**과목명: 시스템프로그래밍**

**2 분반**

**<<Project #3>>**

**서강대학교 [학부명] 컴퓨터공학과**

**[학번] 20171640**

**[이름] 박수진**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 흐름도
   2. 알고리즘
3. **모듈 정의**

3.1 모듈 이름: progaddr(char\*)

3.1.1 기능

3.1.2 사용 변수

3.2 모듈 이름: loader(char\*)

3.2.1 기능

3.2.2 사용 변수

3.3 모듈이름: loader\_pass\_1(int, char(\*)[50])

3.3.1 기능

3.3.2 사용변수

3.4 모듈이름: loader\_pass\_2(int, char(\*)[50])

3.4.1 기능

3.4.2 사용변수

3.5 모듈이름: bp(char\*)

3.5.1 기능

3.5.2 사용변수

3.6 모듈 이름 : run()

3.6.1 기능

3.6.2 사용 변수

3.7 모듈 이름: target\_addr(int)

3.7.1 기능

3.7.2 사용 변수

3.8 모듈이름: target\_val(int, int\*)

3.8.1 기능

3.8.2 사용변수

3.9 모듈이름: store\_inst(int, int)

3.8.1 기능

3.8.2 사용변수

1. **구조체 정의**

4.1 struct estab

4.1.1 기능

4.1.2 사용 변수

4.2 struct bplist

4.2.1 기능

4.2.2 사용 변수

1. **전역 변수 정의**

5.1 estab \*eshead

5.2 bplist \*bphead

5.3 int startaddr

5.4 int tot\_length

5.4 int register\_set[10]

1. **코드 설명**

6.1 20171640.h

6.2 20171640.c

# 프로그램 개요

프로젝트 1, 2에서 구현한 shell을 바탕으로 loader 기능을 추가한다. object file을 받아 linking 및 loading 작업을 하고 가상 메모리에 load된 프로그램을 run으로 실행시킨다. loading 작업이 끝난 후에는 load map을 출력하여 ESTAB을 확인할 수 있도록 한다. 디버깅이 용이하도록 bp(break point) 기능을 추가하여 프로그램이 run하는 동안 break point 마다 정지하여 register에 저장된 값을 확인할 수 있도록 한다.

# 프로그램 설명

## 프로그램 흐름도

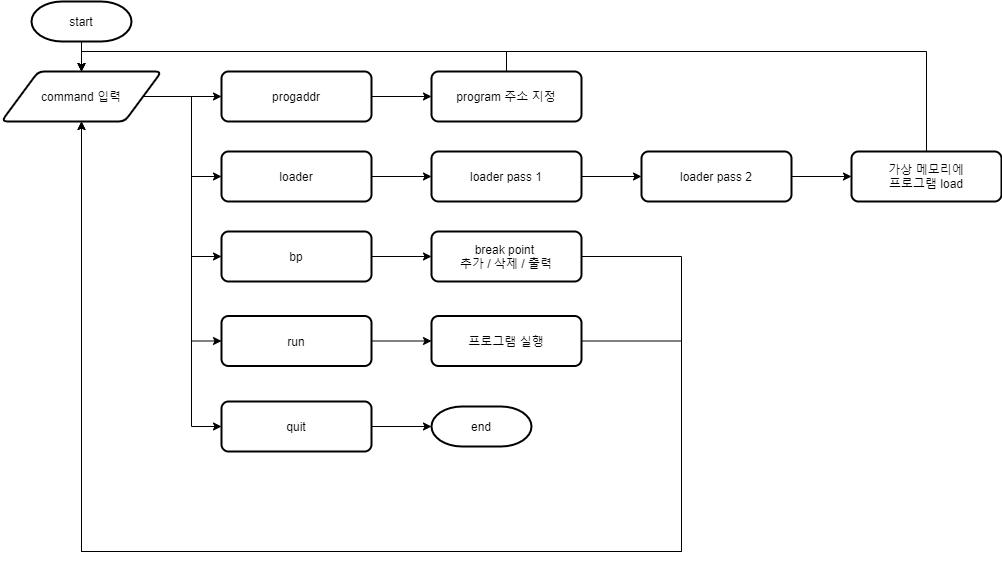


그림 1> 전체 프로그램 흐름도

## 알고리즘 - loader loader 명령어가 입력되면 인자로 들어온 파일의 개수를 확인하고 개수가 정해진 범위를 벗어나는 경우에는 에러 메세지를 출력하고 함수를 종료한다. loader\_pass\_1()을 호출하여 linking & loading을 시작한다. 만약 pass1 또는 pass2 과정에서 에러가 발생하면 에러 메시지를 출력 후 종료한다. load 작업이 정상적으로 종료되면 load map(contents of ESTAB)을 출력하고 레지스터를 초기화한다. i) loader pass 1 pass 1의 주요 목적은 ESTAB을 작성하는 것이다. Pass 1은 다음 pseudo code를 참고하여 작성하였다. pass1에서는 ESTAB 작성이 목표이므로 H record와 D record만 읽으면 된다. (1) 프로그램 시작 주소를 progaddr 명령어로 받은 주소로 설정한다 (default 0) (2) H record에서 program name, program address, program length를 받아온다. (3) 현재까지 저장된 ESTAB을 순회하며 program name과 중복된 symbol이 있는지 확인한다. (3-1) 중복된 symbol이 있으면 에러를 출력하고 load 작업을 중단한다. (3-2) 중복된 symbol이 없으면 program name, address, length를 ESTAB에 저장한다. (4) D record에서 EXTDEF symbol들을 읽어온다. (5) 현재까지 저장된 ESTAB을 순회하며 중복된 symbol이 있는지 확인한다. (5-1) 중복된 symbol이 있으면 에러를 출력하고 load 작업을 중단한다. (5-2) 중복된 symbol이 없으면 symbol과 address를 ESTAB에 저장한다. (6) E record가 나오면 pass1을 종료한 후 pass2를 호출한다. ii) loader pass 2 pass 2가 성공적으로 끝나면 가상 메모리에 프로그램이 로드되어 있어야 한다. Pass2는 다음과 같은 pseudo code를 참고하여 작성하였다. (1) R record에서 EXTREF symbol들과 reference number를 읽어온다. (1-1) ESTAB에 symbol이 존재하지 않으면 에러 출력 후 load를 중단한다. (1-2) ESTAB에 symbol이 존재하면 해당 reference number에 symbol의 address를 저장한다. (2) T record에서 address와 object code를 읽어와서 object code를 memory에 실제로 올린다. (3) M record에서 modification이 필요한 address와 modification 해야 할 자리 수, 그리고 symbol name을 읽어온다. (4) 수정해야할 메모리에 저장된 값과 더해줄(빼줄) symbol의 address 값을 불러온다. (5) 수정해야할 자리 수만큼 symbol의 address를 더해준다(빼준다). (6) E record가 나오면 pass2를 종료한다.

- run

i) target address 계산  
(1) e = 1인 경우 (4형식) 절대 주소이므로 object code에서 address를 읽어 target address로 지정한다.  
(2) p = 1인 경우 (PC relative) opcode에서 displacement를 읽어와서 PC register 값에 더해준다.  
(2-1) 단, displacement가 음수일 경우 아주 큰 양수로 인식할 수 있기 때문에 수정 작업을 거친다.  
(3) b = 1인 경우 (Base relative) opcode에서 displacement를 읽어와서 B register 값에 더해준다. (Base relative는 displacement가 항상 양수이다.)  
(4) x = 1인 경우 (Indexed addressing) target address에 X register 값을 더해 target address를 수정한다.

ii) target address의 value 구하기  
(1) n = 1, i = 1인 경우 (simple addressing) target address에 저장된 값을 가져온다.  
(2) n = 1, i = 0인 경우 (indirect addressing) target address에 저장된 값을 다시 target address로 지정한 후 그 주소에 저장된 값을 가져온다.  
(3) n = 0, i = 1인 경우 (immediate addressing) target address를 그대로 사용한다.

iii) 명령어 실행하기  
해당하는 명령어를 실행하여 register 값을 변경한다.

# 모듈 정의

## 모듈 이름 : progaddr(char\* addr)

### 기능 프로그램의 시작주소를 입력 받아 저장한다.

### 사용 변수

* int tmp1 : string형 progaddr 명령어에서 address만 정수형으로 추출할 변수
* char tmp2[10] : address 이외에 다른 파라미터가 있는지 확인할 변수
* char \*addr : progaddr 뒤에 파라미터로 입력된 스트링 (매개변수)

## 모듈 이름: loader(char \*files)

### 기능

파일을 입력 받아 loader\_pass\_1을 호출하고 성공적으로 loading을 끝내면 load map(ESTAB)을 출력한다.

### 사용 변수

* char fileset[3][50] : 입력된 file명을 최대 3개까지 저장
* char tmp[50] : 파일 개수가 3개를 초과하는지 확인하기 위한 임시 변수
* int filenum : load할 파일의 개수를 저장
* int totlen : load된 프로그램의 총 길이
* estab \*cur : ESTAB을 출력하기 위해 linked list를 순회하는 포인터
* char \*files : loader 뒤에 파라미터로 입력된 스트링 (매개변수)

## 모듈이름: loader\_pass\_1(int filenum, char (\*fileset)[50])

### 기능 loader의 pass1을 수행하여 pass2 작업을 위한 ESTAB을 생성한다. 중복된 symbol이 있으면 에러 처리하고 없으면 새 노드를 ESTAB linked list에 추가한다. pass1 작업이 정상적으로 종료되면 pass2를 호출한다.

### 사용변수

* int i, j : 반복문이나 배열의 인덱스에 사용되는 변수
* int len : control section 하나의 길이
* int csaddr : control section의 시작 주소
* int symaddr : symbol의 주소
* FILE \*fp : object file에 사용할 file pointer
* char line[270] : object file에 저장된 한 줄을 읽는 string
* char symname[10] : symbol의 이름을 저장할 string
* estab \*cur : estab에 중복되는 symbol이 있는지 확인할 pointer
* int filenum : load하는 파일의 개수 (매개변수)
* char (\*fileset)[50] : load에 사용되는 파일들의 이름이 저장된 배열 (매개변수)

## 모듈이름: loader\_pass\_2(int filenum, char (\*fileset)[50])

### 기능 object file에 저장된 object code들을 가상 메모리에 올리는 작업을 한다. R record에서 중복되는 symbol이 있는지 확인 후 없으면 에러 처리하고 있으면 해당 reference number에 symbol address 저장한다. 이 symbol들은 modification에 사용되어 수정할 object code들에 더하거나 빼진다.

### 사용변수

* int i, j : 반복문이나 배열의 인덱스에 사용되는 변수
* int csaddr : control section의 시작 주소
* int execaddr : object code가 올라가는 가상 메모리 주소
* int cslth : control section의 길이
* int refnum : symbol의 reference number
* int rcdlen : T record 한 줄의 길이
* int rcdaddr : T record 또는 M record에서 표기되어 있는 주소
* int symaddr : symbol의 주소
* int objcode : object file에 저장된 object code
* int mod : modification이 필요한 자리 수
* int tmp : modification을 할 때 메모리에 저장되어 있는 값을 읽어온 변수
* FILE \*fp : object file에 사용되는 file pointer
* char operator : M record에서 사용된 operator를 저장하는 변수
* char line[270] : object file에 저장된 한 줄을 읽는 string
* char symname[10] : symbol의 이름이 저장된 string
* char argument[10] : 메모리에 값을 저장할 때 edit 함수의 매개변수로 사용
* estab \*cur : estab을 순회하며 중복된 symbol이 있는지 확인할 pointer
* int refarr[10] : reference number에 해당하는 address를 저장한 배열
* int filenum : load하는 파일의 개수 (매개변수)
* char (\*fileset)[50] : load에 사용되는 파일들의 이름이 저장된 배열 (매개변수)

## 모듈이름: bp(char \*bpcmd)

### 기능 bp 관련 명령들을 수행한다. bp 입력 시 break point 출력, bp [address]을 입력 시 break point 추가, bp clear입력 시 break point를 모두 제거한다.

### 사용변수

* int addr : bp 명령어 parameter로 입력된 break point를 생성할 address
* char tmp[10] : bp 뒤에 허용되지 않은 파라미터가 추가로 있는지 확인할 변수
* bplist \*cur : bplist를 순회하는 pointer
* bplist \*rm : bplist에 저장된 node를 삭제하는데 사용하는 pointer
* char \*bpcmd : bp 명령어 뒤에 파라미터로 입력된 스트링 (매개변수)

## 모듈이름: run()

### 기능 memory에 load된 정보를 바탕으로 실제로 프로그램을 실행하는 함수. mnemonic에 따라 각기 다른 동작이 수행된다. 1, 2, 3, 4 형식에 맞춰PC register를 업데이트한다. 동작이 중단되면 register에 저장된 값들을 출력한다.

