과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 3>>

**서강대학교 컴퓨터공학과**

**20171601**

**강주형**

**목 차**

1. 프로그램 개요 4

2. 프로그램 설명 4

2.1 프로그램 흐름도 4

3. 모듈 정의 5

3.1 모듈 이름 : main() 5

3.1.1 기능 5

3.1.2 사용 변수 5

3.2 모듈 이름: check1(char \*origin,char \*\*command, Sym \*\*\*symhead, int \*symbol\_print\_flag, unsigned int \*progaddress,int \*run\_flag, int \*load\_flag, Bp \*bphead, unsigned char \*MEMORY, Op \*\*ophead, int \*EXEC, int \*bp\_state, char \*objectcode,int \*reg) 6

3.2.1 기능 6

3.2.2 사용 변수 6

3.3 모듈이름: check2(char \*origin,char \*\*command, int TOKEN\_SIZE, Sym \*\*\*symhead,int \*symbol\_print\_flag, unsigned int \*progaddress,int \*load\_flag,Bp \*bphead) 6

3.3.1 기능 6

3.3.2 사용변수 6

3.4 모듈이름: progaddr(char \*origin, char \*\*command, unsigned int \*progaddress) 7

3.4.1 기능 7

3.4.2 사용변수 7

3.5 모듈이름: loader(char \*origin, char \*\*command, unsigned int csaddr, unsigned char \*MEMORY, int TOKEN\_SIZE, int \*EXEC) 7

3.5.1 기능 7

3.5.2 사용변수 7

3.6 모듈이름: bp(char \*origin, char \*\*command, Bp \*\*head, char \*index) 7

3.6.1 기능 7

3.6.2 사용변수 7

3.7 모듈이름: bp\_clear(char \*origin,char \*\*command,Bp \*\*head)7

3.7.1 기능 7

3.7.2 사용변수 8

3.8 모듈이름: bp\_print(Bp \*head) 8

3.8.1 기능 8

3.8.2 사용변수 8

3.9 모듈이름:run(int progaddr, int \*run\_flag, int load\_flag, Bp \*bphead, unsigned char \*MEMORY, Op \*\*ophead, int \*EXEC, int \*bp\_state, char\*objectcode,int \*reg) 8

3.9.1 기능 8

3.9.2 사용변수 8

3.10 모듈이름: print\_loadmap(ESTAB \*head) 8

3.10.1 기능 8

3.10.2 사용변수 8

3.11 모듈이름: print\_reg(int \*reg, int point, int end\_flag) 8

3.11.1 기능 8

3.11.2 사용변수 8

3.12 모듈이름: estab\_insert(ESTAB \*\*head, Estab \*\*tail, char \*name, char \*length, char \*address, unsigned int csaddr) 9

3.12.1 기능 9

3.12.2 사용변수 9

3.13 모듈이름:find\_estab(Estab \*head, char \*name) 9

3.13.1 기능 9

3.13.3 사용변수 9

4. 전역 변수 정의 9

4.1 Node \*head 9

4.2 Op \*ophead[20] 9

4.3 unsigned char MEMORY[MEMORY\_SIZE] 9

4.4 unsigned int address 9

5. 코드 9

5.1 20171601.h 9

5.2 20171601.c 11

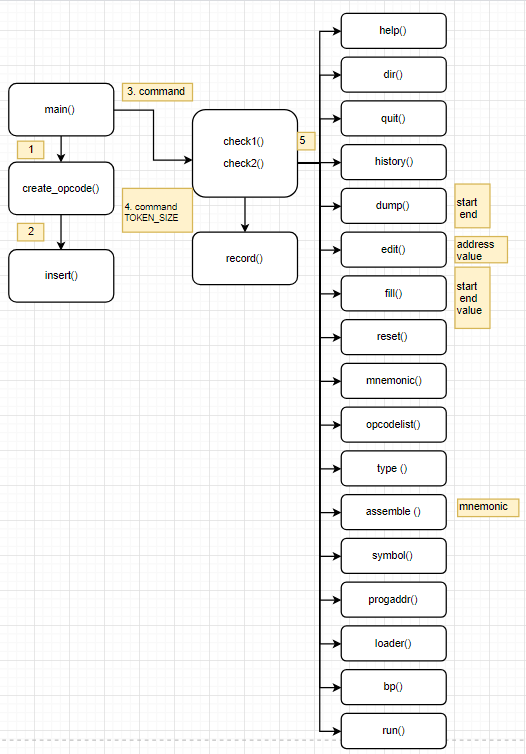
1. **프로그램 개요**

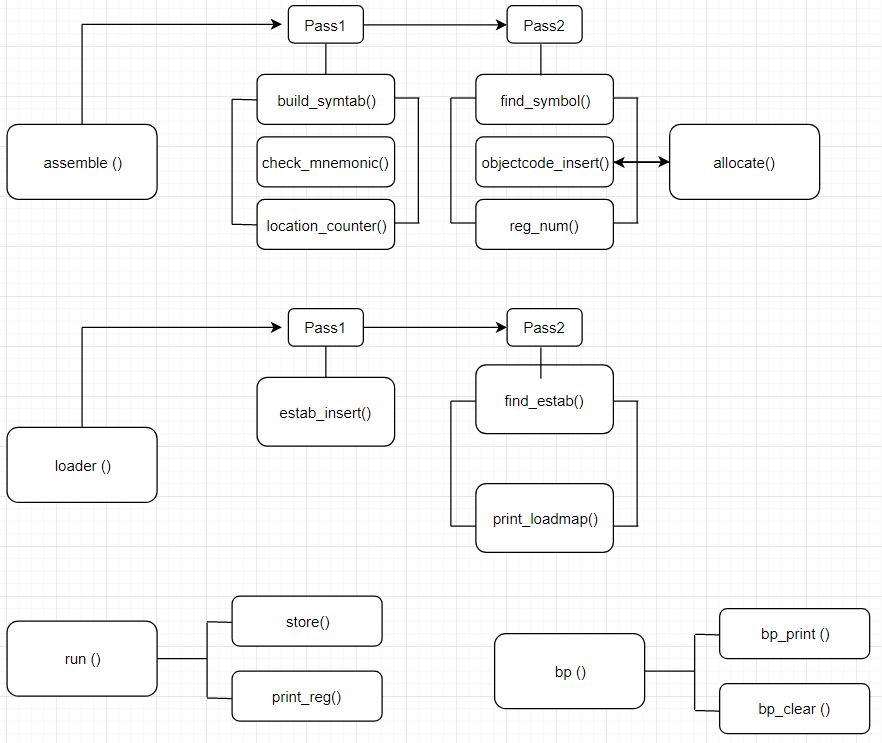
이 프로그램은 프로젝트 1과 2에서 개발한 쉘 프로그램에 추가적인 기능을 더할 것이다. 이 프로젝트에서 한 개 또는 여러 개의 오브젝트 프로그램을 읽고 파일을 서로 연결하여 지정된 위치에 있는 메모리에 로드 할 것이다. 또한 메모리에 로드 된 프로그램을 실행할 것이다.

명령어는 크게 셸 관련 명령어(help, dir, quit, history, type)와 메모리공간 관련 명령어(dump, edit, fill, reset), opcode와 관련된 명령어(opcodelist, opcode mnemonic), SIC/XE 어셈블러 관련 명령어 (assemble, symbol)가 있다.

