**과목명: 시스템프로그래밍**

**1반**

**<<Project #2>>**

**서강대학교 컴퓨터공학과**

**20161563**

**고성빈**

목 차

1 프로그램 개요

2 프로그램 설명

2.1 프로그램 흐름도(종합)

2.2 프로그램 흐름도(프로젝트 2 상세)

3 모듈 및 전역 변수 정의[[1]](#footnote-1)

3.1 메인 함수 변경점

3.1.1 int main()

3.2 help 명령어 변경점

3.3.1 int function\_help()

3.3 assemble 명령어 part 1 – symbol table 구현

3.3.1 구조체 및 전역 변수

3.3.2 int function\_assemble()

3.3.3 void free\_symbol\_table()

3.3.4 int make\_symbol\_table(char\*)

3.3.5 int scan\_asm(FILE\* fp, char\*, int\*, int)

int insert\_symbol(char\*, int)

3.4 assemble 명령어 part 2 – list file 구현

3.4.1 구조체 및 전역 변수

3.4.2 int make\_lst()

3.4.3 void set\_asm\_line()

3.4.4 void reset\_asm\_line()

3.4.5 void insert\_asm\_line(int addr, char\* object code)

3.4.6 void free\_asm\_line()

3.4.7 int symbol\_address(char\*)

3.4.8 int reg(char\*)

3.4.9 int find\_address(char\*, int\*)

3.5 assemble 명령어 part 3 – object file 구현

3.5.1 구조체 및 전역 변수

3.5.2 int make\_obj()

3.5.3 void print\_obli(FILE\* fp)

3.5.4 void print\_relo(FILE\* fp)

3.5.5 void set\_relocate()

3.5.6 void insert\_relocate(int, int)

3.5.7 void free\_relocate()

3.6 type 명령어 구현

3.6.1 int function\_type()

3.7 symbol 명령어 구현

3.7.1 int function\_symbol()

3.7.2 void set\_symbol\_top()

3.7.3 void free\_symbol\_top()

4 코드 설명

4.1 20161563.h 추가 사항

4.2 20161563.c 추가 사항

1 프로그램 개요

이 프로그램은 프로젝트 1을 통해 구현한 셸(Shell)에 assemble 기능을 추가하는 프로그램이다. SIC/XE의 assembly program source 파일을 입력받아 list 파일과 object 파일을 생성하고, 어셈블리 과정 중 생성된 symbol table과 결과물인 object 파일을 볼 수 있는 기능을 제공한다. 교재의 2.2까지 설명된 SIC/XE 어셈블러의 기능을 지원한다.

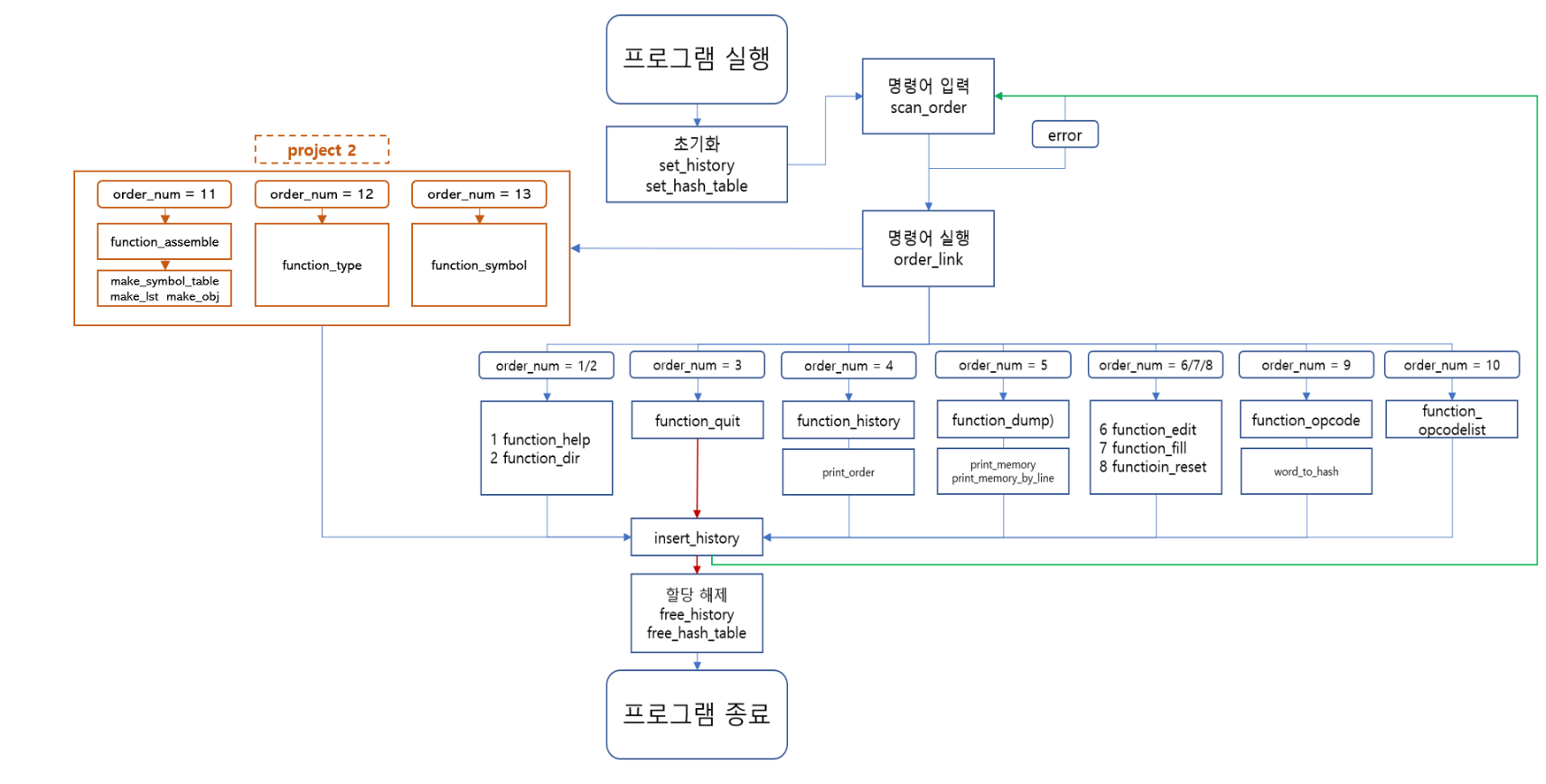
2 프로그램 설명

프로그램은 실행되고 나서 초기화 단계를 거친 후, 종료되기 전까지 다음과 같은 단계를 반복한다.