### 사용변수

* int op1, op2 : 각각 opcode의 앞자리와 뒷자리를 저장한다.
* int opcode : object code에서 opcode를 저장한다.
* int ni : object code에서 ni를 저장한다.
* int xbpe : object code에서 xbpe를 저장한다.
* int taraddr : target address를 저장한다.
* int tarval : target address에 저장된 값을 저장한다.
* int r1, r2 : 2형식에서 사용되는 register number
* int bpflag : 현재 PC가 break point이면 1로 set한다.
* bplist \*cur : breakpoint를 탐색하기 위한 pointer

## 모듈이름: target\_addr(int xbpe)

### 기능 xbpe값과 PC register 값을 이용하여 target address를 계산하여 반환한다.

### 사용변수

* taraddr : target address를 저장
* disp : 3형식에서 displacement를 저장
* abaddr : 4형식에서 absolute address를 저장
* int xbpe : object code의 xbpe를 저장 (매개변수)

## 모듈이름: target\_val(int ni, int \*taraddr)

### 기능 target address에 저장된 값을 찾아서 반환한다.

### 사용변수

* int val : target address에 저장된 value를 저장
* int ni : object code의 ni를 저장 (매개변수)
* int \*taraddr : target address를 저장, indirect addressing이면 target address가 바뀌므로 포인터로 선언하였다 (매개변수)

## 모듈이름: store\_inst(int regnum, int taraddr)

### 기능 프로그램을 실행할 때 store 명령어들에 대한 action을 수행한다.

### 사용변수

* int regnum : register 번호를 저장 (매개변수)
* int taraddr : 메모리에 저장될 target address (매개변수)

# 구조체 정의

### **4.1 struct estab**

### 기능 ESTAB(external symbol table)을 저장하는 linked list에 사용되는 구조체

### 사용변수

- char symbol[30] : control section or symbol name

- int address : address of control section or symbol

- int length : length of control section

- struct estab \*link : next estab node

### **4.2 struct bplist**

### 4.2.1 기능 bp list를 저장하는 linked list에 사용되는 구조체

### 4.2.2 사용변수

- bp : 입력 받은 break point

- struct bplist \*link : next bplist node

# 전역 변수 정의

## estab \*eshead estab linked list의 head pointer

## bplist \*bphead

bplist linked list의 head pointer

## int startaddr

memory에 load 되는 시작 주소

## int tot\_length

load된 프로그램의 최종 길이

## int register\_set[10]

각 register number index마다 register의 값을 저장

# 코드 설명

## 20171640.h

/\* 포함되는 파일 및 라이브러리\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <dirent.h>

/\* 정의되는 구조체\*/

typedef struct hiList { // history linked list 저장할 구조체

char cmd\_name[100]; // 커맨드 저장

struct hiList \*link; // 다음 노드를 가리키는 link

}hiList;

typedef struct OPlist{ // opcode linked list 저장할 구조체

int opcode; // opcode 저장

char inst\_name[10]; // mnemonic 저장

char form[5]; // format 저장

struct OPlist \*link; // 다음 노드를 가리키는 link

} OPlist;

typedef struct symbol{ // label 저장할 linkedlist용 구조체

char name[100]; // label명

int locctr; // location counter

struct symbol \*link; // 다음 노드 link

}symbol;

typedef struct objlist{ // object code를 저장한 linkedlist 용 구조체

char objcode[100]; // objcode

int loc; // location

int flag; // 줄바꿈 필요한지 flag

int mflag; // modification 필요한 경우 flag

struct objlist \*link; // 다음 노드 가리키는 link

}objlist;

typedef struct estab{ // externeal symbol table을 저장한 linked list용 구조체

char symbol[30]; // control section or symbol name

int address; // start address

int length; // length of control section

struct estab \*link; // next node를 가리킴

}estab;

typedef struct bplist{ // bp list를 저장할 linked list용 구조체

int bp; // 입력 받은 break point

struct bplist \*link; // next node

}bplist;

/\* global 변수 \*/

hiList \*hiHead = NULL; // hiList head

OPlist \*OPhead[20]; // OPlist head 배열

symbol \*symhead[26]; // symlist head 배열

objlist \*objhead = NULL; // objlist head

estab \*eshead = NULL; // estab head

bplist \*bphead = NULL; // bplist head

unsigned char memory[1048576]; // 16 X 65536 가상메모리

int dumpend = -1; // dump에서 마지막으로 저장된 end 값

int adr\_start = 0; // 프로그램 시작주소

int pro\_length = 0; // 프로그램 길이

int symflag = 0; // symbol table flag

int startaddr = 0; // memory에 load 되는 시작 주소

int tot\_length = -1; // load된 프로그램의 최종 길이

int register\_set[10]; // 각 register의 value를 저장

/\* 함수 원형 \*/

/\*---------------------proj 1----------------------------\*/

/\*-----------------------------------------\*/

/\* 함수 : editcmd \*/

/\* 목적 : 입력받은 커맨드를 명령어와 \*/

/\* 나머지로 나눈다 \*/

/\* 리턴값 : 없음 \*/

/\*-----------------------------------------\*/

void editcmd(char\*, char\*, char\*);

/\*-----------------------------------------\*/

/\* 함수 : help \*/

/\* 목적 : 프로그램 상에서 사용할 수 있는 \*/

/\* 명령어들을 출력한다 \*/

/\* 리턴값 : 없음 \*/

/\*-----------------------------------------\*/

void help();

/\*-----------------------------------------\*/

/\* 함수 : dir \*/

/\* 목적 : 현재 디렉토리 내 파일들을 \*/

/\* 출력한다 \*/

/\* 리턴값 : 없음 \*/

/\*-----------------------------------------\*/

void dir();

/\*-----------------------------------------\*/

/\* 함수 : history \*/

/\* 목적 : history linked list에 저장된 \*/

/\* 정보를 차례로 출력 \*/

/\* 리턴값 : 없음 \*/

/\*-----------------------------------------\*/

void history();

/\*-----------------------------------------\*/

/\* 함수 : add\_list \*/

/\* 목적 : 입력받은 instruction을 history \*/

/\* linked list에 저장 \*/

/\* 리턴값 : 없음 \*/

/\*-----------------------------------------\*/

void add\_list(char\*);

/\*-----------------------------------------\*/

/\* 함수 : dump \*/

/\* 목적 : 가상 메모리에 저장된 정보를 특정 \*/

/\* 범위 만큼 출력 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

/\*-----------------------------------------\*/

int dump(char\*);

/\*-----------------------------------------\*/

/\* 함수 : edit \*/

/\* 목적 : 가상 메모리의 특정 주소에 저장된 \*/

/\* 값을 치환한다 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

/\*-----------------------------------------\*/

int edit(char\*);

/\*-----------------------------------------\*/

/\* 함수 : fill \*/

/\* 목적 : 가상 메모리에서 원하는 범위만큼의\*/

/\* 값을 특정 값으로 치환한다 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 \*/

/\*-----------------------------------------\*/

int fill(char\*);

/\*-----------------------------------------\*/

/\* 함수 : reset \*/

/\* 목적 : 가상 메모리의 모든 값을 0으로 \*/

/\* 초기화 \*/

/\* 리턴값 : 없음 \*/

/\*-----------------------------------------\*/

void reset();

/\*-----------------------------------------\*/

/\* 함수 : makeOPlist \*/

/\* 목적 : opcode 파일로부터 hash table 생성\*/

/\* 리턴값 : 없음 \*/

/\*-----------------------------------------\*/

void makeOPlist();

/\*-----------------------------------------\*/

/\* 함수 : hashFunction \*/

/\* 목적 : 입력받은 string을 암호화 \*/

/\* 리턴값 : 암호화된 int \*/

/\*-----------------------------------------\*/

int hashFunction(char\*);

/\*-----------------------------------------\*/

/\* 함수 : opcode \*/

/\* 목적 : 입력받은 mnemonic에 해당하는 \*/

/\* opcode 출력 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

/\*-----------------------------------------\*/

int opcode(char\*);

/\*-----------------------------------------\*/

/\* 함수 : opcodelist \*/

/\* 목적 : hashtable에 저장된 opcode list를 \*/

/\* 차례로 출력 \*/

/\* 리턴값 : 없음 \*/

/\*-----------------------------------------\*/

void opcodelist();

/\*---------------------proj 2----------------------------\*/

/\*-----------------------------------------\*/

/\* 함수 : getfilename \*/

/\* 목적 : 문자열의 제일 뒤에 있는 공백들을 \*/

/\* 제거 \*/

/\* 리턴값 : 없음 \*/

/\*-----------------------------------------\*/

void getfilename(char\*);

/\*-----------------------------------------\*/

/\* 함수 : type \*/

/\* 목적 : 파일 안의 내용을 그대로 출력 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

/\*-----------------------------------------\*/

int type(char\*);

/\*-----------------------------------------\*/

/\* 함수 : assemble \*/

/\* 목적 : 입력받은 .asm 파일을 어셈블 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

/\*-----------------------------------------\*/

int assemble(char\*);

/\*-----------------------------------------\*/

/\* 함수 : asm\_pass\_1 \*/

/\* 목적 : location counter 계산하여 \*/

/\* intermediate 파일에 저장 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

/\*-----------------------------------------\*/

int asm\_pass\_1(char\*);

/\*-----------------------------------------\*/

/\* 함수 : asm\_pass\_2 \*/

/\* 목적 : intermediate 파일에서 어셈블하여 \*/

/\* obj, lst 파일 생성 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

/\*-----------------------------------------\*/

int asm\_pass\_2(char\*);

/\*-----------------------------------------\*/

/\* 함수 : ZtoAhash \*/

/\* 목적 : 알파벳 내림차순으로 저장하기 위한\*/

/\* hash function \*/

/\* 리턴값 : index number \*/

/\*-----------------------------------------\*/

int ZtoAhash(char\*);

/\*-----------------------------------------\*/

/\* 함수 : symbolcmd \*/

/\* 목적 : symbol table 내용 출력 \*/

/\* 리턴값 : 에러발생 시 0, 정상종료 시 1 \*/

/\*-----------------------------------------\*/

int symbolcmd();

/\*-----------------------------------------\*/

/\* 함수 : add\_objlist \*/

/\* 목적 : object code를 linked list에 저장 \*/

/\* 리턴값 : 없음 \*/

/\*-----------------------------------------\*/

void add\_objlist(char\*, int, int);

/\*---------------------proj 3----------------------------\*/

/\*-----------------------------------------\*/

/\* 함수 : progaddr \*/

/\* 목적 : 프로그램 시작 주소를 지정 \*/

/\* 리턴값 : 에러 발생시 0, 정상 종료 시 1 \*/

/\*-----------------------------------------\*/

int progaddr(char\*);

/\*-----------------------------------------\*/

/\* 함수 : loader \*/

/\* 목적 : 프로그램을 load하기 위해 pass1과 \*/

/\* pass2를 호출하고 load map 출력 \*/

/\* 리턴값 : 에러 발생시 0, 정상 종료 시 1 \*/

/\*-----------------------------------------\*/

int loader(char\*);