이번 3차 프로젝트에서는 오브젝트 코드 파일을 link하여 excutable image로 변환하는 linker와 실제 메모리에 object code들을 적재하는 loader의 임무수행을 위한 명령어(progaddr, bp, loader, run)이 추가되었다.

1. **프로그램 설명**
   1. **프로그램 흐름도**



****

**그림 1> 프로그램 흐름도**

1. **모듈 정의**
   1. **모듈 이름 : main()**
      1. *기능*

unix shell과 유사한 입력 프롬프트 상태(sicsim> )를 출력하며 사용자로부터 직접 명령어를 입력받으며 quit 명령어가 입력되기 전까지 계속해서 입력받는다. 또한 입력받은 명령어를 공백과 탭을 기준으로 TOKEN화 하여 해당 명령어 수행을 위해 command\_check 모듈로 넘겨주고 return값에 따라 프로그램을 종료할지, history에 저장할지 결정하는 모듈

* + 1. *사용 변수*

char command[INPUT\_SIZE] : 입력받은 명령어를 저장할 배열

char command\_temp[INPUT\_SIZE] : 입력받은 명령어를 TOKEN화 하기 위해 임시로 command를 저장할 배열

char \*\*command\_token : 입력받은 명령어를 TOKEN으로 만들어 저장할 배열

char \*ptr : TOKEN화 하면서 자를 위치를 가리킬 포인터

int flag : quit명령어를 통해 프로그램의 종료를 알리기 위해 true, false 값을 저장할 변수

int TOKEN\_SIZE : 입력받은 명령어의 TOKEN의 개수를 저장할 변수

int symbol\_print\_flag : symbol 명령어 수행 시 출력을 결정을 알리기 위해 true, false값을 저장할 변수

unsigned int address : dump 명령어 수행시 최종 출력된 address값을 저장하기 위한 변수

unsigned char MEMORY[MEMORY\_SIZE] : 셸에서 관리하는 1MB의 메모리를 담는 변수

int i : 반복문 수행을 위한 변수

NODE \*head, \*tail : history에 담긴 node들의 처음과 끝을 가리킬 포인터

OPCODE \*ophead[20] : opcode table의 각 인덱스의 헤드포인터를 담은 구조체

SYMBOL \*symhead : symbol table의 각 인덱스의 헤드포인터를 담은 구조체

int EXEC : 프로그램 로드 시 최초 수행가능한 위치를 저장하기 위한 변수

int run\_flag : run 명령어 수행시 초기화를 위한 flag

int bp\_state : run 명령어 수행중인지 알려주는 state 변수

int \*reg : run 명령어 수행시 register 값을 저장할 배열

unsigned int progaddr : 프로그램 시작 주소 지정을 위한 변수

int load\_flag : load명령어 성공을 알리기 위한 flag

BP \*bphead : break\_point 정보를 담기 위한 구조체

* 1. **모듈 이름: check1(char \*origin,char \*\*command, Sym \*\*\*symhead, int \*symbol\_print\_flag, unsigned int \*progaddress,int \*run\_flag, int \*load\_flag, Bp \*bphead, unsigned char \*MEMORY, Op \*\*ophead, int \*EXEC, int \*bp\_state, char \*objectcode,int \*reg)**
     1. *기능*

Main 모듈에서 넘겨받은 명령어를 TOKEN\_SIZE기준으로 나누어 프로그램에 내장된 명령어들과 동일한 명령어를 입력 받았는지 확인하여, 올바른 명령어를 입력 받았다면 해당명령어를 수행하는 모듈을 수행할 수 있도록 해당 모듈을 부르며, 잘못된 명령어의 경우 에러메시지를 출력하는 모듈 위의 모듈은 TOKEN\_SIZE가 1인 경우이다.

* + 1. *사용 변수*

unsigned int start – dump와 fill 명령어 수행 시 시작번지를 저장할 변수

unsigned int end – dump와 fill 명령어 수행 시 끝번지를 저장할 변수

* 1. **모듈이름: check2(char \*origin,char \*\*command, int TOKEN\_SIZE, Sym \*\*\*symhead,int \*symbol\_print\_flag, unsigned int \*progaddress,int \*load\_flag,Bp \*bphead)**
     1. *기능*

Main 모듈에서 넘겨받은 명령어를 TOKEN\_SIZE기준으로 나누어 프로그램에 내장된 명령어들과 동일한 명령어를 입력 받았는지 확인하여, 올바른 명령어를 입력 받았다면 해당명령어를 수행하는 모듈을 수행할 수 있도록 해당 모듈을 부르며, 잘못된 명령어의 경우 에러메시지를 출력하는 모듈 위의 모듈은 TOKEN\_SIZE가 2 이상인 경우이다.

* + 1. *사용변수*

unsigned int start – dump와 fill 명령어 수행 시 시작번지를 저장할 변수

unsigned int end – dump와 fill 명령어 수행 시 끝번지를 저장할 변수

unsigned int target – edit 명령어 수행 시 타겟번지를 저장할 변수

unsigned int data – edit와 fill 명령어 수행 시 입력할 data를 저장할 변수

unsigned int trash – 정해진 parameter 이외에 값이 들어올 경우를 체크할 변수

int comma\_num – 입력받아야 할 parameter가 존재하는 명령어들에 대하여 comma 개수를 담을 변수

int i – 반복문 수행을 위한 변수

* 1. **모듈이름: progaddr(char \*origin, char \*\*command, unsigned int \*progaddress)**
     1. *기능*

사용자가 입력한 progaddr의 주소값, 즉 프로그램의 적재위치를 저장하기 위한 함수이다.

* + 1. *사용변수*

없음

* 1. **모듈이름: loader(char \*origin, char \*\*command, unsigned int csaddr, unsigned char \*MEMORY, int TOKEN\_SIZE, int \*EXEC)**
     1. *기능*

사용자가 입력한 object filename에 대하여 현재 디렉토리 내에서 찾아서 linking하여 사용자가 지정한 progaddr의 주소부터 메모리에 해당 object code들을 적재한다. Progaddr을 지정하지 않을시 초기값은 0임에 유의한다

* + 1. *사용변수*

Int i, j, k – 반복문 수행을 위한 변수

Int en\_snum, obj\_num – object file과 external symbol의 개수를 저장할 변수

Int error\_flag – linking loading중 error 유무를 알리기 위한 변수

Int addr – 현재 처리중인 object code가 load될 주소를 담고 있는 변수

Int index – extref 배열에 address저장을 위해 index 정보를 담고 있는 변수

Int value – 메모리에 적재시 1바이트씩 저장하기 위해 사용하는 변수

Int half – Modification Record 수행시 하프바이트를 임시적으로 담고 있는 변수

Int extref[11] – external symbol reference 값을 담고 있는 배열

Int exec – 최초 수행을 시작할 excutable 위치정보를 담고 있는 변수

Unsigned int clength – control section의 length 정보를 담고 있는 변수

FILE \*fp – load할 FILE OPEN을 위한 파일 포인터

Char temp[255] - FILE에서 한 줄씩 저장할 버퍼

Char name[10] – 이름 정보를 저장할 배열

Char length[10] – 길이 정보를 저장할 배열

Char address[10] – 주소 정보를 저장할 배열

Char data[2] – 한 바이트 정보를 저장할 배열

Char mod[10] – 값->변수, 변수->값 변환시 임시적인 string을 담을 배열

Char op - operator의 정보를 저장할 변수

ESTAB \*head, \*tail – EXTERNAL SYMBOL TABLE의 HEAD, TAIL 포인터

* 1. **모듈이름: bp(char \*origin, char \*\*command, Bp \*\*head, char \*index)**
     1. *기능*