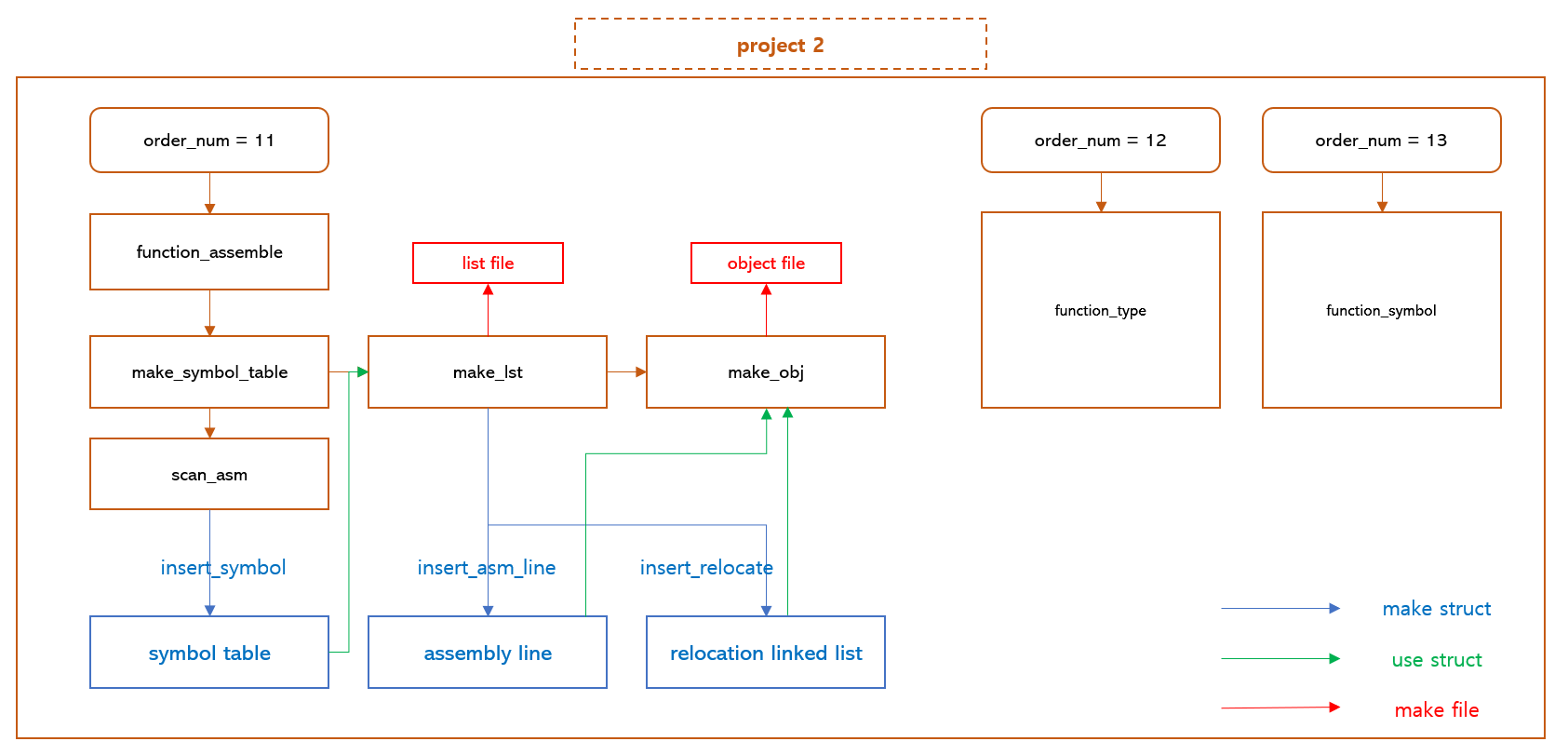
scan\_order로 명령어를 입력받고, order\_link를 통해 해당 명령어의 함수를 실행한다. 해당 함수가 종료된 후 insert\_history 함수를 통해 입력 내용을 기록하고 다시 명령어 입력 단계로 돌아온다. quit 명령어가 입력된 경우에만 반복문을 종료하고, 할당된 메모리를 해제한 뒤 프로그램을 종료한다.

프로젝트 2를 통해 기존의 프로그램에 assemble 기능이 추가되어 어셈블리 파일을 object 파일로 수정할 수 있게 되었다.

2.1 프로그램 흐름도(종합)

프로그램 흐름도는 다음과 같다.

2.2 프로그램 흐름도(프로젝트 2 상세)

위 프로그램에서 project 2 부분을 더 자세하게 표현하면 다음과 같다.

3 모듈 정의

3.1 메인 함수 변경점

프로그램에서 지원하는 명령어의 종류가 늘어남에 따라 기존의 메인 함수에서 실행하던 initialize 기능과 프로그램 종료시의 할당 해제 기능이 추가되었다. 새로운 구조체로 assemble 기능에서의 symbol table을 위한 smbtable, object code를 줄 단위로 저장하기 위한 asmline, format 4 형태에서의 relocation을 위한 relocate linked list를 initialize해주고, 프로그램의 종료시에 할당 해제해준다.

3.1.1 int main()

- main 함수는 set\_history와 set\_hash\_table 함수를 통한 linked list 생성과 반복문 내에서 명령어

입력(scan\_order)과 실행(order\_link) 함수 호출, 이후 프로그램 종료 전 free\_history와

free\_hash\_table을 통해 linked list 할당 해제하는 기능을 한다.

- 추가 함수 :

void set/free\_symbol\_top() : symbol table 작성을 위한 linked list를 메모리 할당/할당 해제한다.

void set/free\_asm\_line() : object code 저장을 위한 linked list를 메모리 할당/할당 해제한다.

void set/free\_relocate() : relocation을 위한 linked list를 메모리 할당/할당 해제한다.

3.2 help 명령어 변경점

기존 10개의 명령어를 출력하던 help 명령어는 프로젝트 2에서 추가된 3개의 명령어 assemble, type, symbol을 추가로 출력해준다.

3.2.1 int function\_help()

- function\_help 함수는 rest\_exist를 확인해 추가적인 인자가 있는지 보고, 없을 경우 명령어 목록을

출력한다. 인자가 있을 경우에는 에러처리한다.

3.3 assemble 명령어 part 1 – symbol table 구현

assemble 명령어는 1개의 인자를 받아 해당 인자의 이름을 가진 assembly 파일을 입력받아 파일을 통해 symbol table과 list 파일을 거쳐 object 파일을 만든다.

symbol table을 구현하는 과정에서 assembly 파일을 읽으며 각 줄에 label이 있는지 확인하고, 있을 경우 해당 label의 이름과 시작 주소를 symbol table에 linked list의 형태로 저장해준다. 시작주소 처리를 위해서 opcode의 format을 확인해 해당 명령어가 몇 바이트를 확인해주어야 한다.