/\*-----------------------------------------\*/

/\* 함수 : loader\_pass\_1 \*/

/\* 목적 : loader pass 1 과정을 진행한다 \*/

/\* ESTAB을 생성한다 \*/

/\* 리턴값 : 에러 발생시 0, 정상 종료 시 1 \*/

/\*-----------------------------------------\*/

int loader\_pass\_1(int, char(\*)[50]);

/\*-----------------------------------------\*/

/\* 함수 : loader\_pass\_2 \*/

/\* 목적 : loader pass 2 과정을 진행한다 \*/

/\* 메모리에 실제로 값을 올린다 \*/

/\* 리턴값 : 에러 발생시 0, 정상 종료 시 1 \*/

/\*-----------------------------------------\*/

int loader\_pass\_2(int, char(\*)[50]);

/\*-----------------------------------------\*/

/\* 함수 : bp \*/

/\* 목적 : break point 관련 명령어들을 수행 \*/

/\* 리턴값 : 에러 발생시 0, 정상 종료 시 1 \*/

/\*-----------------------------------------\*/

int bp(char\*);

/\*-----------------------------------------\*/

/\* 함수 : run \*/

/\* 목적 : 프로그램을 실제로 실행한다 \*/

/\* register에 저장된 값을 출력 \*/

/\* 리턴값 : 에러 발생시 0, 정상 종료 시 1 \*/

/\*-----------------------------------------\*/

int run();

/\*-----------------------------------------\*/

/\* 함수 : target\_addr \*/

/\* 목적 : 메모리에서 object code를 가져와 \*/

/\* target address 계산 \*/

/\* 리턴값 : target address \*/

/\*-----------------------------------------\*/

int target\_addr(int);

/\*-----------------------------------------\*/

/\* 함수 : target\_val \*/

/\* 목적 : target address에 저장된 값 반환 \*/

/\* 리턴값 : target address의 value \*/

/\*-----------------------------------------\*/

int target\_val(int, int\*);

/\*-----------------------------------------\*/

/\* 함수 : store\_inst \*/

/\* 목적 : store 명령어들을 수행 \*/

/\* 리턴값 : 없음 \*/

/\*-----------------------------------------\*/

void store\_inst(int, int);

## 20171640.c

/\* 포함되는 파일 \*/

#include "20171640.h"

/\* 프로그램 시작\*/

int main() {

// oricmd : 처음 입력받은 커맨드

// cmd : 명령어 부분만 잘라낸 char 배열

// idx : 명령어 뒷부분만 잘라낸 char 배열

char oricmd[100], cmd[100], idx[100];

makeOPlist(); // opcode list로 hash table 생성

bphead = (bplist\*)malloc(sizeof(bplist)); // bplist head 할당

bphead->bp = -1;

bphead->link = NULL;

while (1) {

printf("sicsim> ");

// local 변수 초기화

memset(oricmd, '\0', sizeof(oricmd));

memset(cmd, '\0', sizeof(cmd));

memset(idx, '\0', sizeof(idx));

// 커맨드 입력받아 저장

fgets(oricmd, sizeof(oricmd), stdin);

// cmd와 idx로 분리

editcmd(oricmd, cmd, idx);

// 명령어가 "h" 또는 "help" 일 때

if (!strcmp(cmd, "help") || !strcmp(cmd, "h")) {

if (idx[0] == '\0'){ // idx가 비었으면

add\_list(oricmd); // history에 추가

help(); // help 실행

}

else // idx가 비어있지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "d" 또는 "dir" 일 때

else if (!strcmp(cmd, "d") || !strcmp(cmd, "dir")) {

if (idx[0] == '\0'){ // idx가 비었으면

add\_list(oricmd); // history에 추가

dir(); // dir 실행

}

else // idx가 비어있지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "q" 또는 "quit" 일 때

else if (!strcmp(cmd, "q") || !strcmp(cmd, "quit")) {

if (idx[0] == '\0') // idx가 비었으면

break; // 루프문 종료

else // 비어있지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "hi" 또는 "history" 일 때

else if (!strcmp(cmd, "hi") || !strcmp(cmd, "history")) {

if (idx[0] == '\0'){ // idx가 비었으면

add\_list(oricmd); // history에 추가

history(); // history 실행

}

else // 그렇지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "du" 또는 "dump" 일 때

else if (!strcmp(cmd, "du") || !strcmp(cmd, "dump")){

if (dump(idx) == 1) // dump() return 값이 1이면

add\_list(oricmd); // history 추가

}

// 명령어가 "e" 또는 "edit" 일 때

else if (!strcmp(cmd, "e") || !strcmp(cmd, "edit")){

if (edit(idx) == 1) // edit() return 값이 1이면

add\_list(oricmd); // history 추가

}

// 명령어가 "f" 또는 "fill" 일 때

else if (!strcmp(cmd, "f") || !strcmp(cmd, "fill")){

if (fill(idx) == 1) // fill() return 값이 1이면

add\_list(oricmd); // history에 추가

}

// 명령어가 "reset" 일 때

else if (!strcmp(cmd, "reset")){

if (idx[0] == '\0'){ // idx가 비었으면

add\_list(oricmd); // history에 추가

reset(); // reset 실행

}

else // 그렇지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "opcode" 일 때

else if (!strcmp(cmd, "opcode")){

if (opcode(idx) == 1) // opcode() return 값이 1이면

add\_list(oricmd); // history에 추가

}

// 명령어가 "opcodelist" 일 때

else if (!strcmp(cmd, "opcodelist")){

if (idx[0] == '\0'){ // idx가 비었으면

add\_list(oricmd); // history에 추가

opcodelist(); // opcodelist 실행

}

else // 그렇지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "type" 일 때

else if (!strcmp(cmd, "type")){

if (type(idx) == 1) // type() return이 1이면

add\_list(oricmd); // history에 추가

}

// 명령어가 "assemble" 일 때

else if (!strcmp(cmd, "assemble")){

if (assemble(idx) ==1) // assemble() return이 1이면

add\_list(oricmd); // history에 추가

}

// 명령어가 "symbol" 일 때

else if (!strcmp(cmd, "symbol")) {

if (idx[0] == '\0'){ // idx가 비었으면

if (symbolcmd() == 1)

add\_list(oricmd);

}

else // 비어있지 않으면

printf("Invalid Command\n"); // 에러 출력

}

// 명령어가 "progaddr" 일 때

else if (!strcmp(cmd, "progaddr")){

if (progaddr(idx) == 1) // progaddr function이 성공적으로 수행되었으면

add\_list(oricmd); // history에 추가

}

// 명령어가 "loader" 일 때

else if (!strcmp(cmd, "loader")){

if (loader(idx) == 1) // loader function이 성공적으로 수행되었으면

add\_list(oricmd); // history에 추가

}

// 명령어가 "bp" 일 때

else if (!strcmp(cmd, "bp")){

if (bp(idx) == 1) // bp function이 성공적으로 수행되었으면

add\_list(oricmd); // history에 추가

}

// 명령어가 "run" 일 때

else if (!strcmp(cmd, "run")) {

if (idx[0] == '\0'){ // run 뒤에 명령어가 없고

if (run() == 1) // run function이 성공적으로 수행되었으면

add\_list(oricmd); // history에 추가

}

else

printf("Invalid Command\n"); // 아니면 에러 출력

}

// 그 이외의 입력 에러 처리

else

printf("Invalid Command\n");

}

// linked list dellocate를 위한 local 변수

hiList \*hifree, \*tmp1;

OPlist \*opfree, \*tmp2;

int i;

// history linked list dellocate

hifree = hiHead;

while (hifree != NULL){

tmp1 = hifree;

hifree = hifree -> link;

free(tmp1);

}

// Opcodelist linked list dellocate

for (i = 0 ; i < 20 ; i++){

opfree = OPhead[i];

while (opfree != NULL){

tmp2 = opfree;

opfree = opfree->link;

free(tmp2);

}

}

// free symbol table

symbol \*symptr, \*symrmv;

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

symptr = symhead[i];

while (symptr){

symrmv = symptr;

symptr = symptr -> link;

free(symrmv);

}

symhead[i] = NULL;

}

// free objcode list

objlist \*obptr, \*obrmv;

obptr = objhead;

while (obptr){

obrmv = obptr;

obptr = obptr -> link;

free(obrmv);

}

// free bp list

bplist \*bpptr, \*bprmv;

bpptr = bphead;

while (bpptr){

bprmv = bpptr;

bpptr = bpptr -> link;

free(bprmv);

}

return 0;

}

/\* 사용되는 function \*/

/\* start of proj 1 \*/

void editcmd(char\* cmd, char\* new, char\* idx) {

int i = 0, j = 0; // 반복분과 인덱스에 사용하는 임시 변수

cmd[(unsigned)strlen(cmd) - 1] = '\0'; // 커맨드의 '\n'을 '\0'으로 치환

while (cmd[i] == ' ' || cmd[i] == '\t')

// i에 처음으로 공백이 아닌 값의 인덱스 저장

i++;

while (cmd[i] != ' ' && cmd[i] != '\0' && cmd[i] != '\t'){

// 공백 또는 NULL 값이 나오기 전까지의 값을 new에 저장(명령어 부분만 저장)

new[j] = cmd[i];

i++;

j++;

}

if (cmd[i] != '\0'){ // 명령어 뒤에 파라미터가 더 있을 때

j = 0;

while (cmd[i] == ' ' || cmd[i] == '\t')

// i에 처음으로 공백이 아닌 값의 인덱스를 저장

i++;

while (cmd[i] != '\0'){ // idx에 NULL이 나오기 전까지 저장

idx[j] = cmd[i];

i++;

j++;

}

}

}

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");

}

void dir() {

DIR \*ptrDir;

struct dirent \*ptrFile;

struct stat fileStat;

int num = 0;

ptrDir = opendir("."); // 현재 디렉토리 포인터 불러오기

while ((ptrFile = readdir(ptrDir)) != NULL){ // 디렉토리 내의 파일

stat(ptrFile->d\_name, &fileStat);

printf("\t%s", ptrFile->d\_name); // 파일 읽어서 출력

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

printf("/"); // 파일이 디렉토리이면 '/' 붙이기

else if (S\_IXUSR & fileStat.st\_mode)

printf("\*"); // 파일이 실행파일이면 '\*' 붙이기

num++;

if (num % 4 == 0)

printf("\n"); // 4개 파일 출력되면 줄바꿈

}

if (num % 4 != 0)

printf("\n");

closedir(ptrDir); // 디렉토리 포인터 닫기

}

void history() {

hiList \*ptr; // linked list를 탐색하기 위한 임시변수

int num = 1; // 줄 번호 나타낼 정수

ptr = hiHead;

while (ptr != NULL) {

// ptr이 NULL 하기 전까지 linked list 탐색하며 출력

printf(" %-5d%s\n", num, ptr->cmd\_name);

num++;

ptr = ptr->link;

}

}

void add\_list(char\* cmd) {

hiList \*newNode = (hiList\*)malloc(sizeof(hiList)); // 새로운 값을 저장할 노드

hiList \*ptr = hiHead; // linked list 탐색에 사용되는 임시변수

strcpy(newNode->cmd\_name, cmd);

newNode->link = NULL;

// 새 노드에 입력받은 커맨드 저장, link 값은 아직 없으니 NULL

if (hiHead == NULL) { // head가 NULL 이면

hiHead = (hiList\*)malloc(sizeof(hiList));

hiHead = newNode;

// head 할당한 후 새 노드 저장

}

else { // head가 NULL하지 않으면

while (ptr->link != NULL)

ptr = ptr->link; // 새 노드를 연결할 노드까지 이동

ptr-> link = newNode; // 새 노드 연결

}

}

int dump(char\* idx){

int cmflag = -1; // 콤마의 위치 인덱스

int i = 0, j; // 인덱스와 반복문에 사용할 변수

int start, end; // 메모리 시작 값과 마지막 값을 저장할 변수

char tmp[100];

