**

**int n\_opcode;**

**char code[5];**

**char str\_opcode[50];**

**Hnode next;**

**}Hash\_Node;**

**typedef struct MEMORY{**

**unsigned char memory[1048576];**

**int last\_address;**

**int max\_address;**

**}Shell\_Memory;**

**typedef struct{**

**int base\_address;**

**int location;**

**int line;**

**int address;**

**int program\_len;**

**char \*\*argu;**

**}Pass1;**

**typedef struct \_\_SYMBOL\_\_ \*symbolPtr;**

**typedef struct \_\_SYMBOL\_\_{**

**char symbol[50];**

**int address;**

**symbolPtr next;**

**}Symbol;**

**//symbol linked\_list**

**typedef struct{**

**const int size; // 37**

**Symbol \*table[37];**

**Hash \*hashTable;**

**}Symbol\_table;**

**//symbol hashtable**

**typedef struct{**

**char symbol[5];**

**int address;**

**}symbol\_info;**

**//symbol information**

**typedef struct{**

**int location;**

**int format; // format -1 : 주석**

**int label\_flag;**

**int base\_flag;**

**int n\_flag;**

**int modi\_flag;**

**int obj\_code;**

**char asmd[50];**

**char opcode[50];**

**char symbol[50];**

**char operhand[50];**

**char obj\_strcode[50];**

**char comment[100];**

**}line\_inform;**

**//linf information**

**typedef struct{**

**int start;**

**int size;**

**}object\_inform;**

**//object code information**

**#endif**

**5.3 assemble.c**

#include "assemble.h"

/\*

\* 두개의 symbol string을 비교해주는 함수

\* a < b return -1

\* a == b return 0

\* a > b return 1

\*/

int symbol\_cmp(void \*a, void \*b){

symbolPtr A = (symbolPtr)a;

symbolPtr B = (symbolPtr)b;

return strcmp(A->symbol, B->symbol);

}

/\*

\* symbol들을 출력하는 함수

\* 정렬한 뒤 출력한다.

\*/

void print\_symbol(symbol\_info \*symbolTable, int size){

int idx;

char copy[256];

int address;

//selection sort

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

idx = i;

for( int j = i; j < size; ++j){

if ( symbol\_cmp(symbolTable[j].symbol, symbolTable[idx].symbol) > 0 )

idx = j;

}

//swap

address = symbolTable[idx].address;

symbolTable[idx].address = symbolTable[i].address;

symbolTable[i].address = address;

strcpy(copy ,symbolTable[idx].symbol);

strcpy(symbolTable[idx].symbol, symbolTable[i].symbol);

strcpy(symbolTable[i].symbol, copy);

}

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

printf("\t%s\t%04X\n", symbolTable[i].symbol,

symbolTable[i].address);

}

}

/\*

\* whitespace('\t', ' ') 같은 것들을 삭제해주는 함수

\*/

void delete\_whitespace(char\* str){

int whitend\_flag = 0;

int i, len;

int idx;

int first\_char = 0;

int sep\_flag = 0;

len = strlen(str);

for(i = 0; i <= len; i++){

if(i == len){

//EOF 넣음

if(whitend\_flag >= 1)

str[i-whitend\_flag] = '\0';

}

//whitespace 인경우 flag 증가

if(str[i] == '\''){

sep\_flag++;

whitend\_flag = 0;

if(sep\_flag == 2)

sep\_flag = 0;

}

if(sep\_flag)

continue;

if(str[i] == ' ' || str[i] == '\t'){

whitend\_flag++;

if(str[i] == '\t')

str[i] = ' ';

}

else{ // whitespace 아닌 경우

if(whitend\_flag >= 1){ // whitespace가 여려 개인 경우

if(!first\_char)

strncpy(str+idx, str+i, len-i+1);

else if(str[i] == ',')

strncpy(str+idx, str+i, len-i+1), i--;

// whitespace 만큼 당김

else if(whitend\_flag > 1)

strncpy(str+idx+1, str+i, len-i+1);

str[idx+len-i+1] = '\0';

len = strlen(str);

i = idx;

}

// ','인 경우

if(str[i] == ','){

// Add a space behind of ','

if(i+1 < 256 && (str[i+1] != ' ' && str[i+1] != '\t')){

char\* tmp = (char\*)malloc((len-i)\*sizeof(char));

strcpy(tmp, str+i+1);

str[i+1] = ' ';

str[i+2] = '\0';

strcat(str,tmp);

len = strlen(str);

i--;

}

}

// flag 초기화

whitend\_flag = 0;

first\_char = 1;

}

// idx 초기화

if(whitend\_flag == 1)

idx = i;

}

}

//symbol 초기화 함수

void symbol\_init(Symbol\_table \*Stable){

symbolPtr ptr;

symbolPtr nptr;

for ( int i = 0; i < 37; ++i){

for ( ptr = Stable->table[i]; ptr != NULL;){

nptr = ptr->next;

free ( ptr );

ptr = nptr;

}

Stable->table[i] = NULL;

}

}

//opcode find function

//존재하면 opcode pointer를 return

//없으면 return NULL

Hnode opcode\_find(Hash \*hashTable, char \*mnemonic){

int i;

Hnode ptr;

for ( i = 0; i < hashTable->size; ++i ){ // hash\_table에서 일일이 찾는다

for ( ptr = hashTable->Table[i]; ptr != NULL; ptr = ptr -> next ){

if ( strcmp( ptr->str\_opcode , mnemonic ) == 0)

return ptr;