3.3.1 구조체 및 전역 변수

struct smbtable : char name[30](label의 이름 저장), int addr(label의 시작 주소값 저장),

smbtable\* next(linked list의 다음 노드 저장)

sptr top : symbol table의 head를 가리킨다.

3.3.2 int function\_assemble()

- function\_assemble 함수는 파일의 이름을 입력받아 해당 파일이 assembly 파일일 경우 그 파일을 분석하기 위한 함수를 차례로 호출한다. 명령어 assemble 뒤에 파일명이 오지 않거나, 파일이 .asm으로 종료하지 않는 경우, 그리고 make\_symbol\_table 함수에서 에러가 발생했을 경우 에러처리하고 -1을 반환한다.

- 사용 변수

char filename[80] : assemble할 assembly 파일의 이름을 저장한다.

- 호출 함수

free\_symbol\_table : 새 assembly 파일로 symbol table을 만들기 전 미리 symbol table의 top을 제외한 나머지 부분을 할당 해제한다.

make\_symbol\_table(filename) : assembly 파일을 읽고 비어있는 symbol table을 채운다. 문제 상황이 발생할 경우 -1을 반환한다.

make\_lst(filename) : assembly 파일과 symbol table을 확인하며 list 파일과 object code linked list, relocation linked list를 작성한다.

make\_obj(filename) : object code linked list, relocation linked list를 읽어 object file을 작성한다.

3.3.3 void free\_symbol\_table()

- free\_symbol\_table 함수는 sptr top 뒤로 할당되어 있는 파일의 이름을 입력받아 해당 파일이 assembly 파일일 경우 그 파일을 분석하기 위한 함수를 차례로 호출한다.

3.3.4 int make\_symbol\_table(char\* filename)

- make\_symbol\_table 함수는 sptr top 뒤로 할당되어 있는 파일의 이름을 입력받아 해당 파일이 assembly 파일일 경우, 각 줄을 읽어들여 symbol table을 작성한다. filename에 해당하는 이름이 없을 경우, scan\_asm 에서 opcode를 인식하는 데 문제가 있었을 경우 에러처리하고 -1을 반환한다.

- 사용 변수

FILE\* fp : assembly 파일에서 입력을 받기 위한 파일 포인터가 저장된다.

char fname[10] : assembly 프로그램의 첫 줄에 나오는 프로그램의 이름을 저장한다.

char ifst[10] : assembly 프로그램의 첫 줄이 START로 시작하는지 확인한다.

int faddr : 한 줄씩 assembly 파일을 읽을 때 시작 주소를 가리킨다.

int cur\_line : list 파일 기준으로 현재 줄을 나타내 준다.(현재 줄 번호 \* 5)

char vname[10] : 매 줄을 읽을 때 label의 이름을 저장받는다.

int ch : scan\_asm 파일의 반환값을 저장받는다. END가 오면 1, 에러가 발생하면 -1이 저장된다.

- 호출 함수

int scan\_asm(fp, vname, &faddr, cur\_line) : label 이외의 추가적인 정보를 읽어들이고 symbol table에 추가해준다.

3.3.5 int scan\_asm(FILE\* fp, char\* filename, int\* faddr, int cur\_line)

- scan\_asm 함수는 label의 이름을 입력받은 상태에서 opcode와 적절한 operand를 읽어들인다. label이 있을 경우 symbol table에 저장하고, 명령어의 사이즈에 맞게 faddr을 증가시켜준다. 또한 assembly 파일이 부적절한 경우, 또는 포맷 4를 가질 수 없는 opcode의 앞에 +가 붙은 경우 -1을 반환한다.

- 사용 변수

char vopc[10] : opcode를 저장받는다. 변수의 경우 변수 타입을 저장받는다.

char data[10] : opcode의 뒤에 오는 operand나 constant 변수의 내용, 배열의 길이를 저장한다.

int size : vopc에 변수 배열이 저장된 경우, data에 character형으로 저장된 숫자를 integer형으로 변환해 저장한다.

int type : vopc가 opcode인 경우 명령어의 포맷을 저장한다.

int hashnum : vopc가 opcode인 경우 word\_to\_hash 함수를 사용해 opcode 값을 받아온다.

- 호출 함수

int insert\_symbol(vname, \*faddr) : label 이름과 시작 주소를 symbol table에 추가한다.

int word\_to\_hash(vopc) : hash table을 탐색해opcode 값을 반환한다.

3.3.6 int insert\_symbol(char\* vname, int faddr)

- insert\_symbol 함수는 label의 이름과 시작 주소를 symbol table에 추가해준다. symbol table은 symbol 명령어를 통해 출력 시 알파벳순으로 되어있어야 하므로, symbol table은 알파벳 첫 글자로 0~25까지의 index를 나누어 그 안에 사전순으로 정렬해 준다.

- 사용 변수

sptr new : symbol table linked list를 탐색해 현재 label이 들어갈 위치를 찾아 새 structure을 할당한다.

3.4 assemble 명령어 part 2 – list file 구현

assemble 명령어의 list file 구현 부분은 앞서 제작한 symbol table과 함께 assembly 파일을 다시 읽으며 list 파일을 작성한다. list 파일에는 라인 번호(5단위), 그리고 assembly 파일의 내용과 object code가 들어가기 때문에 각 줄별로 현재 라인, 현재 address를 확인해야 하고, object file 작성을 위해 object code를 저장해주어야 한다. 또한 relocation이 발생하는 format 4에 대해서 relocate table에 저장해준다.

3.4.1 구조체 및 전역 변수

struct asmline : assembly 파일의 정보를 라인 단위로 저장한다. int addr(해당 라인의 시작 주소값

저장), char obco[20](해당 라인의 object code 저장), lptr next(linked list의 다음 노드 주소 저장)

struct relocate : relocation 해줄 라인을 저장한다. int addr(해당 라인의 시작 주소값 저장), int

leng(relocation 해줄 길이를 저장한다), lptr next(linked list의 다음 노드 주소 저장)

lptr(asmline\*) lhead : asmline의 헤드를 저장한다.

lptr lnow : asmline의 마지막 노드를 가리킨다.

rptr(relocate\*) rhead : relocate의 헤드를 저장한다.

rptr rtail : relocate의 마지막 노드를 저장한다.

char asmname[10] : assembly 파일의 프로그램명을 저장한다.

int asmst : assembly 파일의 시작 address를 저장한다.

int asmen : assembly 파일의 마지막 address를 저장한다.