// 입력값이 올바르지 않음을 판단할 때 사용할 임시 변수

int adflag = 0; // dump 뒤에 인자가 없을 때 1로 set(마지막 address 저장)

while (idx[i] != '\0'){

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

cmflag = i;

break;

}

i++;

} // cmflag : 콤마의 위치

if (cmflag >= 0){ // 콤마가 있을 경우

if (sscanf(idx, "%x \t , \t %x %s", &start, &end, tmp) != 2){

// 파라미터가 두개 존재하지 않는 경우

// 두 파라미터가 16진수가 아닌 경우 에러

printf("Invalid parameter\n");

return 0;

}

}

else { // 콤마가 없을 경우

if (idx[0] == '\0'){ // dump 뒤로 들어온 파라미터가 없으면

start = dumpend + 1;

end = start + 159;

}

else { // dump 뒤로 들어온 파라미터가 있을 때

if (sscanf(idx, "%x %s", &start, tmp) != 1){

// 파라미터가 하나 존재하지 않는 경우

// 파라미터가 16진수가 아닌 경우 에러

printf("Invalid parameter\n");

return 0;

}

end = start + 159;

}

}

if (end > 0xFFFFF) // end 범위가 최대값을 벗어나면 최대값으로 저장

end = 0XFFFFF;

if (start < 0 || end < 0 || start > 0xFFFFF || end > 0xFFFFF || start > end){

// start나 end가 허용된 범위안에 있지 않으면 에러

printf("Address is out of range\n");

return 0;

}

for (i = start / 16 \* 16; i <= end / 16 \* 16; i+= 16){

printf("%05X ", i); // address 출력 (왼쪽 칼럼)

for (j = i; j < i + 16; j++){

// 메모리 값 16진수로 표현 (가운데 칼럼)

if (j >= start && j <= end)

printf("%02X ", memory[j]);

else

printf(" ");

}

printf("; ");

for (j = i; j < i + 16; j++){

// 메모리 값 ASCII 값으로 표현 (오른쪽 칼럼)

if(j >= start && j <= end && memory[j] >= 0X20 && memory[j] <= 0x7E)

// 20에서 7E 까지의 값은 ASCII 값으로 출력한다

putchar(memory[j]);

else

// 그 이외의 값은 '.' 으로 출력한다

putchar('.');

}

printf("\n");

}

if (adflag == 1) // dump 뒤에 인자가 없으면

dumpend = end; // 마지막으로 저장된 end값 global var에 저장

if (dumpend >= 0xFFFFF) // 저장된 global var이 최댓값을 넘어가면 초기화

dumpend = -1;

return 1;

}

int edit(char\* idx){

int cmflag = -1; // 콤마 인덱스 값

int i = 0; // 인덱스나 반복문에 사용할 변수

int adr, val; // 주소값과 메모리값을 저장할 변수

char tmp[100]; // 임시 변수

while (idx[i] != '\0'){

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

cmflag = i;

break;

}

i++;

} // cmflag : 콤마의 위치

if (cmflag == -1){ // 콤마가 없으면 에러

printf("Invalid parameter\n");

return 0;

}

else{ // 콤마가 있으면

if (sscanf(idx, "%x \t , \t %x %s", &adr, &val, tmp) != 2){

// 파라미터 개수가 2개가 아니거나

// 들어온 파라미터가 16진수가 아니면 에러

printf("Invalid parameter\n");

return 0;

}

}

if (adr < 0x00000 || adr > 0xFFFFF){

// 주소값 범위가 허용된 범위를 벗어나면 에러

printf("Address is out of range\n");

return 0;

}

if (val > 0xFF || val < 0){

// value 값이 허용된 범위를 벗어나면 에러

printf("Value is out of range\n");

return 0;

}

memory[adr] = val; // 메모리에 입력된 16진수 값 저장

return 1;

}

int fill(char\* idx){

int cm1 = -1, cm2 = -1; // 콤마 인덱스 값

int i = 0; // 변수, 반복문에 쓰이는 변수

int start, end, value; // 파라미터로 들어온 start, end, value 값

char tmp[100]; // 임시 변수

while (idx[i] != '\0'){ // 첫번째 콤마 탐색

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

cm1 = i;

break;

}

i++;

}

i++;

while (idx[i] != '\0'){ // 두번째 콤마 탐색

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

cm2 = i;

break;

}

i++;

}

if (cm1 == -1 || cm2 == -1){ // 콤마가 2개가 아니면 에러

printf("Invalid parameter\n");

return 0;

}

if (sscanf(idx, "%x \t , \t %x \t , \t %x %s", &start, &end, &value, tmp) != 3){

// 16진수인 파라미터 값이 3개가 아닐경우 에러

printf("Invalid parameter\n");

return 0;

}

if (start < 0 || end < 0 || start > 0xFFFFF || end > 0xFFFFF || start > end){

// start와 end 값이 범위 내에 없을 경우 에러

printf("Address is out of range\n");

return 0;

}

if (value > 0xFF || value < 0){

// value 값이 허용된 범위를 벗어나면 에러

printf("Value is out of range\n");

return 0;

}

for (i = start; i <= end; i++)

memory[i] = value; // 메모리에 입력받은 값 저장

return 1;

}

void reset(){

memset(memory, 0, sizeof(memory)); // 메모리 값 0으로 초기화

}

void makeOPlist(){

FILE \*fp = fopen("opcode.txt", "r"); // 파일 포인터에 읽기전용으로 로드

char inst[10], form[5];

int op, tmp;

// 파일포인터가 비었으면 에러 출력하고 종료

if (fp == NULL){

printf("Opcode list file is NULL\n");

return;

}

while (1){

if (fscanf(fp, "%x %s %s", &op, inst, form) == EOF) // 파일 정보 저장

break; // 더이상 읽을 줄이 없으면 탐색 중단

// 읽3어들인 정보를 새로운 노드에 저장

OPlist \*newNode = (OPlist\*)malloc(sizeof(OPlist));

newNode->opcode = op;

strcpy(newNode->inst\_name, inst);

strcpy(newNode->form, form);

newNode->link = NULL;

// 명령어를 hash function을 이용하여 암호화된 값으로 변환한다.

tmp = hashFunction(inst);

// 새 노드를 linked list에 연결

if (OPhead[tmp] == NULL){ // 해당 인덱스의 헤드가 NULL이면

OPhead[tmp] = newNode; // 제일 첫 노드에 저장

}

else { // 그렇지 않으면

OPlist \*ptr = OPhead[tmp];

while (ptr->link != NULL)

ptr = ptr->link;

ptr->link = newNode; // 가장 뒤에 있는 노드 뒤에 연결

}

}

fclose(fp); // 파일 포인터 닫기

}

int hashFunction(char\* inst){

int i;

int len = strlen(inst);

// 명령어를 읽어서 정수형으로 암호화 한다

i = inst[0] + inst[len - 1] + len;

return i % 20;

}

int opcode(char\* idx){

char tmp[100]; // 임시 변수

char inst[10]; // 오피코드 검색할 명령어

int i = -1; // 임시 변수

OPlist \*ptr; // linked list 탐색할 때 사용하는 변수

// idx에서 명령어 읽어온다

// 파라미터가 1개가 아니면 에러를 출력한다

if (sscanf(idx, "%s %s", inst, tmp) != 1){

printf("Invalid mnemonic\n");

return 0;

}

i = hashFunction(inst); // hash function으로 암호화

ptr = OPhead[i]; // 암호화된 인덱스로 linked list 탐색

// linked list에서 입력된 instruction과 같은 mnemonic을 찾는다

while (ptr != NULL){

if (!strcmp(ptr->inst\_name, inst))

break;

ptr = ptr->link;

}

// 해당하는 mnemonic이 없으면 에러 처리

if (ptr == NULL){

printf("Invalid mnemonic\n");

return 0;

}

// 찾으면 결과 출력

printf("opcode is %02X\n", ptr->opcode);

return 1;

}

void opcodelist(){

int i; // 반복문에 사용할 변수

OPlist \*ptr; // linked list 탐색할 포인터

for (i = 0; i < 20; i++){ // 모든 index 탐색

printf("%d : ", i);

ptr = OPhead[i];

// 한 인덱스에서 연결된 모든 linked list 출력

if (ptr){

while (ptr -> link != NULL){

printf("[%s,%02X] -> ", ptr->inst\_name, ptr->opcode);

ptr = ptr->link;

}

printf("[%s,%02X]", ptr->inst\_name, ptr->opcode);

}

printf("\n");

}

}

/\* end of proj 1 \*/

/\* start of proj 2 \*/

void getfilename(char \*str){

int i = 0, j; // i, j : 배열의 인덱스를 표현

int flag = 0; // 공백을 제거할지 안할지 나타내는 플래그

// 들어온 string의 가장 끝에 의미없는 공백이 있는지 확인

while (str[i] != '\0'){

if (flag == 0) {

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

flag = 1;

j = i;

}

}

else if (flag == 1){

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

flag = 0;

}

i++;

}

// 공백을 제거한다

if (flag == 1){

while (str[j] != '\0'){

str[j] = '\0';

j++;

}

}

}

int type(char \*f\_name){

FILE \*fp;

char tmp;

// 파일 열기

getfilename(f\_name);

fp = fopen(f\_name, "r");

// 파일 없으면 종료

if (!fp) {

printf("No such file in this directory\n");

return 0;

}

// 파일 내용 출력

while (1) {

tmp = fgetc(fp);

if (tmp == EOF)

break;

printf("%c", tmp);

}

//파일 닫기

fclose(fp);

return 1;

}

int assemble(char \*f\_name){

FILE \*fp;

int i = 0;

char name\_tmp[100];

char filetmp[100];

symbol \*ptr, \*rmv;

objlist \*obptr, \*obrmv;

memset(filetmp,'\0',sizeof(filetmp));

// free symbol table

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

ptr = symhead[i];

while (ptr){

rmv = ptr;

ptr = ptr -> link;

free(rmv);

}

symhead[i] = NULL;

}

symflag = 0;

// free objcode list

obptr = objhead;

while (obptr){

obrmv = obptr;

obptr = obptr -> link;

free(obrmv);

}

objhead = NULL;

// 파일 열기

getfilename(f\_name);

fp = fopen(f\_name, "r");

// 파일이 없으면 에러 출력 후 종료

if (!fp) {

printf("No such file in this directory\n");

return 0;

}

// .asm 파일인지 확인

i = 0;

while (1){

if (f\_name[i] == '.') // i는 '.'이 찍힌 인덱스

break;

name\_tmp[i] = f\_name[i]; // '.' 이 나오기 전까지 파일 이름 저장

i++;

}

i++;

if (strcmp(f\_name+i, "asm")) { // . 다음에 asm이 아니면

printf("Invalid file extension\n"); // 에러 메세지 출력 후

return 0; // 종료

}

// 패스1, 패스2 에서 오류 발생했으면 파일 삭제하고 종료

if (asm\_pass\_1(f\_name) == 0){

strcpy(filetmp, name\_tmp);

strcat(filetmp, ".lst");

remove(filetmp);

memset(filetmp, '\0', sizeof(filetmp));

strcpy(filetmp, name\_tmp);

strcat(filetmp, ".obj");

remove(filetmp);

remove("intermediate");

return 0;

}

remove("intermediate");

// 어셈블 정상작동 되면 생성된 파일명 출력

printf("Successfully assemble %s.\n", f\_name);

symflag = 1;

// 파일 닫고 종료

fclose(fp);

return 1;

}

int asm\_pass\_1(char \*f\_name){

int locctr = 0;

FILE \*src = fopen(f\_name, "r");

FILE \*intmd = fopen("intermediate", "w");

char line[200];

char s1[30], s2[30], s3[30], s4[30], tmpstr[30];

int lnum = 5, cmflag = -1;

int i, k;

adr\_start = 0;

pro\_length = 0;

int tmp = 0, dectmp;

int endflag = 0;

symbol \*symptr, \*pre, \*cur;

OPlist \*opptr;

int idx;