bp 명령어 수행 시 사용자가 입력한 address정보의 에러 유무를 확인 후 저장하는 함수

* + 1. *사용변수*

BP \*temp, \*ptr, \*follow – Insertion Sort를 위한 링크드 리스트 포인터

* 1. **모듈이름: bp\_clear(char \*origin,char \*\*command,Bp \*\*head)**

*3.7.1 기능*

breakpoint들을 모두 지우는 함수

3.7.2 *사용변수*

BP \*temp – breakpoint 삭제를 위한 링크드 리스트 포인터

* 1. **모듈이름: bp\_print(Bp \*head)**
     1. *기능*

저장된 breakpoint들을 출력하는 함수

* + 1. *사용변수*

BP \*temp – bp 출력을 위한 링크드 리스트 포인터

## 모듈이름: store(int reg, unsigned char \*MEMORY, int ta)

*3.9.1 기능*

command\_run()명령어 수행시 STORE instruction 명령어 수행을 위한 함수

*3.9.2 사용변수*

없음

* 1. **모듈이름: run(int progaddr, int \*run\_flag, int load\_flag, Bp \*bphead, unsigned char \*MEMORY, Op \*\*ophead, int \*EXEC, int \*bp\_state, char \*objectcode, int \*reg)**
     1. *기능*

loader에 의해 프로그램 메모리에 적재된 object code들을 실제로 디버깅 하는 함수

* + 1. *사용변수*

Int start – bp check를 위한 instruction의 시작 주소 정보

Int end – bp check를 위한 instruction의 끝 주소 정보

Int end\_flag - run명령이 끝남을 알려주기 위한 flag

Int value, n, I, x, b, p, e, opcode, r1, r2, format – parsing을 위한 opcode format 분석정보가 담길 변수

Int state, ta, indirect, data – addressmode parsing후 분석정보가 담길 변수

Char objectcode[10] – memory에 적재된 object code를 읽어오기 위한 string

BP \*bp\_ptr – Break point 탐색을 위한 포인터

OPCODE \*op\_ptr - OPCODE 탐색을 위한 포인터

* 1. **모듈이름 :** print\_loadmap(ESTAB \*head)
     1. *기능*

정상적으로 object file들의 code들이 메모리에 load되었음을 알리기 위해 loadmap을 출력하는 함수

* + 1. *사용변수*

ESTAB \*ptr

Unsigned int tot – 전체 길이를 저장하기 위한 포인터

* 1. **모듈이름:** print\_reg(int \*reg, int point, int end\_flag)
     1. *기능*

Run 명령어 수행시 breakpoint에서 멈췄을 때 register 출력을 위한 함수

* + 1. *사용변수*

없음

* 1. **모듈이름:** estab\_insert(ESTAB \*\*head, ESTAB \*\*tail, char \*name, char \*length, char \*address, unsigned int csaddr)
     1. *기능*

Loader pass1 수행시 external symbol define record에 있는 symbol들을 table로 build하는 함수

* + 1. *사용변수*

ESTAB \*temp – 새로운 node를 할당하여 저장하기 위한 구조체

Int flag – Duplicate External Symbol Error Check를 위한 flag

* 1. **모듈이름:** find\_estab(ESTAB \*head, char \*name)
     1. *기능*

Loader pass2 수행시 external symbol reference를 위해 estab에서 참조하는 함수

* + 1. *사용변수*

ESTAB \*ptr – ESTAB에서 search를 위한 포인터

1. **전역 변수 정의**
   1. **Node \*head**

history에 저장할 linked list의 head의 위치를 가리킬 포인터

* 1. **Op \*ophead[20]**

opcode.txt 파일을 읽어 opcodelist를담을 20개의 사이즈 hash table head 포인터

* 1. **unsigned char MEMORY[MEMORY\_SIZE]**

shell 내에서 사용하는 1MB의 MEMORY의 값을 담을 변수

* 1. **unsigned int address**

dump 명령어에서 start, end 주소 모두가 주어진 경우를 제외하고 출력을 시작할 위치와 출력을 끝낸 후 다음 명령어 수행 시 출력시작 위치를 저장할 변수

1. **코드**

**5.1**  **20171601.h**

/\*정의되는 상수\*/

#define INPUT\_SIZE 500

#define MEMORY\_SIZE 0x100000

/\*history에 사용될 linked list node\*/

typedef struct \_Node {

char a[INPUT\_SIZE];

struct \_Node \*list;

}Node;

typedef struct \_Op {

int num;

char inst[10];

char type[5];

struct \_Op \*list;

}Op;

typedef struct \_Ob{

int first, mid, end, location;

struct \_Ob \*list;

int format, enter\_flag;

char var[255];

}Ob;

typedef struct \_Sym{

int location;

char name[10];

struct \_Sym \*list;

}Sym;

typedef struct \_Estab{

unsigned int address;

unsigned int length;

char name[10];

struct \_Estab\* list;

}Estab;

typedef struct \_Bp{

int point;

struct \_Bp \*list;

}Bp;

/\*함수 원형\*/

void bp\_clear(char \*origin,char \*\*command,Bp \*\*head);

void bp\_print(Bp \*head);

void bp(char \*origin,char \*\*command,Bp \*\*head, char \*index);

void print\_reg(int \*reg, int point, int end\_flag);

void run(int progaddr, int \*run\_flag, int load\_flag, Bp \*bphead, unsigned char \*MEMORY, Op \*\*ophead, int \*EXEC, int \*bp\_state, char \*objectcode, int \*reg);

void store(int reg, unsigned char \*MEMORY, int ta);

void print\_loadmap(Estab \*head);

int find\_estab(Estab \*head, char \*name);

int estab\_insert(Estab\*\*head, Estab \*\*tail,char \*name, char \*length, char \*address, unsigned int csaddr);

void symbol(Sym \*\*symhead, int flag);

Sym \*allocate(Sym \*data);

int check\_mnemonic(char \*mnemonic, char \*format, int \*opcode, Op \*\*ophead);

int build\_symtab(Sym \*\*symhead, Sym \*node, int line\_num);

int find\_symbol(Sym \*\*symhead, char \*symbol);

void location\_counter(char \*format, unsigned int \*location, char \*mnemonic);

void objectcode\_insert(int format, int first, int mid, int end, int location, Ob\*\* head, Ob\*\* tail, char \*var, int enter\_flag);

int reg\_num(char \*reg);

void type(char \*origin, char \*\*command);

void assemble(char \*origin,char \*\*command, Op \*\*ophead, Sym \*\*\*output, int \*symbol\_print\_flag);

void symbol();

void opcodelist();

void mnemonic(char \*origin,char \*\*command);

void create\_opcode();

void insert(Op \*node);

void check1(char \*origin,char \*\*command,Sym \*\*\*symhead, int \*symbol\_print\_flag,unsigned int \*progaddress,int \*run\_flag,int \*load\_flag, Bp \*bphead, unsigned char \*MEMORY, Op \*\*ophead,int \*EXER, int \*bp\_state, char \*objectcode,int \*reg);

int check2(char \*origin,char \*\*command, int TOKEN\_SIZE, Sym \*\*\*symhead, int \*symbol\_print\_flag, unsigned int \*progaddress,int \*load\_flag,Bp \*bphead);

void quit();

void record(char \*origin,char \*\*command, int TOKEN\_SIZE);

void help();

void history();

void dir();

void dump(unsigned int start, unsigned int end);

void edit(unsigned int origin, unsigned int one);

void fill(unsigned int start, unsigned int end, unsigned int one);

void reset();

void progaddr(char \*origin, char \*\*command, unsigned int \*progaddress);

int loader(char \*origin, char \*\*command, unsigned int csaddr, unsigned char \*MEMORY, int TOKEN\_SIZE, int \*EXEC);