3.4.2 int make\_lst(char\* filename)

- make list 함수는 assembly 파일명을 전달받아 해당 파일명으로 list 파일을 만든다. 이후 label을 제외하고 opcode 또는 변수명을 assembly 파일에서 읽어들인다. opcode인 경우 format을 확인해 operand를 받아주고, 변수일 경우 변수 타입과 그 데이터/배열 크기를 입력받는다. 이후 받은 정보로 object code를 만들어 asmline 구조체에 저장한다.

- 사용 변수

FILE\* fp : \*.asm 파일에서 데이터를 읽어오는데 사용하는 포인터이다.

char fname[10] : 프로그램 이름을 저장한다.

char fstart[10] : 프로그램 이름 뒤에 오는 START를 저장한다. START가 아니면 에러처리한다.

char saddr[10] : start address값을 문자열 형태로 저장한다.

char obli[20] : 한 라인의 object code를 저장한다.

int faddr : saddr에 저장된 값을 정수형으로 바꾸어 저장한다. 이후 줄이 넘어갈 때마다 해당 줄의

시작 address를 저장한다.

char lstname[10] : list파일의 \*.asm 을 \*.lst로 수정해 저장한다.

FILE\* fw : list 파일을 write 형식으로 열어 입력을 위한 파일 포인터를 저장한다.

int base : base의 위치를 가리킨다. -4096으로 초기화를 해주고, base가 나오면 base 값을 바꾼다

base가 나오지 않을 경우 함수의 종료 시에 경고 메시지를 출력한다.

char vname[10] : label을 저장받는다.

char inp[100] : vname이 . 이어서 주석일 경우 . 뒤에 오는 내용을 저장한다.

char vopc[10] : vname이 .이 아닐 경우 opcode를 저장받는다.

char data[10] : opcode가 데이터 타입일 경우, 해당 데이터를 저장받는다.

int type : opcode가 menmonic일 경우, 어느 format인지를 저장받는다.

int hashnum : word\_to\_hash함수를 통해 opcode 값을 저장한다.

optr(= optable\*) now = opcode hash table을 탐색하는 데 사용한다.

char voper[10] : format 3/4일 경우

int obco : opcode와 ni 값을 8bit로 저장한다.

int xbpe : xbpe 값을 4bit로 저장한다.

int ta : target address 값을 저장한다.

- 호출 함수

void reset\_asm\_line() : 이전 assemble에서 사용한 linked list를 초기화한다.

void insert\_asm\_line(int faddr, char\* obli) : 라인의 데이터를 linked list에 추가한다.

int word\_to\_hash(char\* vopc) : mnemonic의 opcode를 반환한다.

int find\_address(voper, &ifnum) : operand를 symbol table에서 탐색해 주소값을 반환한다.

3.4.3 void set\_asm\_line()

- 프로그램의 시작시 asm\_line linked list의 initialize 역할을 한다. lhead를 malloc한다.

3.4.4 void reset\_asm\_line()

- 새 assemble을 시작할 시 asm\_line linked list를 비워준다. lhead를 제외한 이후의 노드를 모두free해준다.

3.4.5 void insert\_asm\_line(int addr, char\* obco)

- 주소값과 object code를 입력받아 asm\_line의 마지막 값을 가리키는 lnow의 다음 노드를 만든다.

3.4.6 void free\_asm\_line()

- 프로그램 종료 시 asm\_line linked list의 모든 노드를 할당 해제한다. reset\_asm\_line으로 헤드를 제외한 노드를 할당 해제한 뒤 헤드를 할당 해제한다.

- 호출 함수

reset\_asm\_line() : linked list의 헤드를 제외한 함수를

3.4.7 int symbol\_address(char\*)

- symbol의 이름을 입력받아서 symbol table을 탐색해 시작 address를 반환한다.

3.4.8 int reg(char\* name)

- register의 이름을 입력받아 register number를 반환한다.

3.4.9 int find\_address(char\* name, int\* ifnum)

- 이름을 입력받아 symbol\_address를 호출해 주소값을 반환받고, 만약 반환받지 못했다면

immediate addressing으로 숫자를 받은 것이므로 숫자를 integer형으로 반환한다.

- 호출 함수

int symbol\_address(name) : 이름으로 address를 탐색한다. 만약 name에 숫자가 저장되어있으면

0을 반환한다.

3.5 assemble 명령어 part 3 – object file 구현

asseble의 part 3부분에서는 앞서 part 2 에서 작성한 asm\_line 구조체와 relocate 구조체를 읽고 object 파일을 작성한다.

3.5.1 구조체 및 전역 변수

struct asmline : assembly 파일의 정보를 라인 단위로 저장한다. int addr(해당 라인의 시작 주소값

저장), char obco[20](해당 라인의 object code 저장), lptr next(linked list의 다음 노드 주소 저장)

struct relocate : relocation 해줄 라인을 저장한다. int addr(해당 라인의 시작 주소값 저장), int

leng(relocation 해줄 길이를 저장한다), lptr next(linked list의 다음 노드 주소 저장)

lptr(asmline\*) lhead : asmline의 헤드를 저장한다.

rptr(relocate\*) rhead : relocate의 헤드를 저장한다.

3.5.2 int make\_obj(char\* filename)

- object 파일을 순서에 맞게 출력하는 함수이다. Head와 END의 출력은 직접 하고, object line은 print\_obli 함수를 통해, relocation line은 print\_relo 함수를 통해 실행하고 relocation table을 초기화한다.

- 사용 변수

FILE\* fw : 내용을 출력할 object 파일의 포인터이다.

- 호출 함수

void print\_obli(FILE\* fw) : object line을 object 파일에 출력한다.

void print\_relo(FILE\* fw) : relocation line을 object 파일에 출력한다.

void free\_relocate() : relocate table을 지운다.

void set\_relocate() : relocate table의 헤드를 작성한다.

3.5.3 void print\_obli(FILE\* fw)

- object code 라인을 \*.obj 파일에 출력한다. line\_buffer를 통해 한 줄에 최대로 들어올 수 있는 바이트 수를 잡고, 명령어의 길이가 line\_buffer보다 길어지거나 배열로 빈 공간이 생기기 전까지 해당 줄에 출력한다. asm\_line의 노드를 전부 출력할 때까지 반복한다.

- 사용 변수

int line\_buf = 30 : 라인 버퍼의 길이. 한 줄에 최대 30 바이트까지 들어올 수 있다.

int saddr : 프로그램의 시작 주소값을 저장한다.

int faddr : 현재 출력해야 할 주소값을 저장한다.

char fr[10] : front. object 파일의 앞부분을 저장한다. T와 6자리의 16진수 시작 address, 현재라인의 길이를 문자열로 저장한다.

char bc[10] : back. object 파일의 뒷부분을 저장한다. 차례로 asm\_line의 노드를 읽어가며 그object code를 저장한다.