}

}

return NULL; // 존재하지 않는 경우

}

//symbol table key 값을 구하는 함수

int get\_Stablekey(Symbol\_table \*Stable, char \*string){

int key = 0;

for ( size\_t i = 0; i < strlen(string); ++i )

key += string[i]; // get key

return key % Stable->size;

}

// 해당하는 문자열과 일치하는 symbol 이 있는지 구해주는 함수

// return 값은 symbol pointer

// 없으면 return NULL

symbolPtr symbol\_find(Symbol\_table \*Stable, char \*string){

int key = get\_Stablekey(Stable, string);

symbolPtr ptr;

for ( ptr = Stable->table[key]; ptr != NULL; ptr = ptr -> next)

if ( strcmp( ptr -> symbol, string) == 0)

return ptr;

return NULL;

}

//symbol insert function

void symbol\_insert(Symbol\_table \*Stable, char \*string, int address){

int key = get\_Stablekey(Stable, string);

symbolPtr ptr = NULL;

ptr = ( symbolPtr ) malloc ( sizeof (Symbol) );

strncpy ( ptr->symbol, string, sizeof(ptr->symbol));

ptr->address = address;

ptr->next = Stable->table[key];

Stable->table[key] = ptr;

}

//register find function

int get\_register(char \*string){

for ( size\_t i=0; i < strlen(string); ++i )

string[i] = ( string[i] >= 'a' && string[i] <= 'z') ?

string[i] -'a' + 'A' : string[i];

if ( strcmp(string,"A") == 0 )

return A;

else if ( strcmp(string,"X") == 0 )

return X;

else if ( strcmp(string,"L") == 0 )

return L;

else if ( strcmp(string,"B") == 0 )

return B;

else if ( strcmp(string,"S") == 0 )

return S;

else if ( strcmp(string,"T") == 0 )

return T;

else if ( strcmp(string,"F") == 0 )

return F;

else if ( strcmp(string,"PC") == 0 )

return PC;

else if ( strcmp(string,"SW") == 0 )

return SW;

return -1; // error

}

//assemble directives find function

int get\_asmd(char \*string) {

for ( size\_t i=0; i < strlen(string); ++i )

string[i] = ( string[i] >= 'a' && string[i] <= 'z') ?

string[i] -'a' + 'A' : string[i]; // 다 대문자로 만듬

if ( strcmp(string,"START") == 0 )

return start;

else if(strcmp(string,"END") == 0)

return end;

else if(strcmp(string,"BASE") == 0)

return base;

else if(strcmp(string,"BYTE") == 0)

return byte;

else if(strcmp(string,"WORD") == 0)

return word;

else if(strcmp(string,"RESB") == 0)

return resb;

else if(strcmp(string,"RESW") == 0)

return resw;

return -1; // error

}

/\*

\* operand가 byte인 경우 line에 operand에 byte 값을 넣고에러 있는지 확인해주는 함수

\*/

int get\_byte(char \*\*argu, line\_inform \*line\_info){

char \*string = &argu[2][1];

char \*error = NULL;

int len = 0;

if ( argu[3] != NULL ) // byte 값에서 추가로 하나 더 적혀 있는 경우 에

return -1;

if ( (argu[2][0] == 'C' || argu[2][0] == 'c') // C인지 X 인지 확인

|| (argu[2][0] == 'X' || argu[2][0] == 'x') ){

argu[2][0] = ( 'A' <= argu[2][0] && 'Z' >= argu[2][0] ) ?

argu[2][0] : (argu[2][0] - 'a' + 'A');

}

if ( argu[2][0] == 'C' ){ // C 인 경우

if ( string[0] != 39 || string[len] != 39)

return -1;

len = strlen(string) - 2;

}

else if ( argu[2][0] == 'X' ) { // X인 경우

strtol(string + 1, &error, 16);

if ( string[0] != 39 || \*error != 39)

return -1;

len = error - string;

}

strcpy(line\_info->operhand, argu[2]);

return len;

}

// location counter function

// return counted location

int loc\_count(char \*string, int asmd, int location){

int len = 0;

char \*token = NULL;

char \*error = NULL;

token = strstr(string, "'");

if ( asmd == byte ){ // byte의 경우 string length로 판단

len = strstr(token + 1, "'") - token - 1;

if ( strstr ( string, "C") != NULL || strstr ( string, "c") != NULL )

location = len;

else if ( strstr ( string, "X") != NULL || strstr ( string, "x") != NULL )

location = len / 2;

else

return -1;

}

else{ // 아닌 경우 word, resb, resw로 판단

string = strtok(string, " \t\n\r");

if ( asmd == word )

location = 3;

else if( asmd == resb )

location = (int) strtol(string, &error, 10);

else if (asmd == resw )

location = (int) strtol(string, &error, 10) \* 3;

if( \*error != '\0')

return -1;

}

return location;

}

//해당하는 문자열이 주석인지 opcode인지 assemble directives인지 구해주는 함수

int get\_type(char \*string, Hash \*hashTable){

if ( strcmp (string , "." ) == 0 )

return comment;

else if( ( string[0] == '+' && opcode\_find ( hashTable, string + 1) != NULL )

|| opcode\_find ( hashTable, string ) != NULL )

return opcode; // opcode

else if ( get\_asmd(string) != -1 )

return asmd; // assembly directives

return label; // label

}

/\*

\* tokenize를 하여서해당하는라인이 comment인지 opcode인지 label이 없는지 확인해주는 함수

\* error가 발견되면 return 1

\* 정상적이라면 return 0

\*/

int get\_argu(char \*buffer, char \*argu[], Hash \*hashTable){

int len = 0, num;

char sep[] = " \t\r\n";

char \*token = NULL;

int type[5];