**5.2**  **20171601.c : 이번 프로젝트에서 새롭게 추가한 함수들만 기록함**

/\*포함되는 파일\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include "20171601.h"

/\*전역 변수\*/

Node \*head;

Op \*ophead[20];

unsigned char MEMORY[MEMORY\_SIZE];

unsigned int address;

/\*프로그램 시작\*/

int main(void)

{

create\_opcode(); // opcode 생성

char command[INPUT\_SIZE] = { "\0" }; // 입력받는 명령어

char command\_temp[INPUT\_SIZE] = { "\0" }; // 명렁어를 나누기 위한 임의의 배열

char \*\*token; // 나눈 명령들 저장

char \*ptr; // 나눈 덩어리 저장을 위한 변수

Sym \*\*symhead = NULL;

Bp \*bphead = NULL;

int TOKEN\_SIZE,i, symbol\_print\_flag = 0, flag=0;

int run\_flag = 0, EXEC = -1, bp\_state=0;

int \*reg = (int \*)malloc(sizeof(int)\*11);

char objectcode[10];

unsigned int progaddress = 0x00;

while (!flag)

{

TOKEN\_SIZE = 1;

printf("sicsim> ");

fgets(command, sizeof(command), stdin);

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

strcpy(command\_temp, command);

//TOKEN의 갯수 세기

ptr = strtok(command\_temp, " \t\n");

if (!ptr) continue;

while (1)

{

ptr = strtok(NULL, " \t\n");

if (!ptr) break;

TOKEN\_SIZE++;

}

token = (char \*\*)malloc(sizeof(char \*)\*(TOKEN\_SIZE + 1));

//TOKEN 분리

strcpy(command\_temp, command);

ptr = strtok(command\_temp, " \t\n");

if (!ptr) continue;

token[0] = (char \*)malloc(sizeof(char) \* (INPUT\_SIZE));

strcpy(token[0], ptr);

for (int i = 1; i < TOKEN\_SIZE; i++) {

ptr = strtok(NULL, " \t\n");

if (ptr == NULL) break;

token[i] = (char \*)malloc(sizeof(char) \* (INPUT\_SIZE));

strcpy(token[i], ptr);

}

if (strcmp(\*token, "quit") == 0 || strcmp(\*token, "q") == 0) // command가 quit일 경우

{

quit(); // 메모리 해제

flag = 1; // 종료

}

else if (strcmp(\*token, "bp") == 0)

{

if(TOKEN\_SIZE == 1)

{

record(command, token, 1);

bp\_print(bphead);

}

else if(TOKEN\_SIZE == 2)

{

if(!strcmp(token[1],"clear"))

bp\_clear(command, token,&bphead);

else bp(command,token,&bphead, token[1]);

}

else printf("Error : Invalid Parameter!!\n");

}

else

{

if (TOKEN\_SIZE == 1) // token의 갯수가 1개라면

check1(command,token,&symhead,&symbol\_print\_flag,&progaddress,&run\_flag, &load\_flag, bphead,MEMORY,ophead,&EXEC,&bp\_state,objectcode,reg); // check1() 함수로 이동

else if (TOKEN\_SIZE >= 2) // token의 갯수가 2개 이상이라면

check2(command,token, TOKEN\_SIZE, &symhead,&symbol\_print\_flag, &progaddress,&load\_flag,bphead); // check2() 함수로 이동

}

//메모리 해제

for (i = 0; i < TOKEN\_SIZE; i++)

{

free(token[i]);

token[i] = NULL;

}

}

}

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

/\*함수 : check1(char \*origin,char \*\*command, Sym \*\*\*symhead, int \*symbol\_print\_flag, unsigned int \*progaddress,int \*run\_flag, int \*load\_flag, Bp \*bphead, unsigned char \*MEMORY, Op \*\*ophead, int \*EXEC, int \*bp\_state, char \*objectcode,int \*reg)

/\*목적 : 입력받은 명령어를 인식하여 해당 명렁을 수행한다.

/\*리턴값 : 없음

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

//Token의 갯수가 1개인 경우

void check1(char \*origin,char \*\*command, Sym \*\*\*symhead, int \*symbol\_print\_flag, unsigned int \*progaddress,int \*run\_flag, int \*load\_flag, Bp \*bphead, unsigned char \*MEMORY, Op \*\*ophead, int \*EXEC, int \*bp\_state, char \*objectcode,int \*reg)

{

unsigned int start, end = 0;

// command가 help인 경우

if (strcmp(\*command, "help") == 0 || strcmp(\*command, "h") == 0)

{

record(origin,command, 1); // history에 기록

help();

}

// command가 dir인 경우

else if (strcmp(\*command, "dir") == 0 || strcmp(\*command, "d") == 0)

{

record(origin,command, 1);

dir();

}

//command가 history인 경우

else if (strcmp(\*command, "history") == 0 || strcmp(\*command, "hi") == 0)

{

record(origin,command, 1);

history();

}

//dump : case 1(no argumennt)

else if (strcmp(\*command, "dump") == 0 || strcmp(\*command, "du") == 0)

{

record(origin,command, 1);

//(last address + 1)+159가 범위를 초과할 경우

if ((address + 160) > 0xFFFFF)

{

//last address=를 0xfffff로 제한 & 다음 주소는 0x00000로지정

start = address;

end = 0xFFFFF;

address = 0;

}

// 범위를 초가하지 않는 경우 start, end address 지정

else

{

start = address;

end = address + 159;

address = end + 1;

}

dump(start, end);

}

//command가 reset인 경우

else if (strcmp(\*command, "reset") == 0)

{

record(origin,command, 1);

reset();

}

// command가 opcodelist인 경우

else if (strcmp(\*command, "opcodelist") == 0)

{

record(origin,command, 1);

opcodelist();

}

else if (strcmp(\*command, "symbol") == 0)

{

record(origin, command, 1);

symbol(\*symhead, \*symbol\_print\_flag);

}

else if (strcmp(\*command, "run") == 0)

{

record(origin,command,1);

run(\*progaddress, run\_flag, \*load\_flag, bphead, MEMORY, ophead, EXEC, bp\_state, objectcode,reg);

}

// 그 외의 명령어가 들어오면 에러 출력

else printf("Error : Invalid Parameter!!\n");

}

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

/\*함수 : check2(char \*origin,char \*\*command, int TOKEN\_SIZE, Sym \*\*\*symhead,int \*symbol\_print\_flag, unsigned int \*progaddress,int \*load\_flag,Bp \*bphead)

/\*목적 : 입력받은 명령어를 인식하여 해당 명렁을 수행한다.