3.5.4 void print\_relo(FILE\* fw)

- relocation code 라인을 \*.obj 파일에 출력한다. relocate의 하나의 노드를 한 줄에 출력한다.

3.5.5 void set\_relocate()

- relocate linked list의 헤드를 생성한다.

3.5.6 void insert\_relocate(int addr, int leng)

- part 2에서 호출된다. relocation이 필요한 object code가 들어올 때 relocate linked list에 새로운 데이터를 추가한다.

3.5.7 void free\_relocate()

- relocate linked list를 모두 할당 해제해준다.

3.6 type 명령어 구현

type 명령어는 하나의 파일명을 인자로 가지며, 해당 파일의 내용을 읽어 출력해준다. 만약 해당 파일이 읽을 수 없는 상태라면 파일을 찾을 수 없다고 에러 메시지를 출력하고, 디렉토리의 이름을 입력받았다면 에러 메시지를 출력한다.

3.5.1 int function\_type()

- 파일명을 입력받아 그 파일의 마지막까지 모든 내용을 출력한다. 만약 일치하는 파일명이 없거나 입력

받은 이름이 디렉토리면 에러 메시지를 출력한 뒤 종료한다.

- 사용 변수

char filename[80] : 화면에 출력해줄 파일명을 저장한다.

FILE\* fp : 파일명에 해당하는 파일을 읽기 전용으로 연 뒤 그 포인터를 저장한다.

struct star buf : 파일명에 해당하는 정보를 저장한다. 이를 가지고 디렉토리인지 폴더인지 확인한다.

3.7 symbol 명령어 구현

symbol 명령어는 인자를 가지지 않으며, 가장 마지막으로 수행한 assemble 명령의 symbol table을 화면에 출력해준다. 만약 한 번도 assemble 명령을 수행하지 않았거나, 가장 마지막으로 수행한 assemble 명령이 실패했다면 symbol table을 읽을 수 없다는 메시지를 출력하고 종료한다.

3.7.1 int function\_symbol()

- symbol table에는 index 0~25에 알파벳 A부터 Z로 시작하는 symbol들이 사전순으로 들어가 있다. 테이블의 앞 index부터 차례로 head에서 마지막까지 출력하면 모든 symbol이 사전 순으로 출력된다.

3.7.2 void set\_symbol\_top()

- 프로그램 시작 시 symbol의 헤드를 할당해준다.

3.7.3 void free\_symbol\_top()

- 프로그램 종료 전 symbol의 헤드를 할당 해제한다.

4 코드 설명

4.1 20161563.h 추가 사항

/\* structures \*/

~~~ line 23

typedef struct \_smbtable {

char name[30];

int addr;

struct \_smbtable\* next;

} smbtable;

typedef smbtable\* sptr;

typedef struct \_asmline {

int addr;

char obco[20];

struct \_asmline\* next;

} asmline;

typedef asmline\* lptr;

typedef struct \_relocate {

int addr;

int leng;

struct \_relocate\* next;

} relocate;

typedef relocate\* rptr;

/\* functions \*/

~~~ line 97

// order assemble

int function\_assemble();

// part 1 symbol table

void free\_symbol\_table(); // reset symbol table before assemble

int make\_symbol\_table(char\*); // make new symbol table

int scan\_asm(FILE\*, char\*, int\*, int); // scan opcode and others by 1 line

int insert\_symbol(char\*, int); // insert symbol to table

// part 2 list file

int make\_lst();

void set\_asm\_line(); // make head pointer of asmline

void reset\_asm\_line(); // delete data of asmline

void insert\_asm\_line(int, char\*); // insert data of asmline

void free\_asm\_line(); // free all asmline

int symbol\_address(char\*); // get symbol name and return symbol address

int reg(char\*); // get register name and return register number

int find\_address(char\*, int\*); // get name(or number) and return its address

// part 3 object file

int make\_obj();

void print\_obli(FILE\*);

void print\_relo(FILE\*);

void set\_relocate();

void insert\_relocate(int, int);

void free\_relocate();

// order type

int function\_type();

// order symbol

int function\_symbol();

void set\_symbol\_top();

void free\_symbol\_top();

/\* global variables \*/

~~~ line 150

// symbol table

sptr top;

// asm linked list

lptr lhead, lnow;

char asmname[10];

int asmst, asmen;

// relocation list

rptr rhead, rtail;

4.2 20161563.c 추가 사항

~~~ line 762

/\* assemblers \*/

/\* assemble(11) : get assembly file and make list & object files \*/

int function\_assemble()

{

// check additional word -> if not, error.

if(rest\_exist == 0)

{

printf("input valid order\n");

return -1;

}

// get name of file

char filename[80];

scanf("%s", filename);

scanf("%\*[^\n]");

scanf("%\*c");

int flen = strlen(filename);

// check if input is .asm file

if(strcmp(filename + flen - 4, ".asm") != 0)

{

printf("input assembly file(end with \*.asm)\n");

return -1;

}

// check if filename is available

FILE\* fp = fopen(filename, "r");

if(fp == NULL)

{

printf("cannot find file\n");

return -1;

}

// erase previous table make new symbol table

free\_symbol\_table();

if(make\_symbol\_table(filename) == -1) // make symbol table with error check

{

// in error case, free symbol table and exit

free\_symbol\_table();

return -1;

}

// make list file

if(make\_lst(filename) == -1)

{

free\_symbol\_table();

free\_relocate();

set\_relocate();

return -1;

}

// make object file

make\_obj(filename);

insert\_history(11, filename);

return 0;

}

/\* assemble part 1 : symbol table \*/

// free previous symbol table before making new symbol table

void free\_symbol\_table()

{

for(int i = 0; i < 26; i++)

{

sptr now = (top+i)->next;

while(now != NULL)

{

sptr temp = now->next;

free(now);

now = temp;

}

(top+i)->next = NULL;

}

return;

}

// read file and make new symbol table

int make\_symbol\_table(char\* filename)

{

FILE\* fp = fopen(filename, "r");

char fname[10] = {0, }, ifst[10];

int faddr;

// read first line

fscanf(fp, "%s %s %X", fname, ifst, &faddr); // scan "name" "start" "addr"

fscanf(fp, "%\*c");

if(strcmp(ifst, "START") != 0)

{

printf("this program has no START at first line\n");

fclose(fp);

return -1;

}

int cur\_line = 10;

while(1)

{

char vname[10] = {0, };

if(fscanf(fp, "%c", &vname[0]) == EOF) // read line

{

// if program has no END

printf("this program has no END\n");

fclose(fp);

return -1;

}

// begin with . : remark, faddr doesn't change

if(vname[0] == '.')