// 한 줄 읽어오기

fgets(line, sizeof(line), src);

// 받아온 줄에서 인자 받기

tmp = sscanf(line, "%s %s %x %s", s1, s2, &adr\_start, tmpstr);

if (tmp){

// START 있으면

if (!strcmp(s2, "START")){

// 정상 형식일 때 START 라인 출력

if (tmp == 3){

locctr = adr\_start;

fprintf(intmd, "%d\t%d\t", lnum, locctr);

lnum += 5;

fprintf(intmd, "%s", line);

}

else {

printf("%d: error: start line error\n", lnum);

return 0;

}

}

// START 없으면 파일 닫고 다시 열기

else {

fclose(src);

src = fopen(f\_name, "r");

}

}

memset(line, '\0', sizeof(line));

// 파읽 끝에 도달할 때 까지 읽기

while (fgets(line, sizeof(line), src) != NULL){

// 초기화

memset(s1, '\0', sizeof(s1));

memset(s2, '\0', sizeof(s2));

memset(s3, '\0', sizeof(s3));

memset(s4, '\0', sizeof(s4));

memset(tmpstr, '\0', sizeof(tmpstr));

tmp = 0;

cmflag = -1;

// 주석이면 그대로 출력

if (line[0] == '.'){

fprintf(intmd, "%d\t%s", lnum, line);

lnum += 5;

}

// 주석이 아니면

else {

// 콤마 있는지 확인

i = 0;

while (line[i] != '\0'){

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

cmflag = i;

break;

}

i++;

} // cmflag : 콤마의 위치

// 입력 받은 라인에서 인자로 값 할당

// label이 안들어온 경우

if (line[0] == ' ' || line[0] == '\t'){

// 콤마가 있으면

if (cmflag != -1){

tmp = sscanf(line, "%s \t %s \t %s \t %s", s2, s3, s4, tmpstr);

if (line[cmflag+1] != ' ' && line[cmflag+1] != '\t'){

if (line[cmflag -1] != ' ' && line[cmflag -1] != '\t'){

k = 0;

while (s3[k] != ',')

k++;

k++;

strcpy(s4, s3+k);

tmp++;

}

}

else if (s4[0] == ','){

if (strlen(s4) == 1){

strcpy(s4, tmpstr);

tmp--;

}

else{

strcpy(s4, s4+1);

s4[strlen(s4) - 1] = '\0';

}

}

if (tmp != 3){

printf("%d: error: wrong input line\n", lnum);

return 0;

}

tmp--;

}

// 콤마가 없으면

else {

tmp = sscanf(line, "%s \t %s \t %s", s2, s3, tmpstr);

if (tmp != 2) {

if (tmp == 1 && !strcmp(s2, "RSUB"))

tmp = 1;

else {

printf("%d: error: wrong input line\n", lnum);

return 0;

}

}

}

}

// 에러처리

else if (line[0] <= '9' && line[0] >= '0') {

printf("%d: error: first char cannot be number\n", lnum);

return 0;

}

// label이 들어온 경우

else {

// 콤마가 있으

if (cmflag != -1){

tmp = sscanf(line, "%s \t %s \t %s \t %s \t %s", s1, s2, s3, s4, tmpstr);

if (line[cmflag+1] != ' ' && line[cmflag+1] != '\t'){

if(line[cmflag-1] != ' ' && line[cmflag-1] != '\t'){

k = 0;

while (s3[k] != ',')

k++;

k++;

strcpy(s4, s3+k);

tmp++;

}

}

else if (s4[0] == ','){

if (strlen(s4) == 1){

strcpy(s4, tmpstr);

tmp--;

}

else{

strcpy(s4, s4+1);

s4[strlen(s4) - 1] = '\0';

}

}

if (tmp != 4){

printf("%d: error: wrong input line\n", lnum);

return 0;

}

tmp--;

}

// 콤마가 없으면

else {

tmp = sscanf(line, "%s \t %s \t %s \t %s", s1, s2, s3, tmpstr);

if (tmp != 3){

printf("%d: error: wrong input line\n", lnum);

return 0;

}

}

}

/\* s1 : label

s2 : directive

s3 : operand

s4 : second operand after comma \*/

// label 이 있을 때

if (s1[0] != '\0'){

// sybol table에서 label 찾기

idx = ZtoAhash(s1);

symptr = symhead[idx];

while (symptr){

if (!strcmp(symptr->name, s1))

break;

symptr = symptr -> link;

}

// label 있으면 중복 에러

if (symptr){

printf("%d: error: duplicate symbol - %s\n", lnum, s1);

return 0;

}

// 없으면 symbol table에 추가

else {

// 새 노드에 값 할당

symbol \*newNode = (symbol\*)malloc(sizeof(symbol));

strcpy(newNode->name, s1);

newNode->link = NULL;

newNode->locctr = locctr;

// head가 비었으면 head가 새 노드

if (symhead[idx] == NULL)

symhead[idx] = newNode;

// head가 NULL이 아니면

else {

pre = symhead[idx];

cur = pre;

// Z to A 순으로 linked list 저장

while (cur){

k = 0;

while (1){

if (cur->name[k] != s1[k])

break;

k++;

}

if (cur->name[k] < s1[k])

break;

pre = cur;

cur = cur -> link;

}

// 올바른 위치에 new node 삽입

if (cur == pre){

symhead[idx] = newNode;

symhead[idx]->link = cur;

}

else{

pre->link = newNode;

newNode->link = cur;

}

}

}

}

// directive가 opcode list에 있는지 확인

if (s2[0] == '+')

idx = hashFunction(s2+1);

else

idx = hashFunction(s2);

opptr = OPhead[idx];

while (opptr){

if (s2[0] != '+'){

if (!strcmp(opptr->inst\_name, s2))

break;

}

else{

if (!strcmp(opptr->inst\_name, s2+1))

break;

}

opptr = opptr -> link;

}

// opcode list에 directive가 있으면

if (opptr){

// 출력형식에 맞춰 출력

if (s1[0] != '\0')

fprintf(intmd, "%d\t%d\t%s", lnum, locctr, line);

else

fprintf(intmd, "%d\t%d\t-%s", lnum, locctr, line);

// format만큼 location counter 증가

if (!strcmp(opptr->form, "1"))

locctr += 1;

else if (!strcmp(opptr->form, "2"))

locctr += 2;

else if (s2[0] == '+')

locctr += 4;

else

locctr += 3;

// 줄번호 증가

lnum += 5;

}

// directive가 "BASE"일 때

else if (!strcmp(s2, "BASE")){

fprintf(intmd, "%d\t-1\t-%s", lnum, line);

lnum += 5;

}

// directive가 "END"일 때

else if (!strcmp(s2, "END") && tmp <= 2){

fprintf(intmd, "%d\t-1\t-%s", lnum, line);

lnum += 5;

endflag = 1;

break;

}

// directive가 "WORD" 일 때

else if (!strcmp(s2, "WORD") && tmp == 3) {

fprintf(intmd, "%d\t%d\t%s", lnum, locctr, line);

locctr += 3;

memset(tmpstr, '\0', sizeof(tmpstr));

tmp = sscanf(s3, "%d %s", &dectmp, tmpstr);

if (tmp != 1){

printf("%d: error: 'WORD' operand error - %s\n", lnum, s3);

return 0;

}

}

// directive가 "BYTE" 일때

else if (!strcmp(s2, "BYTE") && tmp == 3){

fprintf(intmd, "%d\t%d\t", lnum, locctr);

// 뒤에 C이면

if (s3[0] == 'C')

locctr += strlen(s3) -3;

// 뒤에 X이면

else if (s3[0] == 'X')

locctr += (strlen(s3) - 2) / 2;

// 둘다 아니면 에러

else {

printf("%d: error: 'BYTE' input error\n", lnum);

return 0;

}

lnum += 5;

fprintf(intmd, "%s", line);

}

// directive가 "RESW"일 때

else if (!strcmp(s2, "RESW") && tmp == 3){

fprintf(intmd, "%d\t%d\t%s", lnum, locctr, line);

lnum += 5;

locctr += 3 \* atoi(s3);

}

// directive가 "RESB"일 때

else if (!strcmp(s2, "RESB") && tmp == 3){

fprintf(intmd, "%d\t%d\t%s", lnum, locctr, line);

lnum += 5;

locctr += atoi(s3);

}

// 위에 언급된 것 중에 없고 opcodelist에도 없으면 에러

else {

printf("%d: error: Invlid directive - %s\n", lnum, s2);

return 0;

}

}

memset(line, '\0', sizeof(line));

}

// END가 없으면 에러 처리 후 종료

if (endflag == 0){

printf("error: No END sign\n");

return 0;

}

// 파일 포인터 종료

fclose(src);

fclose(intmd);

// 프로그램 길이

pro\_length = locctr - adr\_start;

// 패스2에서 오류 발생시 종료

if (asm\_pass\_2(f\_name) == 0)

return 0;

return 1;

}

int asm\_pass\_2(char \*f\_name){

FILE \*intmd, \*obj, \*lst;

char of\_name[100], lf\_name[100];

char line[200];

char s1[30], s2[30], s3[30], s4[30];

int k = 0, l = 0;

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

int pc\_adr = adr\_start;

int b\_adr = 0;

int tmp = 0;

int lnum, locctr = 0;

int cmflag = -1;

int format;

int idx;

OPlist \*opptr;

symbol \*symptr;

char reg[10][2];

char objcode[10];

int newop, disp;

int dectmp, xbpe;

char strtmp[100], tmpstr[100];

char bytetmp[100];

// 인덱스가 레지스터 값

strcpy(reg[0], "A");

strcpy(reg[1], "X");

strcpy(reg[2], "L");

strcpy(reg[3], "B");

strcpy(reg[4], "S");

strcpy(reg[5], "T");

strcpy(reg[6], "F");

strcpy(reg[7], " ");

strcpy(reg[8], "PC");

strcpy(reg[9], "SW");

memset(of\_name, '\0', sizeof(of\_name));

memset(lf\_name, '\0', sizeof(lf\_name));

// lst, obj 파일명 생성하여 쓰기로 파일 열기

while (f\_name[k] != '.'){

of\_name[k] = f\_name[k];

lf\_name[k] = f\_name[k];

k++;

}

strcat(of\_name, ".obj");

strcat(lf\_name, ".lst");

// 파일 열기

intmd = fopen("intermediate", "r");

obj = fopen(of\_name, "w");

lst = fopen(lf\_name, "w");

// 첫 줄 받아온다

fgets(line, sizeof(line), intmd);

tmp = sscanf(line, "%d \t %d \t %s \t %s \t %s", &lnum, &locctr, s1, s2, s3);

if (tmp){

// START가 있으면 lst, obj 파일 작성

if (!strcmp(s2, "START")){

fprintf(lst, "%d\t%04X\t%s\t%s\t%X\n", lnum, locctr, s1, s2, adr\_start);

fprintf(obj, "H%-6s%06X%06X\n", s1, adr\_start, pro\_length);

}

// 없으면 obj 파일 H 부분 작성

else {

fclose(intmd);

intmd = fopen("intermediate", "r");

fprintf(obj, "H %06X%06X\n", adr\_start, pro\_length);

}

}

// 계속해서 파일 읽기

while (1){

// 초기화

n = 1;

i = 1;

x = 0;

b = 0;

p = 0;

e = 0;

memset(line, '\0', sizeof(line));

memset(s1, '\0', sizeof(s1));

memset(s2, '\0', sizeof(s2));

memset(s3, '\0', sizeof(s3));

memset(s4, '\0', sizeof(s4));

memset(objcode, '\0', sizeof(objcode));

memset(strtmp, '\0', sizeof(strtmp));

memset(bytetmp, '\0', sizeof(bytetmp));

memset(tmpstr, '\0', sizeof(tmpstr));

cmflag = -1;

tmp = 0;

xbpe = 0;

disp = 0;

// 초기화

fgets(line, sizeof(line), intmd);

if (line[0] == '\0')

break;