//tokenize 과정

delete\_whitespace(buffer);

if ( (token = strstr(buffer, ",")) != NULL){

if ( \*(token -1 ) == ' '){

strcpy(token-1, token);

token--;

}

if( \*(token + 1) == ' ')

strcpy(token+1, token + 2);

}

token = strtok(buffer, sep);

while( token != NULL){

argu[len] = token;

type[len++] = get\_type(token, hashTable);

token = strtok(NULL, sep);

}

argu[len] = NULL;

if ( type[0] == comment) // comment

return comment;

else if ( type[0] == opcode) // opcode

return opcode;

else if( type[0] == asmd){ // assembly directives

if ( strcmp(argu[0], "END") == 0 )

return pass1\_end;

return asmd;

}

else if ( type[0] == label){ // label

num = get\_asmd(argu[1]);

if( num == start )

return pass1\_start; // label이 있는 start

return label;

}

return -1;

}

/\*

\* object code를 구하는 함수

\* e = 0인 경우 3 byte로 계산

\* e = 1 인 경우 4 byte로 계산

\*/

int get\_objcode(int opcode, int n, int i, int x,

int b, int p, int e, int address) {

int code = 0;

int bit[34] = { 0, };

int len= 0 , idx = 0;

opcode /= 4;

if ( e == 1) // format 4

len = 31;

else

len = 23;

for ( idx = 5; idx >= 0; --idx){ // get binary code

bit[idx] = opcode % 2;

opcode /= 2;

}

bit[6] = n; bit[7] = i; bit[8] = x;

bit[9] = b; bit[10] = p; bit[11] = e;

for( idx = len; address >= 2; --idx){ // get binary code

bit[idx] = address % 2;

address /= 2;

}

bit[idx] = address;

for ( idx = 0; idx <= len; ++idx){ // get decimal code

code \*= 2;

code += bit[idx];

}

return code;

}

/\*

\* 해당하는라인이 opcode 일 때 location counter function

\* error가 발견 되면 return -1

\* 정상적이라면 return 1을 한다

\*/

int opcode\_location(Pass1 \*Pinfo, Hash \*hashTable, char \*\*argu, line\_inform \*line\_info){

Hnode ptr;

int format = 0;

if(argu[0][0] == '+')

format = 4;

else{

ptr = opcode\_find ( hashTable, argu[0]);

if ( strcmp(ptr->code, "1" ) == 0)

format = 1; // format 1

else if ( strcmp(ptr->code, "2") == 0)

format = 2; // format 2

else if ( strcmp(ptr->code, "3/4") == 0)

format = 3; // format 3

else

return -1;

}

line\_info->location = Pinfo->location;

line\_info->format = format;

Pinfo->location += format;

return 1;

}

/\*

\* immediate이나 indirect의 경우 붙는 '#' '@'를 제거해주고 오류가 있는지 확인하는 함수

\* 오류가 발견되면 return -1

\* else return 0

\*/

int remove\_char(char \*string, Symbol\_table \*symbolTable, int \*n, int \*i){

char \*error = NULL;

if(string[0] == '#' || string[0] == '@') { // immediate or indirect addressing mode

string[0] == '#' ? (\*i = 1) : (\*n = 1);

strcpy(string,string + 1);

strtol(string, &error, 10);

if ( \*error != '\0' && symbol\_find(symbolTable, string) == NULL)//symbol이 아니면 10진수

return -1;

}

else //#도 @도 아니면 n은 1, i는 1(simple mode)

\*i = 1; \*n = 1;

if(strchr(string,'#') != NULL && strchr(string,'@') != NULL) //두 개다 있으면 에러

return -1;

return 0;

}

/\*

\* assemble directive인 경우 line location을 처리 해주는 함수

\* assemble directive 처리과정에서 에러 난 경우 return -1

\* else return location

\*/

int asmd\_process(char \*\*argu, line\_inform \*line\_info, int location){

//int byte\_val;

int asmd = get\_asmd(argu[1]);

strcpy(line\_info->operhand, argu[2]);

if ( asmd == byte )

return loc\_count(line\_info->operhand, byte, location);

else if ( asmd == word )

return loc\_count(line\_info->operhand, word, location);

else if ( asmd == resb )

return loc\_count(line\_info->operhand, resb, location);

else if ( asmd == resw )

return loc\_count(line\_info->operhand, resw, location);

return -1;

}

/\*

\* obj에서 쓸 line을 생성하는 함수이다.