/\*리턴값 : 없음

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

//TOKEN\_SIZE가 2개 이상인 경우

int check2(char \*origin,char \*\*command, int TOKEN\_SIZE, Sym \*\*\*symhead,int \*symbol\_print\_flag, unsigned int \*progaddress,int \*load\_flag,Bp \*bphead)

{

unsigned int start, end, addr, new = 0,trash = 0;

int i, count = 0,EXEC = -1;

if ((strcmp(\*command, "dump") == 0 || strcmp(\*command, "du") == 0) && (TOKEN\_SIZE <=4))

{

// dump : case 1(1 argument)

if ((TOKEN\_SIZE == 2) && (strchr(command[1],',') == NULL))

{

// start parameter 조건

for(i= 0; i < (int)strlen(command[1]); i++)

{

if((command[1][i] >= '0' && command[1][i] <= '9') ||(command[1][i] >= 'a' && command[1][i] <= 'f') ||

(command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ','|| (command[1][0] == '0' && strtol(command[1],NULL,16) >= 0x00000 && strtol(command[1],NULL,16) <= 0xfffff))

continue;

printf("Error : Invalid Parameter!!\n"); // dump XYZ

return 0;

}

//조건 만족하면 변수에 저장

start = strtol(command[1],NULL,16);

// Boundary check

if(start <= 0xfffff && (int)start >= 0x00000)

{

if((start + 160) > 0xfffff)

{

end = 0xfffff;

address = 0;

}

else

{

end = start + 159;

address = end +1;

}

dump(start, end);

record(origin,command, 2);

}

else printf("Error : Invalid Parameter!!\n");

}

// token이 2개 이상일 경우

else

{

for(i = 1; i < TOKEN\_SIZE - 1; i++)

{

strcat(command[1], " ");

strcat(command[1], command[i + 1]);

}

for(i = 0; i < (int)strlen(command[1]); i++)

if(command[1][i] == ',') count++;

//컴마의 갯수가 2개 이상이면 에러메세지 출력

if(count >= 2) printf("Error : Invalid Parameter!!\n");

else

{

i = sscanf(command[1], "%x , %x %x", &start, &end,&trash);

// start와 end를 읽었다면

if(i == 2)

{

// Parameter Check

for(i = 0; i < (int)strlen(command[1]); i++)

{

if((command[1][i] >= '0' && command[1][i] <= '9') || (command[1][i] >= 'a' && command[1][i] <= 'f') ||

(command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ',' || command[1][i] == ' '||((int)start>=0x00000 &&start<=0xfffff && end<=0xfffff && (int)end >= 0x00000))

continue;

printf("Error : Invalid Parameter!!\n");

return 0;

}

// Boundary Check

if(start <= end)

{

if(((int)start >= 0 &&start <= 0xfffff) && ((int)end >= 0 && end <= 0xfffff))

{

dump(start, end);

record(origin,command, 2);

if(end == 0xfffff) address = 0;

else address = end + 1;

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

}

}

// command가 인 경우

else if (strcmp(\*command, "edit") == 0 || strcmp(\*command, "e") == 0)

{

//Parameter Check

if (TOKEN\_SIZE == 2 && strchr(command[1],',') == NULL)

{

printf("Error : Invalid Parameter!!\n");

return 0;

}

else

{

for(i = 1; i < TOKEN\_SIZE - 1; i++)

{

strcat(command[1], " ");

strcat(command[1], command[i + 1]);

}

for(i = 0; i < (int)strlen(command[1]); i++)

if(command[1][i] == ',') count++;

// 컴마 갯수 Check

if(count >= 2) printf("Error : Invalid Parameter!!\n");

else

{

i = sscanf(command[1], "%x , %x %x", &addr, &new,&trash);

if(i == 2)

{

//Parameter Chech

for(i = 0; i < (int)strlen(command[1]); i++)

{

if((command[1][i] >= '0' && command[1][i] <= '9') || (command[1][i] >= 'a' && command[1][i] <= 'f') ||

(command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ',' || command[1][i] == ' ' || ((int)addr >= 0x00000 && addr<=0xfffff))

continue;

printf("Error : Invalid Parameter!!\n");

return 0;

}

// Boundary Check

if((int)addr >= 0x00000 && addr <= 0xFFFFF)

{

if((int)new >= 0x00 && new <= 0xFF)

{

edit(addr, new);

record(origin,command, TOKEN\_SIZE);

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

}

}

// command가 fill인 경우

else if (strcmp(\*command, "fill") == 0 || strcmp(\*command, "f") == 0)

{

for(i = 1; i < TOKEN\_SIZE - 1; i++)

{

strcat(command[1], " ");

strcat(command[1], command[i + 1]);

}

for(i = 0; i < (int)strlen(command[1]); i++)

if(command[1][i] == ',') count++;

if(count != 2) printf("Error : Invalid Parameter!!\n");

else

{

i = sscanf(command[1], "%x , %x , %x %x", &start, &end,&new,&trash);

if(i == 3)

{

for(i = 0; i < (int)strlen(command[1]); i++)

{

if((command[1][i] >= '0' && command[1][i] <= '9') || (command[1][i] >= 'a' && command[1][i] <= 'f') ||

(command[1][i] >= 'A' && command[1][i] <= 'F') ||command[1][i] == ',' || command[1][i] == ' ' || ((int)start >= 0x00000 && start <= 0xfffff && (int)end >=0x00000 && end <= 0xfffff))

continue;

printf("Error : Invalid Parameter!!\n");

return 0;

}

if((start <= end)&&((int)start >= 0x00000 && start <= 0xFFFFF)&&(((int)end >= 0x00000 && end <= 0xFFFFF)))

{

if((int)new >= 0x00 && new <= 0xFF)

{

fill(start, end, new);

record(origin,command, TOKEN\_SIZE);

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

else printf("Error : Invalid Parameter!!\n");

}

}

else if (strcmp(\*command, "opcode") == 0)

mnemonic(origin,command);

else if (strcmp(\*command, "type") == 0)

{

if(TOKEN\_SIZE == 2)

type(origin, command);

else

{

printf("Error : Invalid Parameter!!\n");

return 0;

}

}

else if (strcmp(\*command, "assemble") == 0)

{

if(TOKEN\_SIZE == 2)

assemble(origin, command, ophead, symhead, symbol\_print\_flag);

else

{

printf("Error : Invalid Parameter!!\n");

return 0;

}

}

else if (strcmp(\*command, "progaddr") == 0)

{

if(TOKEN\_SIZE == 2)

progaddr(origin, command, progaddress);

else

{

printf("Error : Invalid Parameter!!\n");

return 0;

}

}

else if (strcmp(\*command, "loader") == 0)

{

if(TOKEN\_SIZE <= 4)

{

\*load\_flag = loader(origin, command, \*progaddress, MEMORY, TOKEN\_SIZE, &EXEC);

return \*load\_flag;

}

else

{

printf("Error : Invalid Parameter!!\n");

return 0;

}

}

else if (strcmp(\*command, "bp") == 0)

{

if(strcmp(command[1],"clear") == 0)

{

bp\_clear(origin,command,&bphead);

}

else bp(origin,command,&bphead, command[1]);

}

//그 외 명령어가 들어올 경우 에러메세지 출력

else printf("Error : Invalid Parameter!!\n");

return 0;

}

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

/\*함수 : progaddr(char \*origin, char \*\*command, unsigned int \*progaddress)

/\*목적 : 사용자가 입력한 progaddr의 주소값, 즉 프로그램의 적재위치를 저장한다.