// 콤마 있는지 확인

k = 0;

while (line[k] != '\0'){

if (line[k] == ','){

cmflag = k;

break;

}

k++;

} // cmflag : 콤마의 위치

// 인자 입력받기

// 콤마 없으면

if (cmflag == -1){

tmp = sscanf(line, "%d \t %d \t %s \t %s \t %s", &lnum, &locctr, s1, s2, s3);

}

// 콤마 있으면

else {

tmp = sscanf(line, "%d %d \t %s \t %s \t %s \t %s \t %s", &lnum, &locctr, s1, s2, s3, s4, tmpstr);

if (line[cmflag+1] != ' ' && line[cmflag+1] != '\t'){

if(line[cmflag-1] != ' ' && line[cmflag-1] != '\t'){

k = 0;

while (s3[k] != ',')

k++;

k++;

strcpy(s4, s3+k);

tmp++;

s3[strlen(s3) - strlen(s4) - 1] = '\0';

}

}

else if (s4[0] == ','){

if (strlen(s4) == 1){

strcpy(s4, tmpstr);

tmp--;

}

else{

strcpy(s4, s4+1);

s4[strlen(s4) - 1] = '\0';

}

}

else{

s3[strlen(s3) - 1] = '\0';

}

}

/\* s1 : label

s2 : directive

s3 : operand

s4 : 2nd operand after comma \*/

// 주석이면 그대로 출력

if (s1[0] == '\0'){

fprintf(lst, "%s", line);

}

else {

// 'END' 면 종료

if (!strcmp(s2, "END")){

fprintf(lst, "%d\t\t\t%s\t%s\n", lnum, s2, s3);

break;

}

// 'BASE'면 b 레지스터에 값 넣

if (!strcmp(s2, "BASE")){

idx = ZtoAhash(s3);

symptr = symhead[idx];

while (symptr){

if (!strcmp(symptr->name, s3))

break;

symptr = symptr -> link;

}

if (!symptr){

printf("%d: error: 'BASE' symbol error - %s\n", lnum, s3);

return 0;

}

b\_adr = symptr->locctr;

fprintf(lst, "%d\t\t\t%s\t%s\n", lnum, s2, s3);

}

// operand 확인

switch (s3[0]){

// '#' 있으면 immediate addresing

case '#':

n = 0;

i = 1;

b = 0;

if (s3[1] >= '0' && s3[1] <= '9')

p = 0;

break;

// '@' 있으면 indirect addressing

case '@':

n = 1;

i = 0;

break;

// 보통의 경우는 simple addressing

default:

n = 1;

i = 1;

break;

}

// opcode list에 있는 명령어인지 확인

if (s2[0] == '+')

idx = hashFunction(s2+1);

else

idx = hashFunction(s2);

opptr = OPhead[idx];

while (opptr){

if (s2[0] != '+'){

if (!strcmp(opptr->inst\_name, s2))

break;

}

else{

if (!strcmp(opptr->inst\_name, s2+1))

break;

}

opptr = opptr -> link;

}

// opcode list에 입력된 mnemonic이 있으면

if (opptr){

// 몇 형식인지 확인

if (!strcmp(opptr->form, "1"))

format = 1;

else if (!strcmp(opptr->form, "2"))

format = 2;

else if (s2[0] == '+'){

format = 4;

b = 0;

p = 0;

e = 1;

}

else

format = 3;

// pc register에 값 저장

pc\_adr = locctr + format;

// 1형식인 경우

// OPCODE가 obj code

if (format == 1){

if (s3[0] != '\0'){

printf("%d: error: format1 cannot have operand\n", lnum);

return 0;

}

if (s1[0] == '-')

fprintf(lst, "%d\t%04X\t\t%s\t\t%02X\n", lnum, locctr, s2, opptr->opcode);

else

fprintf(lst, "%d\t%04X\t%s\t%s\t\t%02X\n", lnum, locctr, s1, s2, opptr->opcode);

// 오브젝트 코드 문자열로 저장

sprintf(objcode, "%02X", opptr->opcode);

add\_objlist(objcode, locctr, 0);

}

// 2형식인 경우

// OPcode 뒤에 register 번호

else if (format == 2){

// 인자 없으면 에러

if (s3[0] == '\0'){

printf("%d: error: format2 need at least one operand\n", lnum);

return 0;

}

// 유효한 레지스터인지 확인

for (k = 0 ; k < 10 ; k++){

if (!strcmp(s3, reg[k]))

break;

}

// 아니면 에러

if (k == 10){

printf("%d: error: wrong register name - %s\n", lnum, s3);

return 0;

}

// 파일에 출력

if (s1[0] == '-')

fprintf(lst, "%d\t%04X\t\t%s\t%s", lnum, locctr, s2, s3);

else

fprintf(lst, "%d\t%04X\t%s\t%s\t%s", lnum, locctr, s1, s2, s3);

l = 0;

if (cmflag != -1){ // 콤마 있으면

for (l = 0; l < 10 ; l++){

if (!strcmp(s4, reg[l]))

break;

}

if (l == 10){

printf("%d: error: wrong register name - %s\n", lnum, s4);

return 0;

}

fprintf(lst, ", %s", s4);

}

fprintf(lst, "\t\t%02X%d%d\n", opptr->opcode, k, l);

// 오브젝트 코드 문자열로 저장

sprintf(objcode, "%02X%d%d", opptr->opcode, k, l);

add\_objlist(objcode, locctr, 0);

}

// 3형식인 경우

else if (format == 3){

// opcode 뒤에 2 bits 값을 n, i로 대체

newop = opptr->opcode / 2 / 2 \* 2 \*2 + 2 \* n + i;

// xbpe 값

xbpe = 2\*2\*2\*x+2\*2\*b+2\*p+e;

// immediate addressing (#)

if (n == 0 && i == 1){

tmp = sscanf(s3+1, "%d %s", &dectmp, strtmp);

// # 뒤에 상수 이면

if (tmp == 1){

// object 코드 저장

sprintf(objcode, "%02X%01X%03X", newop, xbpe, dectmp);

}

else{

tmp = strlen(s3) - 1;

strcpy(s3, s3+1);

s3[tmp] = '\0';

}

}

// 상수가 아니어서

//오브젝코드가 생성되지 않았을때

if (objcode[0] == '\0'){

// indirect addressing (@)

if (n == 1 && i == 0){

tmp = strlen(s3) - 1;

strcpy(s3, s3+1);

s3[tmp] = '\0';

}

// symbol table에 있는 operand 인지 확인

if (s3[0] != '\0'){

idx = ZtoAhash(s3);

symptr = symhead[idx];

while (symptr){

if (!strcmp(symptr->name, s3))

break;

symptr = symptr -> link;

}

// 없으면 에러 출력하고 종료

if (!symptr){

printf("%d: error: symbol error - %s\n", lnum, s3);

return 0;

}

// disp 값 저장

p = 1; b = 0;

disp = symptr->locctr - pc\_adr;

// pc relative 사용할 수 없는 경우

if (disp > 2047 || disp < (-2048)){

// base relative 사용

b = 1;

p = 0;

disp = symptr->locctr - b\_adr;

}

if (disp < 0)

disp = disp & 4095;

if (cmflag != -1){ // 콤마 있을 때

// X 레지스터 들어오면 x 조정

if (!strcmp(s4, "X")){

x = 1;

}

// X 레지스터 아니면 에러

else {

printf("%d: error: wrong register name - %s\n", lnum, s4);

return 0;

}

}

}

// obj code 저장

xbpe = 2\*2\*2\*x + 2\*2\*b + 2\*p + e;

sprintf(objcode, "%02X%01X%03X", newop, xbpe, disp);

}

// lst 파일에 출력

if (s1[0] == '-')

fprintf(lst, "%d\t%04X\t\t%s\t%s", lnum, locctr, s2, s3);

else

fprintf(lst, "%d\t%04X\t%s\t%s\t%s", lnum, locctr, s1, s2, s3);

if (cmflag != -1)

fprintf(lst, ", %s\t%s\n", s4, objcode);

else

fprintf(lst, "\t\t%s\n", objcode);

add\_objlist(objcode, locctr, 0);

}

// 4형식인 경우

else if (format == 4){

// opcode 뒤에 2 bits 값을 n, i로 대체

newop = opptr->opcode / 2 / 2 \* 2 \*2 + 2 \* n + i;

// xbpe 값

xbpe = 2\*2\*2\*x+2\*2\*b+2\*p+e;

// immediate addressing (#)

if (n == 0 && i == 1){

tmp = sscanf(s3+1, "%d %s", &dectmp, strtmp);

// # 뒤에 상수 이면

if (tmp == 1){

// object 코드 저장

sprintf(objcode, "%02X%01X%05X", newop, xbpe, dectmp);

}

else{

tmp = strlen(s3) - 1;

strcpy(s3, s3+1);

s3[tmp] = '\0';

}

add\_objlist(objcode, locctr, 0);

}

// 상수가 아니어서

//오브젝코드가 생성되지 않았을때

if (objcode[0] == '\0'){

// indirect addressing (@)

if (n == 1 && i == 0){

tmp = strlen(s3) - 1;

strcpy(s3, s3+1);

s3[tmp] = '\0';

}

// symbol table에 있는 operand 인지 확인

if (s3[0] != '\0'){

idx = ZtoAhash(s3);

symptr = symhead[idx];

while (symptr){

if (!strcmp(symptr->name, s3))

break;

symptr = symptr -> link;

}

// 없으면 에러 출력하고 종료

if (!symptr){

printf("%d: error: symbol error - %s\n", lnum, s3);

return 0;

}

if (cmflag != -1){ // 콤마 있을 때

// X 레지스터 들어오면 x 조정

if (!strcmp(s4, "X")){

x = 1;

}

// X 레지스터 아니면 에러

else {

printf("%d: error: wrong register name - %s\n", lnum, s4);

return 0;

}

}

}

// obj code 저장

xbpe = 2\*2\*2\*x + 2\*2\*b + 2\*p + e;

sprintf(objcode, "%02X%01X%05X", newop, xbpe, symptr->locctr);

add\_objlist(objcode, locctr, 1);

}

// lst 파일에 출력

if (s1[0] == '-')

fprintf(lst, "%d\t%04X\t\t%s\t%s", lnum, locctr, s2, s3);

else

fprintf(lst, "%d\t%04X\t%s\t%s\t%s", lnum, locctr, s1, s2, s3);

if (cmflag != -1)

fprintf(lst, ", %s\t%s\n", s4, objcode);

else

fprintf(lst, "\t\t%s\n", objcode);

}

}

// 입력이 opcode list에 저장된 mnemonic이 아니면

else {

// 'BYTE'인 경우

if (!strcmp(s2, "BYTE")){

// 'C' 형식으로 들어올 때

if (s3[0] == 'C'){

strcpy(bytetmp, s3+2);

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

sprintf(objcode, "%02X", bytetmp[0]);

k = 1;

while (bytetmp[k] != '\0'){

sprintf(strtmp, "%02X", bytetmp[k]);

strcat(objcode, strtmp);

k++;

}

}

// 'X' 형식으로 들어올 때

else if (s3[0] == 'X'){

strcpy(objcode, s3+2);

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

}

fprintf(lst, "%d\t%04X\t%s\t%s\t%s\t\t%s\n", lnum, locctr, s1, s2, s3, objcode);

add\_objlist(objcode, locctr, 0);

}

// 'WORD'인 경우

else if (!strcmp(s2, "WORD")){

sscanf(s3, "%d", &k);

sprintf(objcode, "%06X", k);

fprintf(lst, "%d\t%04X\t%s\t%s\t%s\t\t%s\n", lnum, locctr, s1, s2, s3, objcode);

add\_objlist(objcode, locctr, 0);

}

// 'RESB' 나 'RESW' obj code 없음

else if (!strcmp(s2, "RESB") || !strcmp(s2, "RESW")){

fprintf(lst,"%d\t%04X\t%s\t%s\t%s\n", lnum, locctr, s1, s2, s3);