\* label이 있는 경우 argu[0] = label argu[1] = opcode or assembly directives

\*/

int make\_line(char \*string, int type, size\_t idx, int \*flag,

Symbol\_table \*Stable, Pass1 \*Pinfo, line\_inform \*line\_info) {

char \*error;

int label\_type;

if ( type == comment ){ // comment 인 경우

line\_info[idx].format = -1;

strncpy( line\_info[idx].comment, string,

sizeof(line\_info[idx].comment));

return 0;

}

else if ( type == opcode ){ // opcode인 경우

if ( opcode\_location(Pinfo, Stable->hashTable,

Pinfo->argu, &line\_info[idx]) == -1 ) // opcode location counter function

return -1; // error

strcpy( line\_info[idx].opcode, Pinfo->argu[0]);

if(Pinfo->argu[1] != NULL)

strcpy( line\_info[idx].operhand, Pinfo->argu[1]);

}

else if ( type == asmd){ // assembly directives

strcpy(line\_info[idx].asmd, Pinfo->argu[0]);

strcpy(line\_info[idx].operhand, Pinfo->argu[1]);

line\_info[idx].location = Pinfo->location;

if ( strcmp( line\_info[idx].asmd, "BASE" ) == 0 ){ // base인경우

line\_info[idx].base\_flag = 1; // base-flag

strcpy(line\_info[idx].operhand, Pinfo->argu[1]);

}

else

Pinfo->location += asmd\_process( Pinfo->argu , &line\_info[idx], Pinfo->location);

// base가 아닌 경우 location 구해줌

}

else if ( type == pass1\_start ){ // start인 경우

if ( (\*flag) ) // start가 처음이 아닌 경우

return -1;

Pinfo->location = (int)strtol(Pinfo->argu[2], &error, 16);

if( \*error != '\0' ) // 16진수에 에러 있는 경우

return -1;

Pinfo->program\_len = Pinfo->location;

line\_info[idx].location = Pinfo->location;

strcpy(line\_info[idx].asmd, "START");

strncpy(line\_info[idx].symbol, Pinfo->argu[0], sizeof(line\_info[idx].symbol));// start symbol

strcpy(line\_info[idx].asmd, Pinfo->argu[1]);

strcpy(line\_info[idx].operhand, Pinfo->argu[2]); // file start line

line\_info[idx].label\_flag = 1; // label이 있으므로 1

if ( !symbol\_find(Stable, line\_info[idx].symbol) ){

symbol\_insert(Stable, line\_info[idx].symbol,

Pinfo->location);

}

else

return -1;

}

else if ( type == pass1\_end ){ // end인 경우

strcpy( line\_info[idx].asmd, Pinfo->argu[0] );

strcpy( line\_info[idx].operhand, Pinfo->argu[1]);

line\_info[idx].location = Pinfo->location;

Pinfo->program\_len = line\_info[idx-1].location - Pinfo->program\_len + 1; //program length check

return 0;

}

else if ( type == label ) { // label이 잇는 경우

strcpy( line\_info[idx].symbol , Pinfo->argu[0]);

if ( !symbol\_find(Stable, line\_info[idx].symbol) ){

symbol\_insert(Stable, line\_info[idx].symbol,

Pinfo->location); // symbol 삽입

}

else // 이미 있는 경우 에러 처리

return -1;

line\_info[idx].label\_flag = 1;

label\_type = get\_type(Pinfo->argu[1], Stable->hashTable);

if ( label\_type == opcode ) { // opcode인 경우

if ( opcode\_location(Pinfo, Stable->hashTable,

&(Pinfo->argu[1]), &line\_info[idx]) == -1 )

return -1;

strcpy( line\_info[idx].opcode, Pinfo->argu[1]);

strcpy( line\_info[idx].operhand, Pinfo->argu[2]);

}

else if ( label\_type == asmd){ // assemble directives

strcpy(line\_info[idx].asmd, Pinfo->argu[1]);

strcpy(line\_info[idx].operhand, Pinfo->argu[2]);

line\_info[idx].location = Pinfo->location;

if ( strcmp( line\_info[idx].asmd, "BASE" ) == 0 )

line\_info[idx].base\_flag = 1;

else

Pinfo->location += asmd\_process( Pinfo->argu , &line\_info[idx], Pinfo->location);

}

else

return -1;

strcpy( line\_info[idx].asmd, Pinfo->argu[1]);

}

else // 아무것도 아닌 경우 에러 처리

return -1;

\*flag = 1; // start가 나온 경우 start가 처음에 나왓는지 체크

return 1;

}

/\*

\* command type 일 때 file 내용을 보여주는 함수

\*/

void print\_file(char \*filename){

FILE \*fp;

char buffer[500];

fp = fopen(filename, "r");

if ( fp == NULL )

fprintf(stderr, "FILE OPEN ERROR\n");

while (fgets(buffer, sizeof(buffer), fp) != NULL)

printf("%s", buffer);

}

/\*

\* asseble command를 수행하는 함수

\* symbolTable은 symbol Table의 정보를 가지고 있고 success\_symbol이란 변수는 assemble 성공하였을 때 symbol을 저장하는 구조체다.

\* 에러가 있는 경우 return 1

\* 없는 경우 return 0

\*/

int command\_assemble(Symbol\_table \*symbolTable, symbol\_info \*success\_symbol, char \*command){

FILE \*fp;

char buffer[256];

char copy[256];

char \*argu[10];

char \*filename;

char \*ptr;

int type, len, flag = 0; // flag는 시작이 start인지 아닌지 check

size\_t idx = 0;

line\_inform line\_info[200];

Pass1 Pinfo;

filename = strtok(command, " \t");

filename = strtok(NULL, " \t");

ptr = strstr(filename, "."); // get filename

if ( ptr == NULL || strcmp( ptr, ".asm") != 0){ // not assembly file

printf("please input assembly file\n");

return 1; // error

}

fp = fopen(filename, "r");

if( fp == NULL){ // no file error

printf("NO file error!\n");

return 1;