{

char c;

fscanf(fp, "%c", &c);

if(c == '\n');

else

fscanf(fp, "%\*[^\n]%\*c");

}

else

{

if(vname[0] == ' ')

vname[0] = 0;

else

if(fscanf(fp, "%s", vname + 1) == EOF)

{

// if program has no END

printf("this program has no END\n");

fclose(fp);

return -1;

}

// scanf opcode and others, change address

int ch = scan\_asm(fp, vname, &faddr, cur\_line);

if(ch == 1)

break;

else if(ch == -1)

// already sent error message in scan\_asm

return -1;

}

cur\_line += 5;

}

fclose(fp);

return 0;

}

// get opcode & operand in one line

int scan\_asm(FILE\* fp, char\* vname, int\* faddr, int cur\_line)

{

// get word and check opcode(type)/variable

char vopc[10];

fscanf(fp, "%s", vopc);

if(strcmp(vname, "") != 0)

if(insert\_symbol(vname, \*faddr) == -1)

{

printf("at line %d, unvalid symbol\n", cur\_line + 5);

return -1;

}

// case for variables

if (strcmp(vopc, "BYTE") == 0)

{

char data[10];

fscanf(fp, "%s", data);

fscanf(fp, "%\*c");

int dsize = strlen(data) - 3;

if (data[0] == 'C')

\*faddr += dsize;

else if (data[0] == 'X')

\*faddr += dsize / 2;

}

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

{

char data[10];

fscanf(fp, "%s", data);

fscanf(fp, "%\*c");

\*faddr += 3;

}

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

{

char data[10];

fscanf(fp, "%s", data);

fscanf(fp, "%\*c");

int size = 0;

for(int i = 0; i < strlen(data); i++)

{

size \*= 10;

size += data[i] - '0';

}

\*faddr += size;

}

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

{

char data[10];

fscanf(fp, "%s", data);

fscanf(fp, "%\*c");

int size = 0;

for(int i = 0; i < strlen(data); i++)

{

size \*= 10;

size += data[i] - '0';

}

\*faddr += 3 \* size;

}

// special case : BASE (no adress increase)

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

fscanf(fp, "%\*[^\n]%\*c");

// special case : RSUB (no operand behind)

else if (strcmp(vopc, "RSUB") == 0)

{

fscanf(fp, "%\*c");

\*faddr += 3;

}

// special case : END (last of program)

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

{

return 1;

}

// case for opcode

else

{

// check opcode start with + : type 4

int type = 0;

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

{

type = 4;

strcpy(vopc, vopc + 1);

}

// check the opcode is in hash table and its type

int hashnum = word\_to\_hash(vopc);

optr now = opcode[hashnum];

while (1)

{

// name not found, error case

if (now == NULL)

{

printf("at line %d, opcode error\n", cur\_line);

fclose(fp);

return -1;

}

// name found, check it's type

if (strcmp(now->name, vopc) == 0)

{

// type 1~3

if (type == 0)

type = now->data[0] - '0';

// type 3 with + -> 4

else if (now->data[0] == '3');

// type 1 or 2 with + -> error

else

{

printf("at line %d, format error\n", cur\_line);

fclose(fp);

return -1;

}

\*faddr += type;

break;

}

now = now->next;

}

// dump rest data

if(type == 1) // type 1 get '\n'

fscanf(fp, "%\*c");

else // other types

fscanf(fp, "%\*[^\n]%\*c");

}

return 0;

}

// insert variable name to symbol table

int insert\_symbol(char\* vname, int faddr)

{

if(vname[0] < 'A' || vname[0] > 'Z')

return -1;

sptr now = top + vname[0] - 'A';

sptr prev = now;

while(1)

{

// vname is faster : insert vname between prev & now

if(strcmp(vname, now->name) == -1)

{

sptr new = (sptr)malloc(sizeof(smbtable));

strcpy(new->name, vname);

new->addr = faddr;

prev->next = new;

new->next = now;

break;

}

// 2 same name, error

else if(strcmp(vname, now->name) == 0)

{

return -1;

}

// vname is slower & last : make vname come last

else if(now->next == NULL)

{

sptr new = (sptr)malloc(sizeof(smbtable));

strcpy(new->name, vname);

new->addr = faddr;

now->next = new;

new->next = NULL;

break;

}

// vname is slower & now last : can search more

else

{

prev = now;

now = now->next;

}

}

return 0;

}

/\* assemble part 2 : list file \*/

// make list file

int make\_lst(char\* filename)

{

FILE\* fp = fopen(filename, "r");

char fname[10], fstart[10], saddr[10];

fscanf(fp, "%s %s %s%\*c", fname, fstart, saddr);

strcpy(asmname, fname);

reset\_asm\_line();

// object code of line

char obli[20] = {0, };

// start address(str\_hex to num)

int faddr = 0;

for(int i = 0; i < strlen(saddr); i++)

{

faddr \*= 16;

faddr += saddr[i] - '0';

}

asmst = faddr;

// make name of list file

char lstname[10];

strcpy(lstname, filename);

strcpy(lstname + strlen(lstname) - 3, "lst");

FILE\* fw = fopen(lstname, "w");

fprintf(fw, "%-8s %-4s %8s %-15s %14s %s\n\n", "Line", "Loc", "", "Source statement", "", "Object Code");

fprintf(fw, "%4d %04X %4s %-10s %-10s %X\n", 5, faddr, "", fname, fstart, faddr);

int base = -4096; // after BASE comes, saves its register

// print by line

for(int i = 10; ; i += 5)

{

fprintf(fw, "%4d ", i);

char vname[10] = {0, };

fscanf(fp, "%[^\n' ']", vname);

// line with remark

if(vname[0] == '.')