/\*리턴값 : 없음

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

void progaddr(char \*origin, char \*\*command, unsigned int \*progaddress)

{

sscanf(command[1], "%x", progaddress);

if(\*progaddress > 0xFFFFF)

{

printf("Error : Invalid Parameter!!\n");

return;

}

else printf("Program starting address set to %#x.\n",\*progaddress);

record(origin, command, 2);

return;

}

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

/\*함수 : loader(char \*origin, char \*\*command, unsigned int csaddr, unsigned char \*MEMORY, int TOKEN\_SIZE, int \*EXEC)

/\*목적 : 사용자가 입력한 object filename에 대하여 현재 디렉토리 내에서 찾아서 linking하여 사용자가 지정한 progaddr의 주소부터 메모리에 해당 object code들을 적재한다.

/\*리턴값 : load\_flag의 값

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

int loader(char \*origin, char \*\*command, unsigned int csaddr, unsigned char \*MEMORY, int TOKEN\_SIZE, int \*EXEC)

{

int i, j, k, es\_num, obj\_num, error\_flag, addr, index, value, half, extref[11], exec;

unsigned int clength;

FILE \*fp;

char temp[255], name[10], length[10], address[10], data[2], mod[10], op;

Estab \*head = NULL, \*tail = NULL;

//the number of object files

obj\_num = TOKEN\_SIZE - 1;

//pass1

for(i = 0; i < obj\_num; i++)

{

fp = fopen(command[i + 1], "rt");

//Empty Check

if(!fp)

{

printf("There is no %s file\n", command[i + 1]);

return 0;

}

//Parsing

while(fgets(temp, sizeof(temp), fp))

{

//Head Record

if(temp[0] == 'H')

{

strncpy(name, temp + 1, 6);

strncpy(address, temp + 7, 6);

strncpy(length, temp + 13, 6);

name[6] = address[6] = length[6] = '\0';

error\_flag = estab\_insert(&head, &tail, name, length, address, csaddr);

if(error\_flag) return 0;

clength = tail->length;

}

//Define Record

else if(temp[0] == 'D')

{

//Count The Number Of External Symbol

es\_num = strlen(temp + 1) / 12;

for(j = 0; j < es\_num; j++)

{

strncpy(name, temp + 1 + (j \* 12), 6);

strncpy(address, temp + 7 + (j \* 12), 6);

strcpy(length, "000000");

name[6] = address[6] = length[6] = '\0';

error\_flag = estab\_insert(&head, &tail, name, length, address, csaddr);

if(error\_flag) return 0;

}

}

//Ignored Record

else if(temp[0] == 'R');

else if(temp[0] == 'T');

else if(temp[0] == 'M');

else if(temp[0] == 'E');

else if(temp[0] == '.');

else

{

printf("Error\n");

return 0;

}

strcpy(name, "");

strcpy(address, "");

strcpy(length, "");

strcpy(temp, "");

}

csaddr += clength;

fclose(fp);

}

//Check Memory Bound

if(csaddr >= 0x1000000)

{

printf("MEMORY Boundary Error\n");

return 0;

}

//pass2

for(i = 0; i < obj\_num; i++)

{

fp = fopen(command[i + 1], "rt");

//Parsing

while(fgets(temp, sizeof(temp), fp))

{

//Head Record

if(temp[0] == 'H')

{

strncpy(name, temp + 1, 6);

name[6] = '\0';

extref[1] = addr = find\_estab(head, name);

//Store Control Section Address

if(addr == -1) return 0;

}

//Reference Record

else if(temp[0] == 'R')

{

//Check The Number of External Symbol

es\_num = strlen(temp + 1) / 8;

if((strlen(temp + 1) % 8) > 2) es\_num++;

for(j = 0; j < es\_num; j++)

{

strncpy(name, temp + 3 + (j \* 8), 6);

if(name[strlen(name) - 1] == '\n')

{

for(k = strlen(name) - 1; k < 6; k++) name[k] = ' ';

name[k] = '\0';

}

//Store Address Of External Symbol In Array

extref[2 + j] = find\_estab(head, name);

if(extref[2 + j] == -1)

{

printf("Undefined External Symbol Error\n");

return 0;

}

}

}

//Text Record

else if(temp[0] == 'T')

{

strncpy(address, temp + 1, 6);

strncpy(length, temp + 7, 2);

address[6] = length[2] = '\0';

sscanf(address, "%x", &addr);

sscanf(length, "%x", &clength);

for(j = 0; j < (int)clength; j++)

{

strncpy(data, temp + 9 + (2 \* j), 2);

data[2] = '\0';

sscanf(data, "%x", &value);

MEMORY[addr + j + extref[1]] = (unsigned char)value;

}

}

//Modification Record

else if(temp[0] == 'M')

{

if(strlen(temp)>10)

{

strncpy(address, temp + 1, 6);

strncpy(length, temp + 7, 2);

address[6] = length[2] = '\0';

op = temp[9];

sscanf(temp + 10, "%d", &index);

//Check Reference Error

if(index > es\_num + 1)

{

printf("External Symbol Reference Error\n");

return 0;

}

//Character -> Value

sscanf(address, "%x", &addr);

sscanf(length, "%x", &clength);

sprintf(mod, "%02X%02X%02X", MEMORY[addr + extref[1]], MEMORY[addr + extref[1] + 1], MEMORY[addr + extref[1] + 2]);

sscanf(mod, "%x", &value);

if(clength == 5) half = value & 0x00F00000;

// '+' / '-' check -> Modification

if(op == '+') value += extref[index];

else if(op == '-') value -= extref[index];

else

{

printf("Operator Error\n");

return 0;

}

//Carry Bit Check

if(clength == 5)

{

value = value & 0xFFFFF;

value = half + value;

}

else if(clength == 6) value = value & 0xFFFFFF;

//Value -> Character

sprintf(mod, "%06X", value);

for(j = 0; j < 3; j++)

{

strncpy(data, mod + (2 \* j), 2);

data[2] = '\0';

sscanf(data, "%x", &value);

MEMORY[addr + extref[1] + j] = (unsigned char)value;

}

}

else continue;

}

//Ignored Record

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

{

//Executable Address

if(strlen(temp) > 3)

{

strncpy(data, temp + 1, 6);

sscanf(data, "%06X", &exec);

\*EXEC = extref[1] + exec;

}

}

else if(temp[0] == 'D');

else if(temp[0] == '.');

else return 0;

strcpy(temp, "");

strcpy(name, "");

strcpy(address, "");

strcpy(length, "");

}

fclose(fp);

}

record(origin, command, TOKEN\_SIZE);

//Success

print\_loadmap(head);

free(head);

return 2;

}

void print\_loadmap(Estab \*head)

{

Estab \*temp = head;

unsigned int tot = 0;

printf("control\t\tsymbol\t\taddress\t\tlength\n");

printf("section\t\tname\n");

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

while(temp)

{

if(temp->length)

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

else

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

tot += temp->length;

temp = temp->list;

}

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

printf("\t\t\t\ttotal length\t%04X\n",tot);

}

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

/\*함수 : bp(char \*origin, char \*\*command, Bp \*\*head, char \*index)

/\*목적 : bp 명령어 수행 시 사용자가 입력한 address정보의 에러 유무를 확인 후 저장한다.