}

for ( int i = 0; i < 200; ++i){ // initialize

line\_info[i].label\_flag = 0;

line\_info[i].location = 0;

}

symbol\_init(symbolTable);

// algorithm for pass1 of assembler

while ( fgets(buffer, sizeof(buffer) , fp) != NULL) {

strtok(buffer, "\n\r\0");

len = strlen(buffer);

if ( len == 0)

continue;

if ( buffer[len - 1] == '\n')

buffer[len - 1] = '\0';

len = strlen(buffer);

if ( len == 0)

continue;

strncpy ( copy , buffer, sizeof(copy) );

type = get\_argu( copy , argu, symbolTable->hashTable); // 해당하는 라인이 어떤 형식인지 구해주는 함수

Pinfo.argu = argu;

//make line function

if ( make\_line(copy, type, idx++, &flag, symbolTable, &Pinfo, line\_info) == -1 )

return 1;

}

fclose(fp);

//pass algorithm function

flag = assembler\_pass2(filename, symbolTable, idx, &Pinfo, line\_info);

if ( flag == -1 ) //start가나왔는데 처음이 아니면

return 1; // error

// assemble 성공한 경우 symbol 저장

idx = 0;

for ( int i = 0; i < symbolTable->size; ++i){

for( symbolPtr sptr = symbolTable->table[i]; sptr != NULL; sptr = sptr->next){

success\_symbol[idx].address = sptr->address;

strcpy(success\_symbol[idx].symbol, sptr->symbol);

idx++;

}

}

success\_symbol[idx].address = -1;

return 0;

}

/\*

\* byte인 경우 object 코드를 생성해주는 함수

\* error가 발견 되면 return 1

\* else return 0

\*/

int obj\_byte(FILE \*fp, Symbol\_table \*symbolTable,

line\_inform \*line\_info, object\_inform \*obj\_info,

int \*obj\_flag, int \*idx){

char c = line\_info->operhand[0]; // 'C' or 'X'저장

char copy[30];

char tmp[5];

char \*error = NULL;

int len = 0;

strcpy(copy, line\_info->operhand);

c = ( 'A' <= c && c <= 'Z') ? c : c - 'a' + 'A';

if ( c == 'C') { // char

if ( strstr(copy, "'") == NULL){ // byte : C'~'

printf("Error in byte\n");

return 1;

}

line\_info->obj\_code = -2; // objcode 생성하기 위한 정보

if ( strlen(copy + 2) > 31 ){ // byte는 최대 31길이

printf("Error in byte length\n");

return 1;

}

for ( int i = 2; copy[i] != 39 && copy[i] != '\0'; ++i){ // byte code 출력

fprintf(fp, "%02X",(int)copy[i]); // byte

sprintf(tmp, "%02X", (int)copy[i]); //

strcat(line\_info->obj\_strcode, tmp);

len++;

}

fprintf(fp, "\n");

}

else if( c == 'X'){ // hex 인 경우

strtol(copy + 2, &error , 16); // 16진수인지 체크

if ( strstr(copy, "'") == NULL || \*error != 39){

printf("Error in byte\n");

return 1;

}

for ( int i = 2; copy[i] != 39 && copy[i] != '\0'; ++i ){

len = i - 1;

if( i >= 62){ // hex 이므로 31 bit 넘는지 체크.

printf("Error in byte\n");

return 1;

}

}

len /= 2;

strtok (copy +2 , "'");

fprintf(fp, "%s\n", copy + 2 );

if(obj\_info[\*idx].size + len <= 30) // objcode information 저장

obj\_info[\*idx].size += len;

else{

obj\_info[++(\*idx)].size += len;

obj\_info[\*idx].start = line\_info->location;

}

line\_info->obj\_code = -2;

}

else{

printf("Error in byte\n");

return 1;

}

if(\*obj\_flag == 1){ // object code 시작줄인 경우

obj\_info[++(\*idx)].start = line\_info->location;

\*obj\_flag = 0;

}

if ( obj\_info[\*idx].size + len <= 30) // object code 길이가 30이하

obj\_info[\*idx].size += len;

else{

obj\_info[(\*idx)].size += len;

obj\_info[\*idx].start = line\_info->location; // object code 새로운 라인

}

return 0;

}

/\*

\* opcode인 경우 object code 생성하는 함수

\* if error found return 1

\* else return 0

\*/

int obj\_opcode(FILE \*fp, Hash \*hashTable, Symbol\_table \*symbolTable,

line\_inform \*line\_info, object\_inform \*obj\_info,

int \*obj\_idx, int \*obj\_flag, int \*arr, int base\_address) {

Hnode hashptr;

symbolPtr sptr;

char tmp1[50] , tmp2[50], tmp3[50];

char \*error;

int regi\_num;

int symbol\_num;

int immediate;

int format = line\_info->format;

fprintf(fp,"%-10s\t%-10s\t",line\_info->opcode,line\_info->operhand);

hashptr = opcode\_find(hashTable, line\_info->opcode); // opcode

if(format == 1) { //format 1인 경우

fprintf(fp,"\n");

line\_info->obj\_code = (int)strtol(line\_info->operhand,NULL,16);

}

else if ( format == 2) { //format 2인 경우

sprintf(tmp1,"%X",hashptr->n\_opcode);

strcpy(tmp2,line\_info->operhand);

if(strstr(tmp2,",") != NULL) {// ','가 있는 경우

regi\_num = get\_register(strtok(tmp2,",")); // register 구함

sprintf(tmp3,"%d", regi\_num);

if(regi\_num == -1) { //error

printf("ERROR in format 2\n");

return -1;