{

char inp[100];

fscanf(fp, "%c", &inp[0]);

if(inp[0] == '\n')

fprintf(fw, "%10s.", "");

else

{

fscanf(fp, "%[^\n]%\*c", inp);

fprintf(fw, "%10s.%5s%s", "", "", inp);

}

fprintf(fw, "\n");

}

// line without remark (variable / opcode / base,end)

else

{

// get word and check opcode(type)/variable

char vopc[10];

fscanf(fp, "%s", vopc);

// variable

if(strcmp(vopc, "BYTE") == 0)

{

char data[10];

fscanf(fp, "%s%\*c", data);

fprintf(fw, "%04X %-10s %-10s %-15s", faddr, vname, vopc, data);

if(data[0] == 'C')

{

for(int i = 2; i < strlen(data) - 1; i++)

sprintf(obli + 2\*i - 4, "%02X", data[i]);

fprintf(fw, "%s", obli);

insert\_asm\_line(faddr, obli);

faddr += strlen(data) - 3;

}

else if(data[0] == 'X')

{

for(int i = 2; i < strlen(data) - 1; i++)

sprintf(obli + i - 2, "%c", data[i]);

fprintf(fw, "%s", obli);

insert\_asm\_line(faddr, obli);

faddr += (strlen(data) - 3) / 2;

}

}

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

{

char data[10];

fscanf(fp, "%s%\*c", data);

fprintf(fw, "%04X %-10s %-10s %-15s", faddr, vname, vopc, data);

faddr += 3;

int dec = 0;

for(int i = 0; i < strlen(data); i++)

{

dec \*= 10;

dec += data[i] - '0';

}

sprintf(obli, "%06X", dec);

fprintf(fw, "%s", obli);

insert\_asm\_line(faddr, obli);

}

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

{

char data[10];

fscanf(fp, "%s%\*c", data);

// str\_dec to num

int dec = 0;

for(int i = 0; i < strlen(data); i++)

{

dec \*= 10;

dec += data[i] - '0';

}

fprintf(fw, "%04X %-10s %-10s %d", faddr, vname, vopc, dec);

faddr += dec;

}

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

{

char data[10];

fscanf(fp, "%s%\*c", data);

// str\_dec to num

int dec = 0;

for(int i = 0; i < strlen(data); i++)

{

dec \*= 10;

dec += data[i] - '0';

}

fprintf(fw, "%04X %-10s %-10s %d", faddr, vname, vopc, dec);

faddr += 3 \* dec;

}

// special case : BASE

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

{

char data[10];

fscanf(fp, "%s%\*c", data);

fprintf(fw, "%20s %-10s %s", "", vopc, data);

base = symbol\_address(data);

}

// special case : RSUB (no operands behind)

else if(strcmp(vopc, "RSUB") == 0)

{

fscanf(fp, "%\*c");

strcpy(obli, "4F0000");

fprintf(fw, "%04X %6s %-8s %-8s %16s %s", faddr, "", vname, vopc, "", obli);

insert\_asm\_line(faddr, obli);

faddr += 3;

}

// special case : END (finish program)

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

{

asmen = faddr;

char data[10];

fscanf(fp, "%s%\*[^\n]", data);

fprintf(fw, "%10s%-10s %-10s\n", "", vopc, data);

fclose(fp);

break;

}

// operand

else

{

// check opcode start with '+'

int type = 0;

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

{

type = 4;

strcpy(vopc, vopc + 1);

}

// check type of opcode

int hashnum = word\_to\_hash(vopc);

optr now = opcode[hashnum];

while(1)

{

// name found, check type

if(strcmp(now->name, vopc) == 0)

{

if(type == 0)

type = now->data[0] - '0';

break;

}

now = now->next;

}

fprintf(fw, "%04X %-10s ", faddr, vname);

// format 1

if(type == 1)

{

fscanf(fp, "%\*c");

fprintf(fw, "%-10s ", now->name);

sprintf(obli, "%02X", now->num);

fprintf(fw, "%14s %s", "", obli);

insert\_asm\_line(faddr, obli);

}

// format 2

else if(type == 2)

{

char r1[5], r2[5] = {0, }, r3[10];

fscanf(fp, "%s", r1);

strcpy(r3, r1);

// case with , : two registers

if(r1[strlen(r1)-1] == ',')

{

r1[strlen(r1)-1] = '\0';

fscanf(fp, "%s", r2);

strcat(r3, " ");

strcat(r3, r2);

}

fscanf(fp, "%\*c");

sprintf(obli, "%02X%1x%1X", now->num, reg(r1), reg(r2));

fprintf(fw, "%-10s %-14s %s", now->name, r3, obli);

insert\_asm\_line(faddr, obli);

}

// format 3 & 4

else

{

char voper[10];

fscanf(fp, "%s", voper);

if(type == 3)

fprintf(fw, "%-10s ", now->name);

else

fprintf(fw, "+%-9s ", now->name);

int obco = now->num / 4 \* 4; // make object code

int xbpe = 0;

// check x

char x;

fscanf(fp, "%c", &x);

if(x == '\n')

x = 0;

else

{

fscanf(fp, "%\*[^\n]%\*c");

voper[strlen(voper) - 1] = '\0';

x = 1;

}

xbpe += 8 \* x;

int ta; // target address

int ifnum = 0;

// immediate addressing(get address) ni = 01

if(voper[0] == '#')

{

obco += 1;

ta = find\_address(voper + 1, &ifnum);

}

// indirect addressing(get and go to data) ni = 10

else if(voper[0] == '@')

{

obco += 2;

ta = find\_address(voper + 1, &ifnum);

}

// normal addressing ni = 11

else

{

obco += 3;

ta = find\_address(voper, &ifnum);

}

int pc = faddr + 3;

// type 4 : bp = 00, e = 1

if(type == 4)

{

xbpe += 1;

// insert relocation for object file, not immediate(i=1)

if(obco % 4 == 3)

insert\_relocate(faddr + 1, 5);

}

// case with number bp = 00

if(ifnum);

// using program counter

else if(pc - 2048 <= ta && ta < pc)

{

ta -= pc;

ta += 4096;

xbpe += 2;

}

else if(pc <= ta && ta < pc + 2048)

{

ta -= pc;

xbpe += 2;

}

// using base address

else if(base <= ta && ta < base + 4096)

{

ta -= base;

xbpe += 4;

}

// range error at format 3

else if(type == 3 && ta >= 4096)

{

printf("at line %d : out of range\n", i);

return -1;

}

// range error at format 4

else if(type == 4 && ta >= 1<<20)

{

printf("at line %d : out of range\n", i);

return -1;

}

if(x)

strcat(voper, ", X");

if(type == 4) sprintf(obli, "%02X%1X%05X", obco, xbpe, ta);

else sprintf(obli, "%02X%1X%03X", obco, xbpe, ta);

fprintf(fw, "%-14s %s", voper, obli);

insert\_asm\_line(faddr, obli);

}

faddr += type;

}

fprintf(fw, "\n");

}

}

if(base == -4096) // base undeclared : send warning message

printf("WARNING : BASE undeclared\n");

fclose(fw);

return 0;

}

// malloc head pointer of asmline linked list

void set\_asm\_line()

{

lhead = (lptr)malloc(sizeof(asmline));

lhead->addr = 0;

strcpy(lhead->obco, "");

lhead->next = NULL;

lnow = lhead;

return;