}

}

}

}

// .lst 파일 출력 완료

// .obj 파일 출력 시작

// T 부분

objlist \*objptr = objhead;

char ttmp[62];

memset(ttmp, '\0', sizeof(ttmp));

do{

fprintf(obj, "T");

fprintf(obj, "%06X", objptr->loc);

while (objptr && strlen(ttmp) + strlen(objptr->objcode) <= 62){

strcat(ttmp, objptr->objcode);

objptr = objptr -> link;

if (!objptr)

break;

if (objptr->flag == 1)

break;

}

fprintf(obj, "%02X", (unsigned int)strlen(ttmp) / 2);

fprintf(obj, "%s\n", ttmp);

memset(ttmp, '\0', sizeof(ttmp));

} while (objptr);

// M 부분

objptr = objhead;

while (objptr){

if (objptr->mflag ==1){

fprintf(obj, "M");

fprintf(obj, "%06X", objptr->loc+ 1);

fprintf(obj, "05\n");

}

objptr = objptr->link;

}

// E 부분

fprintf(obj, "E");

fprintf(obj, "%06X\n", adr\_start);

fclose(intmd);

fclose(obj);

fclose(lst);

return 1;

}

int ZtoAhash(char\* label){

// Z to A 순서로

return 25 - (label[0] - 'A');

}

int symbolcmd(){

symbol \*ptr;

int i = 0;

// symbol table 만들어져있는지 확인

if (symflag == 1){

// symbol table 내용 차례로 출력

while (i < 26){

ptr = symhead[i];

while (ptr) {

printf("\t%s\t%04X\n", ptr->name, ptr->locctr);

ptr = ptr -> link;

}

i++;

}

}

// table 없으면 에러

else {

printf("No symbol table\n");

return 0;

}

return 1;

}

void add\_objlist(char\* objcode, int loc, int m){

objlist \*newNode = (objlist\*)malloc(sizeof(objlist)); // 새로운 값을 저장할 노드

objlist \*ptr = objhead; // linked list 탐색에 사용되는 임시변수

strcpy(newNode->objcode, objcode);

newNode->link = NULL;

newNode->loc = loc;

newNode->flag = 0;

newNode->mflag = 0;

// 새 노드에 값 저장

if (objhead == NULL) { // head가 NULL 이면

objhead = (objlist\*)malloc(sizeof(objlist));

objhead = newNode;

// head 할당한 후 새 노드 저장

if (m == 1)

newNode->mflag = 1;

}

else { // head가 NULL하지 않으면

while (ptr->link != NULL)

ptr = ptr->link; // 새 노드를 연결할 노드까지 이동

ptr-> link = newNode; // 새 노드 연결

if ((newNode->loc) - (ptr->loc) >= 4096)

newNode->flag = 1;

if (m == 1)

newNode->mflag = 1;

}

}

/\* end of proj 2 \*/

/\* start of proj 3 \*/

int progaddr(char\* addr){

int tmp1; // string형 명령어에서 address 부분만 추출할 변수

char tmp2[10]; // address 말고 다른 입력 있는지 확인할 변수

// string에서 address 부분 추출

// progaddr 뒤에 들어온 파라미터가 1개가 아니면 에러

if (sscanf(addr, "%X %s", &tmp1, tmp2) != 1){

printf("Invalid command\n");

return 0;

}

// 전역변수 startaddr을 입력받은 수로 할당

startaddr = tmp1;

return 1;

}

int loader(char\* files){

char fileset[3][50], tmp[50];

int filenum, totlen = 0;

estab \*cur;

// parameter string에서 파일명을 하나씩 가져온다

filenum = sscanf(files, "%s %s %s %s", fileset[0], fileset[1], fileset[2], tmp);

// 파일 개수가 1~3이 아니면 return 0

if (filenum <= 0 || filenum > 3)

return 0;

// ESTAB head pointer 할당

eshead = (estab\*)malloc(sizeof(estab));

eshead->address = -1;

eshead->link = NULL;

// load 과정에서 에러가 나면 종료

if (loader\_pass\_1(filenum, fileset) == 0)

return 0;

// load map 출력(ESTAB)

printf("control\tsymbol\taddress\tlength\nsection\tname\n");

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

cur = eshead;

while (1){

cur = cur->link;

if (cur->length != -1){ // control section(program name)

printf("%s\t\t%04X\t%04X\n", cur->symbol, cur->address, cur->length);

totlen += cur->length;

}

else // symbol name

printf("\t%s\t%04X\n", cur->symbol, cur->address);

if (!cur->link)

break;

}

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

printf(" total length %04X\n", totlen);

// 프로그램 길이 전역 변수에 저장

tot\_length = totlen;

memset(register\_set, 0, sizeof(register\_set)); // 레지스터0으로 초기화

register\_set[8] = startaddr; // PC는 프로그램 시작주소

register\_set[2] = totlen; // L은 프로그램 길이

// ESTAB 할당 해제

free(eshead);

return 1;

}

int loader\_pass\_1(int filenum, char (\*fileset)[50]){

int i, j, len = 0, csaddr = startaddr, symaddr;

FILE \*fp;

char line[270], symname[10];

estab \*cur;

for (i = 0; i < filenum; i++){ // object 파일 수만큼 반복

fp = fopen(fileset[i], "r"); // object 파일 open

// 파일이 없으면 error 출력

if (!fp){

printf("No such file in this directory\n");

return 0;

}

memset(line, '\0', sizeof(line)); // buffer 초기화

// object file 한줄씩 읽어오기

while (fgets(line, sizeof(line), fp)){

// read H record

if (line[0] == 'H'){

sscanf(line, "%\*c%6s%6x%6x", symname, &symaddr, &len);

// program name과 중복된 symbol 있는지 확인

cur = eshead;

while (1){

// ESTAB에 중복된 symbol 있으면 에러

if (!strcmp(symname, cur->symbol)){

printf("Duplicate external symbol: %s\n", symname);

return 0;

}

if (!(cur->link))

break;

cur = cur->link;

}

// 중복된 symbol 없으면 새 node 만들어서 연결

estab \*newNode;

newNode = (estab\*)malloc(sizeof(estab));

strcpy(newNode->symbol, symname);

newNode->address = csaddr;

newNode->length = len;

newNode->link = NULL;

cur->link = newNode;

}

// read D record

else if (line[0] == 'D'){

j = 1;

while(1){

sscanf(line + j, "%6s%6X", symname, &symaddr);

// external symbol과 중복된 symbol 있는지 확인

cur = eshead;

while (1){

// ESTAB에 중복된 symbol 있으면 에러

if (!strcmp(symname, cur->symbol)){

printf("Duplicate external symbol: %s\n", symname);

return 0;

}

if (!(cur->link))

break;

cur = cur->link;

}

// 중복된 symbol 없으면 새 노드 만들어서 연결

estab \*newNode;

newNode = (estab\*)malloc(sizeof(estab));

strcpy(newNode->symbol, symname);

newNode->address = symaddr + csaddr;

newNode->length = -1;

newNode->link = NULL;

cur->link = newNode;

// 다음 external symbol 확인을 위해 index를 12 올린다

j += 12;

// D record가 끝나면 while문 종료

if (line[j] == ' ' || line[j] == '\n' || line[j] == '\0' || line[j] == EOF)

break;

}

}

// end loop

else if (line[0] == 'E')

break;

memset(line, '\0', sizeof(line)); // buffer 초기화

}

csaddr += len; // 다음 프로그램의 시작 주소 저장

fclose(fp); // 파일 포인터 닫기

}

// pass2에서 에러가 나면 return 0

if (loader\_pass\_2(filenum, fileset)== 0)

return 0;

return 1;

}

int loader\_pass\_2(int filenum, char (\*fileset)[50]){

int i, j;

int csaddr = startaddr, execaddr = startaddr, cslth;

int refnum, rcdlen, rcdaddr, symaddr, objcode, mod, tmp;

FILE \*fp;

char operator;

char line[270], symname[10], argument[10];

estab \*cur;

for (i = 0; i < filenum; i++){ // object 파일 수만큼 반복

fp = fopen(fileset[i], "r"); // 파잎 포인터 열기

// 파일이 없으면 error 출력

if (!fp){

printf("No such file in this directory\n");

return 0;

}

int refarr[10]; // reference number array

refarr[1] = csaddr; // reference number 1번은 control section address

memset(line, '\0', sizeof(line)); // buffer 초기화

// object file 한줄씩 읽어오기

while (fgets(line, sizeof(line), fp)){

execaddr = csaddr;

// read H record

if (line[0] == 'H'){

sscanf(line, "%\*c%\*6s%\*6x%6x", &cslth);

}

// read R record

else if (line[0] == 'R'){

j = 1;

while(1){

// 숫자로 시작하면 reference number

if (line[j] >= '0' && line[j] <= '9'){

sscanf(line + j, "%2x%6s", &refnum, symname);

// symbol의 address를 해당 reference number에 저장

cur = eshead;

while (1){

if (!strcmp(symname, cur->symbol)){

refarr[refnum] = cur->address;

break;

}

// symbol이 ESTAB에 없으면 에러

if (!(cur->link)){

printf("There is no external symbol in ESTAB: %s\n", symname);

return 0;

}

cur = cur->link;

}

j += 8;

}

// 숫자로 시작하지 않으면 넘어간다

else {

j += 6;

}

// R record 끝나면 while문 종료

if (line[j] == ' ' || line[j] == '\n' || line[j] == '\0' || line[j] == EOF)

break;

}

}

// read T record

else if (line[0] == 'T'){

j = 1;

sscanf(line + j, "%6X%2X", &rcdaddr, &rcdlen);

execaddr += rcdaddr;

j += 8;

// 1 byte씩 메모리에 저장

while (j < (rcdlen \* 2 + 9)){

sscanf(line + j, "%2X", &objcode);

sprintf(argument, "%X, %X", execaddr + (j - 9) / 2, objcode);

edit(argument);

j += 2;

memset(argument, '\0', sizeof(argument));

}

}

// read M record

else if (line[0] == 'M'){

sscanf(line + 1, "%6X%2X%c%6s", &rcdaddr, &mod, &operator, symname);

execaddr = rcdaddr + csaddr;

// reference number 사용

if (symname[0] >= '0' && symname[0] <= '9'){

symaddr = refarr[atoi(symname)];

}

// symbol name 사용

else {

// ESTAB에서 symbol 탐색

cur = eshead;

while (1){

if (!strcmp(symname, cur->symbol)){

symaddr = cur->address;

break;

}

// symbol이 없으면 에러

if (!(cur->link)){

printf("There is no external symbol in ESTAB: %s\n", symname);

return 0;

}

cur = cur->link;

}

}

// modification

// 메모리에서 해당하는 값 가져오기

tmp = 0;

for (j = 0; j < (mod + 1) / 2; j++){

tmp += memory[execaddr + j];

if (j != (mod + 1) / 2 - 1)

tmp <<= 8;

}

// 수정할 부분이 홀수이면 앞에 halfbyte는 없앤다

if ((mod % 2) == 1)

tmp &= 0x0FFFFF;

// - operator면 음수로 바꿔준다

if (operator == '-'){

symaddr \*= -1;

}

// operator가 +, - 둘 다 아니면 에러

else if (operator != '+'){

printf("wrong operator: %c\n", operator);

return 0;

}

// M record에 있는 symbol의 address와 더해준다

tmp += symaddr;

// 수정할 메모리 뒤에서부터 바뀐 값으로 변경

for (j = (mod + 1) / 2 - 1; j >= 1; j--){

memory[execaddr + j] = tmp & 0xFF;

tmp >>= 8;

}

// 수정할 부분이 홀수이면 앞에 half byte는 살려두고 뒤 half byte만 변경한다

if ((mod % 2) == 1){

memory[execaddr + j] /= 16;

memory[execaddr + j] \*= 16;

memory[execaddr + j] += tmp & 0x0F;