}

strcat(tmp1, tmp3);

regi\_num = get\_register(strtok(NULL,"\t\n\r "));

sprintf(tmp3,"%d",regi\_num);

if(regi\_num == -1) {

printf("ERROR in format 2\n");

return -1;

}

strcat(tmp1,tmp3);

fprintf(fp,"%s\n",tmp1);

line\_info->obj\_code = (int)strtol(tmp1,NULL,16);

}

else { //인자가 하나인 경우

regi\_num = get\_register(tmp2);

sprintf(tmp3,"%d",regi\_num);

if(regi\_num == -1) { // error

printf("ERROR in format 2\n");

return 1;

}

strcat(tmp1,tmp3);

strcat(tmp1,"0");

fprintf(fp,"%s\n",tmp1);

line\_info->obj\_code = (int)strtol(tmp1,NULL,16);

}

}

else if ( format == 3 ) { //format 3인 경우

if(strcmp(line\_info->opcode,"RSUB") == 0) { //RSUB는 특이

fprintf(fp,"4F0000\n");

line\_info->obj\_code = 0x4F0000;

}

else {

strcpy(tmp1, line\_info->operhand);

if(strstr(tmp1,",")!=NULL) { //x를 담당, index mode 확인

if(strstr(tmp1,"X") != NULL || strstr(tmp1,"x") != NULL) {

arr[2] = 1; // x = 1

strtok(tmp1,",");

}

else {

printf("Error in format 3\n");

return 1;

}

}

if(remove\_char(tmp1,symbolTable, &arr[0], &arr[1] ) == -1) { //#, @ 및 n, i를 담당

printf("ERROR in immediate or indirect addressing\n");

return 1;

}

if( ( sptr = symbol\_find(symbolTable, tmp1)) != NULL) { //operhand가 symbol인 경우

symbol\_num = sptr->address;

if(symbol\_num - line\_info->location >= -2048 && symbol\_num - line\_info->location <= 2047) {

arr[4] = 1; // p = 1

symbol\_num = symbol\_num - line\_info->location - 3; //pc 가능 범위에 있으면 pc

if(symbol\_num < 0) // hex code를 위해

symbol\_num = symbol\_num & 0x00000FFF;

}

else if(symbol\_num - base\_address >= 0 && symbol\_num - base\_address <=4095) {

arr[3] = 1; // b = 1

symbol\_num = symbol\_num - base\_address;

}

else { //둘 다 아니면 에러

printf("ERROR in range\n");

return 1;

}

line\_info->obj\_code = get\_objcode( hashptr->n\_opcode, arr[0], arr[1], arr[2],

arr[3], arr[4], arr[5], symbol\_num);

//obj code calculate

fprintf(fp,"%06X\n",line\_info->obj\_code);

}

else { //immediate인 경우

arr[0] = 0; arr[1] = 1;

immediate = (int)strtol(tmp1, &error, 10);

if ( \*error != '\0' ) { //10진수가 아니면

printf("Error in format 3\n");

return 1;

}

if ( immediate > 0x1000 ) {

printf("Error in immediate\n");

return 1;

}

line\_info->obj\_code = get\_objcode(hashptr->n\_opcode, arr[0], arr[1], arr[2],

arr[3], arr[4], arr[5], immediate);

fprintf(fp,"%06X\n",line\_info->obj\_code);

}

}

}

else if ( format == 4 ) { //format 4인 경우

arr[5] = 1; // e = 1

strcpy(tmp2, line\_info->operhand);

strcpy(tmp3, line\_info->opcode);

strcpy(tmp3, tmp3+1);

hashptr = opcode\_find(hashTable, tmp3); // opcode 인지 확인

if(strstr(tmp2,",")!=NULL) {

if(strstr(tmp2,"X") != NULL || strstr(tmp2,"x")) { // index mode check

arr[2] = 1; // x = 1

strtok(tmp2,",");

}

else {

printf("ERROR in format 4\n");

return 1;

}

}

if(remove\_char(tmp2, symbolTable, &arr[0], &arr[1]) == -1) { // immeidate or indirect check

printf("ERROR in immediate or indirect addressing\n");

return 1;

}

sptr = symbol\_find(symbolTable, tmp2); // symbol check

if(sptr != NULL) { //operhand가 symbol인 경우

line\_info->obj\_code = get\_objcode(hashptr->n\_opcode,arr[0], arr[1], arr[2],

arr[3],arr[4],arr[5],sptr->address);

fprintf(fp,"%08X\n",line\_info->obj\_code);

line\_info->modi\_flag = 1;

}

else { //immediate인 경우

symbol\_num = (int)strtol(tmp2,&error,10);

arr[0] = 0; arr[1] = 1;

if(\*error != '\0') { //10진수가 아니면

printf("Error in immediate\n"); //에러 처리

return 1;

}

if(symbol\_num>0x1000) {

printf("ERROR in format 4\n");

return 1;

}

line\_info->obj\_code = get\_objcode(hashptr->n\_opcode,arr[0], arr[1], arr[2],

arr[3],arr[4],arr[5],symbol\_num);

fprintf(fp,"%08X\n",line\_info->obj\_code);

}

}

//ojbect code 만들때 쓸 정보 저장

if(\*obj\_flag == 1) {

obj\_info[++(\*obj\_idx)].start = line\_info->location;