/\*리턴값 : 없음

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

void bp(char \*origin, char \*\*command, Bp \*\*head, char \*index)

{

Bp \*temp, \*ptr, \*follow;

temp = (Bp\*)malloc(sizeof(Bp));

sscanf(index, "%x", &temp->point);

temp->list = NULL;

if(!(\*head)) \*head = temp;

else

{

ptr = \*head;

//Insertion Sort

while(ptr && ptr->point <= temp->point)

{

if(ptr->point == temp->point)

{

printf("Duplicate Break Point\n");

return;

}

follow = ptr;

ptr = ptr->list;

}

temp->list = ptr;

if(ptr == \*head) \*head = temp;

else follow->list = temp;

}

//Success

record(origin, command, 1);

printf("[ok] create breakpoint %s\n",index);

}

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

/\*함수 : bp\_clear(char \*origin,char \*\*command,Bp \*\*head)

/\*목적 : command\_bp()함수를 통해 저장된 모든 Break Points를 free하는 함수

/\*리턴값 : 없음

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

void bp\_clear(char \*origin,char \*\*command,Bp \*\*head)

{

Bp \*temp;

while(\*head)

{

temp = \*head;

(\*head) = (\*head)->list;

free(temp);

}

record(origin, command, 2);

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

}

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

/\*함수 : bp\_print(Bp \*head)

/\*목적 : breakpoint를 출력하는 함수

/\*리턴값 : 없음

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

void bp\_print(Bp \*head)

{

Bp \*temp = head;

if(!temp)

{

printf("no breakpoints set.\n");

return;

}

printf("breakpoints\n");

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

while(temp)

{

printf("%04X\n",temp->point);

temp=temp->list;

}

}

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

/\*함수 : store(int reg, unsigned char \*MEMORY, int ta)

/\*목적 : run()명령어 수행시 STORE instruction 명령어 수행을 위한 함수

/\*리턴값 : 없음

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

//Store instruction function

void store(int reg, unsigned char \*MEMORY, int ta)

{

MEMORY[ta] = (unsigned char)((reg & 0xFF0000) / 0x010000);

MEMORY[ta + 1] = (unsigned char)((reg & 0x00FF00) / 0x000100);

MEMORY[ta + 2] = (unsigned char)((reg & 0x0000FF) / 0x000001);

}

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

/\*함수 : run(int progaddr, int \*run\_flag, int load\_flag, Bp \*bphead, unsigned char \*MEMORY, Op \*\*ophead, int \*EXEC, int \*bp\_state, char \*objectcode, int \*reg)

/\*목적 : loader()에 의해 프로그램 메모리에 적재된 object code들을 실제로 디버깅 하는 함수

/\*리턴값 : 없음

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

void run(int progaddr, int \*run\_flag, int load\_flag, Bp \*bphead, unsigned char \*MEMORY, Op \*\*ophead, int \*EXEC, int \*bp\_state, char \*objectcode, int \*reg)

{

int start, end, end\_flag = 0;

int value, n, i, x, b, p, e, opcode, r1, r2, k, format;

int state = -1, ta, data, indirect;

Bp \*bp\_ptr = bphead;

Op \*op\_ptr;

//Check Exist Loaded Object Files

if(!load\_flag)

{

printf("There is no loaded object files\n");

return ;

}

//Excutable Position Allocate

if(\*EXEC == -1) \*EXEC = progaddr;

// 0 -> A

// 1 -> X

// 2 -> L

// 3 -> B

// 4 -> S

// 5 -> T

// 6 -> F

// 8 -> PC

// 9 -> SW

//Initialize

if(!(\*run\_flag))

{

for(k = 0; k < 11; k++)

reg[k] = 0;

reg[8] = \*EXEC;

reg[2] = 0xFFFFFF;

\*run\_flag = 1;

}

//Run

while(reg[8] < 0x100000)

{

format = 0;

//Parsing

if(!(\*bp\_state))

{

sprintf(objectcode, "%02X%02X%02X", MEMORY[reg[8]], MEMORY[reg[8] + 1], MEMORY[reg[8] + 2]);

objectcode[6] = '\0';

sscanf(objectcode, "%06X", &reg[10]);

}

opcode = (value & 0xFC0000) / 0x010000;

start = reg[8];

end = reg[8] + 2;

if((value & 0x001000) / 0x001000 == 1) end++;

//Search opcode

for(k = 0; k < 20; k++)

{

op\_ptr = ophead[k];

while(op\_ptr)

{

if(op\_ptr->num == opcode)

{

//Search format

if(op\_ptr->type[0] == '1') format = 1;

else if(op\_ptr->type[0] == '2') format = 2;

else if(op\_ptr->type[0] == '3') format = 3;

break;

}

op\_ptr = op\_ptr->list;

}

}

//BreakPoint Check

if(!(\*bp\_state))

{

while(bp\_ptr)

{

if(bp\_ptr->point >= start && bp\_ptr->point <= end)

{

print\_reg(reg, bp\_ptr->point, end\_flag);

\*bp\_state = 1;

return ;

}

bp\_ptr = bp\_ptr->list;

}

bp\_ptr = bphead;

}

else \*bp\_state = 0;

if(format == 0)

{

printf("OPCODE NOT FOUND\n");

return ;

}

//format 1

else if(format == 1)

{

reg[8]++;

}

//format 2

else if(format == 2)

{

r1 = (value & 0x00F000) / 0x001000;

r2 = (value & 0x000F00) / 0x000100;

//CLEAR

if(opcode == 0xB4) reg[r1] = 0;

//COMPR

else if(opcode == 0xA0)

{

if(reg[r1] == reg[r2]) state = 0; //EQUAL

else if(reg[r1] > reg[r2]) state = 1; //GREAT

else if(reg[r1] < reg[r2]) state = 2; //LESS

}

//TIXR

else if(opcode == 0xB8)

{

reg[1]++;

if(reg[1] == reg[r1]) state = 0; //EQUAL

else if(reg[1] > reg[r1]) state = 1; //GREAT

else if(reg[1] < reg[r1]) state = 2; //LESS

}

//ADDR

else if(opcode == 0x90) reg[r2] = reg[r1] + reg[r2];

//DIVR

else if(opcode == 0x9C) reg[r2] = reg[r2] / reg[r1];

//MULTR

else if(opcode == 0x98) reg[r2] = reg[r2] \* reg[r1];

//SUBR

else if(opcode == 0x94) reg[r2] = reg[r2] - reg[r1];

else continue;

//PC register update

reg[8] += 2;

}

//format 3/4

else if(format == 3)

{

n = (value & 0x020000) / 0x020000;

i = (value & 0x010000) / 0x010000;

x = (value & 0x008000) / 0x008000;

b = (value & 0x004000) / 0x004000;

p = (value & 0x002000) / 0x002000;

e = (value & 0x001000) / 0x001000;

ta = value & 0x000FFF;

reg[8] += 3;

//format 4

if(e == 1)