}

// 짝수이면 해당 메모리 값 1byte 모두 변경한다

else

memory[execaddr + j ] = tmp & 0xF;

}

// loop end

else if (line[0] == 'E')

break;

// buffer 초기화

memset(line, '\0', sizeof(line));

}

csaddr += cslth; // 다음 프로그램의 시작주소

fclose(fp); // 파일 포인터 닫기

}

return 1;

}

int bp(char \*bpcmd){

int addr;

char tmp[10];

bplist \*cur = bphead, \*rm;

// "bp"만 입력했을 경우

if (bpcmd[0] == '\0'){

// break point 모두 출력

printf("\t breakpoint\n");

printf("\t ----------\n");

// bp list를 탐색하며 모두 출력한다

while (cur->link){

cur = cur->link;

printf("\t %X\n", cur->bp);

}

}

// "bp [address]" 입력했을 경우

else if (sscanf(bpcmd, "%X %s", &addr, tmp) == 1 && strcmp(bpcmd, "clear")){

// bp가 프로그램 범위를 벗어나면 에러

if (addr < startaddr || addr > startaddr + tot\_length){

printf("Breakpoint is out of program\n");

return 0;

}

// 새 node를 만들어서 연결

bplist \*newNode;

newNode = (bplist\*)malloc(sizeof(bplist));

newNode->bp = addr;

newNode->link = NULL;

while (cur->link)

cur = cur->link;

cur->link = newNode;

// bp 추가 완료 메세지 출력

printf("\t [ok] create breakpoint %X\n", newNode->bp);

}

// "bp clear" 입력했을 경우

else if (!strcmp("clear", bpcmd)){

// bp list를 돌면서 head 빼고 모두 할당 해제한다

cur = cur->link;

while (cur){

rm = cur;

cur = cur->link;

free(rm);

}

bphead->link = NULL;

// bp 삭제 완료 메세지 출력

printf("\t [ok] clear all breakpoints\n");

}

// 그 외에는 잘못된 명령

else {

printf("Invalid command\n");

return 0;

}

return 1;

}

/\*register\*/

/\*--------\*/

/\* A 0 \*/

/\* X 1 \*/

/\* L 2 \*/

/\* B 3 \*/

/\* S 4 \*/

/\* T 5 \*/

/\* F 6 \*/

/\* PC 8 \*/

/\* SW 9 \*/

/\*--------\*/

int run(){

int op1, op2, opcode, ni, xbpe;

int taraddr, tarval;

int r1, r2;

int bpflag = 0;

bplist \*cur;

while(register\_set[8] < startaddr + tot\_length){ // PC 값이 프로그램 범위 이내일 때 loop문 반복

// bp pointer 초기화

cur = bphead;

// bp flag 0으로 set

bpflag = 0;

// 해당 memory에서 opcode 가져오기

op1 = memory[register\_set[8]] / 16;

op2 = memory[register\_set[8]] % 16;

opcode = memory[register\_set[8]] & 0xFC;

// 1형식일 경우

if (op1 == 0xC || op1 == 0xF){

register\_set[8] += 1; // PC 값 1 더해준다 (1형식)

}

// 2형식일 경우

else if (op1 >= 0x9 && op1 <= 0xB){

r1 = memory[register\_set[8] + 1] / 16; // 첫 번째 register number

r2 = memory[register\_set[8] + 1] % 16; // 두 번째 register number

register\_set[8] += 2; // PC 값 2 더해준다 (2형식)

// 명령어 실행

switch (opcode){

case 0x90: // ADDR

register\_set[r2] += register\_set[r1];

break;

case 0x94: // SUBR

register\_set[r2] -= register\_set[r1];

break;

case 0xA0: // COMPR

if (register\_set[r1] > register\_set[r2])

register\_set[9] = '>';

else if (register\_set[r1] < register\_set[r2])

register\_set[9] = '<';

else

register\_set[9] = '=';

break;

case 0xA4: // SHIFTL

while (r2--){

register\_set[r1] = ((register\_set[r1] << 1) & 0xFFFFF0) + (register\_set[r1] / 0x800000);

}

register\_set[r1] &= 0xFFFFFF;

break;

case 0xA8: // SHIFTR

while (r2--){

register\_set[r1] = (register\_set[r1] >> 1) + ((register\_set[r1] & 0x01) \* 0x800000);

}

register\_set[r1] &= 0xFFFFFF;

break;

case 0xAC: // RMO

register\_set[r2] = register\_set[r1];

break;

case 0xB4: // CLEAR

register\_set[r1] = 0;

break;

case 0xB8: // TIXR

register\_set[1]++;

if (register\_set[1] > register\_set[r1])

register\_set[9] = '>';

else if (register\_set[1] < register\_set[r1])

register\_set[9] = '<';

else

register\_set[9] = '=';

break;

}

}

// 3형식 또는 4형식일 경우

else {

ni = op2 - opcode % 16; // ni값 계산

xbpe = memory[register\_set[8] + 1] / 16; // xbpe값 계산

taraddr = target\_addr(xbpe); // target address 계산

tarval = target\_val(ni, &taraddr); // target address의 value 계산

// 명령어 실행

switch (opcode){

// Load

case 0x00: // LDA

register\_set[0] = tarval;

break;

case 0x68: // LDB

register\_set[3] = tarval;

break;

case 0x50: // LDCH

register\_set[0] &= 0xFFFF00;

register\_set[0] += (tarval >> 16);

break;

case 0x70: // LDF

register\_set[6] = tarval;

break;

case 0x08: // LDL

register\_set[2] = tarval;

break;

case 0x6C: // LDS

register\_set[4] = tarval;

break;

case 0x74: // LDT

register\_set[5] = tarval;

break;

case 0x04: // LDX

register\_set[1] = tarval;

break;

// Store

case 0x0C: // STA

store\_inst(0, taraddr);

break;

case 0x10: // STX

store\_inst(1, taraddr);

break;

case 0x14: // STL

store\_inst(2, taraddr);

break;

case 0x54: // STCH

memory[taraddr] = (unsigned char)(register\_set[0] & 0xF);

break;

case 0x78: // STB

store\_inst(3, taraddr);

break;

case 0x7C: // STS

store\_inst(4, taraddr);

break;

case 0x80: // STF

store\_inst(6, taraddr);

break;

case 0x84: // STT

store\_inst(5, taraddr);

break;

case 0xE8: // STSW

store\_inst(9, taraddr);

break;

// Arithmetic operation

case 0x18: // ADD

register\_set[0] += tarval;

break;

case 0x1C: // SUB

register\_set[0] -= tarval;

break;

// Compare

case 0x28: // COMP

if (register\_set[0] > tarval)

register\_set[9] = '>';

else if (register\_set[0] < tarval)

register\_set[9] = '<';

else

register\_set[9] = '=';

break;

case 0x2C: // TIX

register\_set[1]++;

if (register\_set[1] > tarval)

register\_set[9] = '>';

else if (register\_set[1] < tarval)

register\_set[9] = '<';

else

register\_set[9] = '=';

break;

// Jump

case 0x30: // JEQ

if (register\_set[9] == '=')

register\_set[8] = taraddr;

break;

case 0x34: // JGT

if (register\_set[9] == '>')

register\_set[8] = taraddr;

break;

case 0x38: // JLT

if (register\_set[9] == '<')

register\_set[8] = taraddr;

break;

case 0x3C: // J

register\_set[8] = taraddr;

break;

case 0x48: // JSUB

register\_set[2] = register\_set[8];

register\_set[8] = taraddr;

break;

case 0x4C: // RSUB

register\_set[8] = register\_set[2];

break;

// Logical operation

case 0x40: // AND

register\_set[0] &= tarval;

break;

case 0x44: // OR

register\_set[0] |= tarval;

break;

// I/O

case 0xE0: // TD

register\_set[9] = '<';

break;

case 0xDC: // WD

break;

case 0xD8: // RD

break;

}

}

// 현재 PC가 bp list에 있는지 확인

while (cur->link){

cur = cur->link;

if (cur->bp == register\_set[8]){

bpflag = 1;

break;

}

}

// bp list에 있으면 반복문 종료

if (bpflag)

break;

}

// register 값들 출력

printf("A : %06X X : %06X\n", register\_set[0], register\_set[1]);

printf("L : %06X PC : %06X\n", register\_set[2], register\_set[8]);

printf("B : %06X S : %06X\n", register\_set[3], register\_set[4]);

printf("T : %06X\n", register\_set[5]);

// bp에서 멈췄으면 "Stop at checkpoint" 출력

if (bpflag)

printf(" Stop at checkpoint[%X]\n", cur->bp);

// 프로그램의 끝이면 "End program" 출력

else {

printf(" End program\n");

register\_set[8] = startaddr;

register\_set[2] = tot\_length;

}

return 1;

}

int target\_addr(int xbpe){

int taraddr = 0;

int disp, abaddr;

// e 값이 0인 경우 (3형식)

if (!(xbpe % 2)){

// 해당 주소에서 displacement 값 가져오기

disp = memory[register\_set[8] + 1] % 16;

disp <<= 8;

disp += memory[register\_set[8] + 2];

// PC 값 3 더해주기 (3형식)

register\_set[8] += 3;

// p 값이 1인 경우 (PC relative)

if (xbpe & 2){

// dispacement가 음수인 경우

if (disp & (1 << 11)){

// 2's complement 거꾸로 진행하여 양수 값 구한다

disp -= 1;

disp = ~disp;

disp = disp & 0xFFF;

// 양수 값을 PC 값에서 뺀 값이 target address

taraddr = register\_set[8] - disp;

}

// displacement가 양수인 경우

else

// PC 값에 displacement를 더해서 target address를 구한다

taraddr = register\_set[8] + disp;

}

// b 값이 1인 경우 (BASE relative)

else if (xbpe & 4){

taraddr = register\_set[3] + disp;

}

// xbpe가 모두 0인 경우(숫자 바로 올리는 경우)

else if (!xbpe){

// dispacement가 숫자값이므로 그대로 target address에 저장

taraddr = disp;

}

}

// e 값이 1인 경우 (4형식)

else {

// 절대 주소이므로 memory에서 그대로 가져온다

abaddr = memory[register\_set[8] + 1] % 16;

abaddr <<= 8;

abaddr += memory[register\_set[8] + 2];

abaddr <<= 8;

abaddr += memory[register\_set[8] + 3];

// PC 값 4 더하기 (4형식)

register\_set[8] += 4;

// memory에 저장된 address값이 그대로 target address가 된다

taraddr = abaddr;

}

// x가 1인 경우 (indexed adressing)

if (xbpe & 8){

// target address에 X register value만큼 더해준다

taraddr += register\_set[1];

}

// target address 반환

return taraddr;

}

int target\_val(int ni, int \*taraddr){

int val;

// n=0, i=1인 경우 (immediate addressing)

if (ni == 1){

// target address 값이 그대로 올라간다

val = \*taraddr;

}

else {

// simple addressing의 경우 (n=1, i=1)

// target address에 해당하는 주소에 저장된 값을 가져온다

val = memory[\*taraddr];

val <<= 8;

val += memory[\*taraddr + 1];

val <<= 8;

val += memory[\*taraddr + 2];

// indirect addressing의 경우 (n=1, i=0)

if (ni == 2){

// target address에 저장된 주소가 새로운 target 주소가 된다

// target address 변경 후 해당 주소에 저장된 값을 가져온다

\*taraddr = val;

val = memory[\*taraddr];

val <<= 8;

val += memory[\*taraddr + 1];

val <<= 8;

val += memory[\*taraddr + 2];

}

}

// target addres value return

return val;

}

void store\_inst(int regnum, int taraddr){

// store instruction

// register에 저장된 값을 메모리에 1byte씩 올린다

memory[taraddr] = (unsigned char)((register\_set[regnum] & 0xFF0000) >> 16);

memory[taraddr + 1] = (unsigned char)((register\_set[regnum] & 0x00FF00) >> 8);

memory[taraddr + 2] = (unsigned char)(register\_set[regnum] & 0x0000FF);

}