}

// free asmline except for head

void reset\_asm\_line()

{

lnow = lhead->next;

while(lnow != NULL)

{

lptr lnext = lnow->next;

free(lnow);

lnow = lnext;

}

lhead->next = NULL;

lnow = lhead;

return;

}

// insert address & object code in asmline

void insert\_asm\_line(int addr, char\* obco)

{

if(strcmp(obco, "") == 0)

return;

lnow->next = (lptr)malloc(sizeof(asmline));

lnow = lnow->next;

lnow->addr = addr;

strcpy(lnow->obco, obco);

lnow->next = NULL;

return;

}

// free all asm line linked list

void free\_asm\_line()

{

reset\_asm\_line();

free(lhead);

return;

}

// get symbol name and find adress

int symbol\_address(char\* name)

{

if(name[0] - 'A' < 0 || name[0] - 'A' > 25)

return 0;

sptr now = top + name[0] - 'A';

while(now != NULL)

{

if(strcmp(now->name, name) == 0)

return now->addr;

now = now->next;

}

return 0;

}

// retur number of register

int reg(char\* name)

{

if (strcmp(name, "A") == 0) return 0;

else if (strcmp(name, "X") == 0) return 1;

else if (strcmp(name, "L") == 0) return 2;

else if (strcmp(name, "PC") == 0) return 8;

else if (strcmp(name, "SW") == 0) return 9;

else if (strcmp(name, "B") == 0) return 3;

else if (strcmp(name, "S") == 0) return 4;

else if (strcmp(name, "T") == 0) return 5;

else if (strcmp(name, "F") == 0) return 6;

return 0;

}

// get name(or number) and retur irs address

int find\_address(char\* name, int\* ifnum)

{

int ret = 0;

// symbol case

if(ret = symbol\_address(name))

return ret;

// other case : number

else

{

for(int i = 0; i < strlen(name); i++)

{

ret \*= 10;

ret += name[i] - '0';

}

\*ifnum = 1;

return ret;

}

}

/\* assemble part 3 : object file \*/

// make object file

int make\_obj(char\* filename)

{

// make name of list file

char objname[10];

strcpy(objname, filename);

strcpy(objname + strlen(objname) - 3, "obj");

FILE\* fw = fopen(objname, "w");

// header record

fprintf(fw, "H%-6s%06X%06X\n", asmname, asmst, asmen);

// text record

print\_obli(fw);

// relocation record

print\_relo(fw);

free\_relocate();

set\_relocate();

// end record

fprintf(fw, "E%06X\n", asmst);

fclose(fw);

return 0;

}

void print\_obli(FILE\* fw)

{

int line\_buf = 30;

lptr lread = lhead->next;

if(lread == NULL)

return;

int saddr = 0, faddr = 0;

while(lread != NULL)

{

char fr[10] = {0, };

sprintf(fr,"T%06X", lread->addr);

char bc[100] = {0, };

saddr = lread->addr;

faddr = saddr - 1;

while(1)

{

faddr += strlen(lread->obco) / 2;

// last of object code

if(lread->next == NULL)

{

sprintf(fr + 7, "%02X", faddr - saddr + 1);

strcat(bc, lread->obco);

lread = lread->next;

break;

}

// variable comes next

else if(faddr + 1 != lread->next->addr)

{

sprintf(fr + 7, "%02X", faddr - saddr + 1);

strcat(bc, lread->obco);

lread = lread->next;

break;

}

// reached buffer

else if(faddr == saddr + line\_buf)

{

sprintf(fr + 7, "%02X", faddr - saddr + 1);

strcat(bc, lread->obco);

lread = lread->next;

break;

}

else if(faddr > saddr + line\_buf)

{

faddr -= strlen(lread->obco) / 2;

sprintf(fr + 7, "%02X", faddr - saddr + 1);

break;

}

strcat(bc, lread->obco);

lread = lread->next;

}

fprintf(fw, "%s%s\n", fr, bc);

}

return;

}

void print\_relo(FILE\* fw)

{

rptr temp = rhead->next;

while(temp != NULL)

{

fprintf(fw, "M%06X%02X\n", temp->addr, temp->leng);

temp = temp->next;

}

return;

}

void set\_relocate()

{

rhead = (rptr)malloc(sizeof(relocate));

rhead->addr = -1;

rhead->leng = 0;

rhead->next = NULL;

rtail = rhead;

return;

}

void insert\_relocate(int addr, int leng)

{

rtail->next = (rptr)malloc(sizeof(relocate));

rtail = rtail->next;

rtail->addr = addr;

rtail->leng = leng;

rtail->next = NULL;

return;

}

void free\_relocate()

{

rptr now = rhead;

while(now != NULL)

{

rptr new = now->next;

free(now);

now = new;

}

return;

}

/\* type(12) : print contents of the file \*/

int function\_type()

{

// check additional word -> if not, error.

if(rest\_exist == 0)

{

printf("input valid order\n");

return -1;

}

char filename[80];

scanf("%s", filename);

scanf("%\*[^\n]");

scanf("%\*c");

// open file

FILE\* fp = fopen(filename, "r");

if(fp == NULL)

{

printf("cannot find file\n");

return -1;

}

// check if directory

struct stat buf;

lstat(filename, &buf);

if(S\_ISDIR(buf.st\_mode))

{

printf("given name is directory\n");

fcose(fp);

return -1;

}

insert\_history(12, filename);

while(1)

{

char c;

if(fscanf(fp, "%c", &c) == EOF)

break;

printf("%c", c);

}

fclose(fp);

return 0;

}

/\* symbol(13) : print symbol table of last assembled file \*/

int function\_symbol()

{

if(rest\_exist == 1)

{

dump\_rest();

return -1;

}

int ch = 0;

for(int i = 0; i < 26; i++)

{

sptr now = (top+i)->next;

while(now != NULL)

{

ch = 1;

printf("%-8s%-8s%04X\n", "\0", now->name, now->addr);

now = now->next;

}

}

if(ch == 0)

{

printf("no assemble yet or last assemble was error!\n");

return -1;

}

insert\_history(13, "");

return 0;

}

// make top of symbol table(linked list) at start of program

void set\_symbol\_top()

{

top = (sptr)malloc(26\*sizeof(smbtable));

for(int i = 0; i < 26; i++)

{

strcpy((top+i)->name, "");

(top+i)->addr = -1;

(top+i)->next = NULL;

}

return;

}

// free top of symbol table(linked list) at end of program

void free\_symbol\_top()

{

free\_symbol\_table();

free(top);

return;

}

1. 원활한 설명을 위해 모듈 정의 단계와 전역 변수 정의 단계를 합쳤습니다. [↑](#footnote-ref-1)