{

format = 4;

sprintf(objectcode, "%s%02X", objectcode, MEMORY[reg[8]]);

objectcode[8] = '\0';

sscanf(objectcode, "%08X", &value);

reg[8] += 1;

ta = value & 0x000FFFFF;

}

//relative mode

if(b == 1)

{

ta += reg[3];

ta = ta & 0xFFFFF;

}

else if(p == 1)

{

if(format == 3)

{

if((ta & 0x800) == 0x800) ta += 0xFF000;

ta += reg[8];

}

else ta += reg[8];

ta = ta & 0xFFFFF;

}

//indexed mode

if(x == 1)

{

ta += reg[1];

ta = ta & 0xFFFFF;

}

//simple addressing

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

{

if(ta <= 0xFFFFD)

{

sprintf(objectcode, "%02X%02X%02X", MEMORY[ta], MEMORY[ta + 1], MEMORY[ta + 2]);

sscanf(objectcode, "%06X", &data);

}

else

{

printf("Bound Error\n");

return ;

}

}

//indirect addressing

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

{

if(ta == 0xFFFFFF) end\_flag = 1;

else if(ta > 0xFFFFD)

{

printf("Bound Error\n");

return;

}

if(!end\_flag)

{

sprintf(objectcode, "%02X%02X%02X", MEMORY[ta], MEMORY[ta + 1], MEMORY[ta + 2]);

sscanf(objectcode, "%06X", &indirect);

}

if(indirect == 0xFFFFFF) end\_flag = 1;

else if(indirect > 0xFFFFD)

{

printf("Bound Error\n");

return ;

}

if(!end\_flag)

{

sprintf(objectcode, "%02X%02X%02X", MEMORY[indirect], MEMORY[indirect + 1], MEMORY[indirect + 2]);

sscanf(objectcode, "%06X", &data);

}

ta = indirect;

}

//immediate addressing

else if(n == 0 && i == 1) data = ta;

//COMP

if(opcode == 0x28)

{

if(reg[0] == data) state = 0;

else if(reg[0] > data) state = 1;

else if(reg[0] < data) state = 2;

}

//TD // RD // WD

else if(opcode == 0xE0 || opcode == 0xD8 || opcode == 0xDC) state = -1;

//J

else if(opcode == 0x3C) reg[8] = ta;

//JEQ

else if(opcode == 0x30)

{

if(state == 0) reg[8] = ta;

}

//JGT

else if(opcode == 0x34)

{

if(state == 1) reg[8] = ta;

}

//JLT

else if(opcode == 0x38)

{

if(state == 2) reg[8] = ta;

}

//JSUB

else if(opcode == 0x48)

{

reg[2] = reg[8];

reg[8] = ta;

}

//LDA

else if(opcode == 0x00) reg[0] = data;

//LDB

else if(opcode == 0x68) reg[3] = data;

//LDCH

else if(opcode == 0x50)

{

reg[0] = reg[0] & 0xFFFF00;

data = (data & 0xFF0000) / 0x010000;

reg[0] = reg[0] | data;

}

//LDL

else if(opcode == 0x08) reg[2] = data;

//LDS

else if(opcode == 0x6C) reg[4] = data;

//LDT

else if(opcode == 0x74) reg[5] = data;

//LDX

else if(opcode == 0x04) reg[1] = data;

//RSUB

else if(opcode == 0x4C) reg[8] = reg[2];

//STA

else if(opcode == 0x0C) store(reg[0], MEMORY, ta);

//STB

else if(opcode == 0x78) store(reg[3], MEMORY, ta);

//STCH

else if(opcode == 0x54) MEMORY[ta] = (unsigned char)(reg[0] & 0x0000FF);

//STL

else if(opcode == 0x14) store(reg[2], MEMORY, ta);

//STS

else if(opcode == 0x7C) store(reg[4], MEMORY, ta);

//STSW

else if(opcode == 0xE8) store(reg[9], MEMORY, ta);

//STT

else if(opcode == 0x84) store(reg[5], MEMORY, ta);

//STX

else if(opcode == 0x10) store(reg[1], MEMORY, ta);

//TIX

else if(opcode == 0x2C){

reg[1]++;

if(reg[1] == data) state = 0;

else if(reg[2] > data) state = 1;

else if(reg[2] < data) state = 2;

}

}

}

//Read All Memory Address -> END

if(reg[8] > 0xFFFFF) end\_flag = 1;

//End\_program

print\_reg(reg, 0, end\_flag);

if(end\_flag) \*run\_flag = 0;

}

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

/\*함수 : print\_loadmap(Estab \*head)

/\*목적 : 정상적으로 object file들의 code들이 메모리에 load되었음을 알리기 위해 loadmap을 출력하는 함수

/\*리턴값 : 없음

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

void print\_loadmap(Estab \*head)

{

Estab \*temp = head;

unsigned int tot = 0;

printf("control\t\tsymbol\t\taddress\t\tlength\n");

printf("section\t\tname\n");

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

while(temp)

{

if(temp->length)

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

else

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

tot += temp->length;

temp = temp->list;

}

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

printf("\t\t\t\ttotal length\t%04X\n",tot);

}

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

/\*함수 : print\_reg(int \*reg, int point, int end\_flag)

/\*목적 : Run 명령어 수행시 breakpoint에서 멈췄을 때 register 출력을 위한 함수

/\*리턴값 : 없음

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

//Print Register At Breakpoint function

void print\_reg(int \*reg, int point, int end\_flag)

{

printf("\tA : %06X X : %06X\n", reg[0], reg[1]);

printf("\tL : %06X PC : %06X\n", reg[2], reg[8]);

printf("\tB : %06X S : %06X\n", reg[3], reg[4]);

printf("\tT : %06X\n", reg[5]);

if(!end\_flag) printf("Stop at checkpoint[%04X]\n", point);

else printf("End program\n");

}

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

/\*함수 : estab\_insert(Estab \*\*head, Estab \*\*tail, char \*name, char \*length, char \*address,unsigned int csaddr)

/\*목적 : Loader pass1 수행시 external symbol define record에 있는 symbol들을 table로 build하는 함수

/\*리턴값 : 실패하면 1, 성공하면 0 리턴

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

int estab\_insert(Estab \*\*head, Estab \*\*tail, char \*name, char \*length, char \*address,unsigned int csaddr)

{

Estab \*temp;

int flag = find\_estab(\*head, name);

if(flag != -1) // 존재한다면

{

printf("Duplicate External Symbol Error\n");

return 1;

}

while(1)

{

if(tail != NULL)

break;

}

temp = (Estab \*)malloc(sizeof(Estab));

strcpy(temp->name, name);

sscanf(length, "%x", &temp->length);

sscanf(address,"%x", &temp->address);

temp->address += csaddr;

temp->list = NULL;

if(!(\*head)) \*head = \*tail = temp;

else

{

(\*tail)->list = temp;

(\*tail) = temp;

}

return 0;

}

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

/\*함수 : find\_estab(Estab \*head, char \*name)

/\*목적 : Loader pass2 수행시 external symbol reference를 위해 estab에서 참조하는 함수

/\*리턴값 : -1

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

int find\_estab(Estab \*head, char \*name)

{

Estab \*temp = head;

while(temp)

{

if(!strcmp(name, temp->name))

return temp->address;

temp = temp->list;

}

return -1;

}