\*obj\_flag = 0;

}

if(obj\_info[\*obj\_idx].size + format <= 30)

obj\_info[\*obj\_idx].size += format;

else {

obj\_info[++(\*obj\_idx)].size += format;

obj\_info[\*obj\_idx].start = line\_info->location;

line\_info->n\_flag = 1;

}

return 0;

}

/\*

\* object file 만드는 함수

\* 위에서 저장한 obj information들을 이용하여 만듬

\*/

void make\_objfile(FILE \*fp, int start\_address, int start\_line, line\_inform \*line\_info, object\_inform \*obj\_info) { //오브젝트 파일 생성 함수

int i = -1, idx = start\_line; //임시 변수

int end\_flag = 0; //end 플래그

fprintf(fp,"H%s %06X%06X",line\_info[idx].symbol, line\_info[idx].location, start\_address); // start line

while(1) { //end가 아닐 때 까지

if(end\_flag == 1)

break;

i++;

fprintf(fp,"\nT%06X%02X", obj\_info[i].start, obj\_info[i].size); // start address and size

while(1) { // line 끝날 대 까지

idx++;

if(get\_asmd(line\_info[idx].asmd) == end) { // 도중에 end 들어온 경우

end\_flag = 1;

break;

}

if(get\_asmd(line\_info[idx].asmd) == resb

|| get\_asmd(line\_info[idx].asmd) == resw) { //resb, resw를 처리하는 부분

if(line\_info[idx].n\_flag == 1) { //줄바꿈을 한다

while(line\_info[++idx].n\_flag == 1);

}

}

else if(line\_info[idx].obj\_code != -1

&& line\_info[idx].obj\_code != -2) {

if(line\_info[idx].format == 2) //format 2는 4글자

fprintf(fp, "%04X", line\_info[idx].obj\_code);

else //아니면 6글자

fprintf(fp, "%06X", line\_info[idx].obj\_code);

}

else if(line\_info[idx].obj\_code == -2) //BYTE의 경우는 문자열을 출력

fprintf(fp, "%s", line\_info[idx].obj\_strcode);

if(line\_info[ idx + 1 ].n\_flag == 1) // 줄바꿈

break;

}

}

for(idx = 0; idx<200; idx++) { // Modification

if(line\_info[idx].modi\_flag == 1) {

fprintf(fp,"\nM%06X05",line\_info[idx].location + 1);

}

}

fprintf(fp,"\nE%06X\n",line\_info[start\_line].location); //마지막줄

}

/\*

\* pass2 algorithm을 구현한 함수이다

\* if error found return 1

\* else return 0

\*/

int assembler\_pass2(char \*filename, Symbol\_table \*symbolTable, int length,

Pass1 \*Pinfo, line\_inform \*line\_info){

FILE \*fp, \*lstfp , \*objfp;

char lstname[30];

char objname[30];

char buffer[256];

char bef[256];

char copy[256];

char \*errorp;

int arr[7];

int linenum = 0, type, num;

size\_t idx = 0, len;

int end\_flag = 0, error = 0, obj\_flag = 0, obj\_idx = 0;

int n, i, x, b, p , e;

int asmd, start\_line;

Hash \*hashTable = symbolTable->hashTable;

symbolPtr sptr;

object\_inform obj\_info[300];

strcpy(copy, filename);

strtok(copy, ".");

strcpy(lstname, copy);

strcat(lstname, ".lst");

strcpy(objname, copy);

strcat(objname, ".obj");

fp = fopen(filename, "r");

lstfp = fopen(lstname, "w"); // lst file pointer

objfp = fopen(objname, "w"); // object file pointer

if ( lstfp == NULL || objfp == NULL){

fprintf(stderr, "FILE OPEN ERROR!\n");

return -1;

}

while ( fgets(buffer, sizeof(buffer), fp) != NULL){

if(error == 1) { //error find code

printf("\*\*\*\*\*\*\*\*\*\*\*\*\* ERROR LINE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("line : %d : %s", linenum , bef);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

fclose(lstfp);

fclose(objfp);

remove(lstname);

remove(objname);

return -1;

}

n = 0, i = 0, x = 0, b = 0, p = 0, e = 0;

strtok(buffer, "\n\r\0");

len = strlen(buffer);

if ( len == 0)

continue;

if ( buffer[len - 1] == '\n')

buffer[len - 1] = '\0';

len = strlen(buffer);

if ( len == 0)

continue;

strcpy(bef, buffer); // error check 할 때 출력할 string

linenum += 5;

fprintf(lstfp,"%-5d\t",linenum); // line number

strcpy(copy, buffer);

asmd = get\_asmd(line\_info[idx].asmd); // assemble directives check

if(line\_info[idx].format == -1) { //comment인 경우

fprintf(lstfp,"\t%s\n",buffer);

idx++;

continue;

}

else if( asmd == base || asmd == end ) { //base나 end인 경우

if( asmd == base) { //base인 경우

if ( ( sptr = symbol\_find(symbolTable, line\_info[idx].operhand)) != NULL)

Pinfo->base\_address = sptr->address; // base address 저장

else {

printf("Error in base\n");

error = 1;

continue;

}

}

else // base 아니면 end

end\_flag = 1;

fprintf(lstfp,"\t\t%-10s\t%-10s\n",

line\_info[idx].asmd,line\_info[idx].operhand);

idx++;

continue;

}

fprintf(lstfp,"%04X\t",line\_info[idx].location); //주소 출력

if ( line\_info[idx].label\_flag ) // label이 있는 경우

fprintf(lstfp, "%s\t", line\_info[idx].symbol); // label 출력

else

fprintf(lstfp, "\t");

if(opcode\_find(hashTable, line\_info[idx].opcode) == NULL &&

opcode\_find(hashTable, (line\_info[idx].opcode)+1) == NULL &&

get\_asmd(line\_info[idx].asmd) == -1) { //opcode 및 assemble directives 둘 다 없는 경우

printf("Error in opcode or symbol\n");

error = 1;

continue;

}

if(get\_asmd(line\_info[idx].asmd) != -1) { //asmd가 있는 경우

fprintf(lstfp,"%-10s\t%-10s\t",line\_info[idx].asmd, line\_info[idx].operhand); //asmd 출력

type = get\_asmd(line\_info[idx].asmd);

if(type == 0) { //start인 경우

line\_info[idx].obj\_code = -1;

fprintf(lstfp,"\n");

obj\_info[obj\_idx].start = line\_info[idx].location; //오브젝트 파일 시작 주소

start\_line = idx;

idx++;

continue;

}

else if(type == 3) { //byte인 경우

error = obj\_byte(lstfp, symbolTable, &line\_info[idx],

obj\_info, &obj\_flag, &obj\_idx);

// object byte code 생성

idx++;

continue;

}

else if(type == 4){ //word인 경우

num = strtol(line\_info[idx].operhand, &errorp, 10);

if(\*errorp != '\0') {

printf("Error in WORD\n");

error = 1;

continue;

}

line\_info[idx].obj\_code = num;

if(line\_info[idx].obj\_code < 0)

line\_info[idx].obj\_code &= 0x00FFFFFF; // 3byte

fprintf(lstfp,"%06X\n",line\_info[idx].obj\_code);

//object code 생성중

if(obj\_flag == 1) {

obj\_info[++obj\_idx].start = line\_info[idx].location; // 라인 시작 주소

obj\_flag = 0;

}

if(obj\_info[i].size + 3 <= 30)

obj\_info[obj\_idx].size += 3; // object code line size check

else {

obj\_info[++obj\_idx].size += 3;

obj\_info[obj\_idx].start= line\_info[idx].location; // 새로운 줄

line\_info[idx].n\_flag = 1;

}

}

else if(type == 5) { //resb인 경우

num = strtol(line\_info[idx].operhand, &errorp, 10);

if( \*errorp != '\0') {

printf("Error in WORD\n");

error = 1;

continue;

}

fprintf(lstfp,"\n");

line\_info[idx].n\_flag = 1;

obj\_flag = 1;

}

else if(type == 6) { //resw인 경우

num = strtol(line\_info[idx].operhand, &errorp, 10);

if( \*errorp != '\0') {

printf("Error in RESW\n");

error = 1;

continue;

}

fprintf(lstfp,"\n");

line\_info[idx].n\_flag = 1;

obj\_flag = 1;

}

}

else { //opcode 인 경우

arr[0] = n; arr[1] = i; arr[2] = x; arr[3] = b; arr[4] = p; arr[5] = e;

error = obj\_opcode(lstfp, hashTable, symbolTable, &line\_info[idx],

obj\_info,&obj\_idx, &obj\_flag, arr, Pinfo->base\_address);

if ( error )

continue;

}

idx++;

}

if ( !end\_flag && !error){

printf("NO END in assemble code\n");

remove(lstname);

remove(objname);

return -1;

}

// 위에서 저장한 obj information으로 작성

make\_objfile(objfp, Pinfo->program\_len , start\_line, line\_info, obj\_info);

printf("\toutput file : [%s], [%s]\n", lstname, objname);

fclose(lstfp);

fclose(objfp);

return 1;

}

/\*

\* 현재 디렉토리에 있는 filename 파일을 읽어서 화면에 출력한다.

\* 현재 디렉토리에 존재하지 않거나 디렉토리인 경우에는 에러 메세지를 출력한다.

\*/

int command\_type(char \*buffer){

DIR \*dirp;

struct dirent \*direntp;

struct stat buf;

int flag = -1;

char \*filename;

filename = strtok(buffer, " \t");

filename = strtok(NULL, " \t");

if(filename == NULL){

printf("Please input filename\n");

return -1;

}

if( (dirp = opendir(".")) == NULL){ //diropen error check

printf("Can not Open Directory\n");

return 1;

}

while( (direntp = readdir(dirp)) != NULL){ // dir에 있는 모든 파일을 읽을 때 까지

stat(direntp->d\_name, &buf);

if( S\_ISDIR(buf.st\_mode) ) //directory인 경우

continue;

else if( strcmp (direntp->d\_name, filename ) == 0){

flag = 1;

print\_file(filename);

}

}

closedir(dirp);

if ( flag == 1)

return 1;

else{

printf("type command error!\n");

return -1;

}

}

/\*

\* symbol command 수행하는 함수이다.

\* 각 symbol의 주소값을 출력한다.

\*/

void command\_symbol(symbol\_info \*symbolTable) {

int size = 0;

for ( int i = 0; i < 500; ++i){

if ( symbolTable[i].address == -1)

size = i;

}

if(size != 0)

print\_symbol(symbolTable, size);

else

printf("NO SYMBOL TABLE\n");

}