**과목명: 시스템프로그래밍**

**2 분반**

**<<Project #1>>**

**서강대학교 [학부명] 컴퓨터공학과**

**[학번] 20171640**

**[이름] 박수진**

목 차

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

3.1 모듈 이름 : main()

3.1.1 기능

3.1.2 사용 변수

3.2 모듈 이름: editcmd(char \*cmd, char \*new, char \*idx)

3.2.1 기능

3.2.2 사용 변수

3.3 모듈이름: help()

3.3.1 기능

3.3.2 사용변수

3.4 모듈이름: dir()

3.4.1 기능

3.4.2 사용변수

3.5 모듈이름: history()

3.5.1 기능

3.5.2 사용변수

3.6 모듈 이름 : add\_list(char \*cmd) 6

3.6.1 기능

3.6.2 사용 변수

3.7 모듈 이름: dump(char \*idx)

3.7.1 기능

3.7.2 사용 변수

3.8 모듈이름: edit(char \*idx)

3.8.1 기능

3.8.2 사용변수

3.9 모듈이름: fill(char \*idx)

3.9.1 기능

3.9.2 사용변수

3.10 모듈이름: reset()

3.10.1 기능

3.10.2 사용변수

3.11 모듈 이름: makeOPlist()

3.11.1 기능

3.11.2 사용 변수

3.12 모듈이름: hashFunction(char \*inst)

3.12.1 기능

3.12.2 사용변수

3.13 모듈이름: opcode(char \*idx)

3.13.1 기능

3.13.2 사용변수

3.14 모듈이름: opcodelist()

3.14.1 기능

3.14.2 사용변수

1. **구조체 정의**

4.1 struct hiList

4.1.1 기능

4.1.2 사용 변수

4.2 struct OPlist

4.2.1 기능

4.2.2 사용 변수

1. **전역 변수 정의**

5.1 hiList \*hiHead

5.2 OPlist \*OPhead[20]

5.3 unsigned char memory[1048576];

5.4 int dumpend

1. **코드 설명**

6.1 20171640.h

6.2 20171640.c

6.3 Makefile

# 프로그램 개요

이 프로그램은 SIC/XE 머신을 구현하기 위한 기초가 된다. 셸과 유사한 입력 프롬프트를 가지며 셸 관련 명령어들을 처리한다. 1MB의 가상 메모리를 만든 후 이 메모리를 명령어를 통해 수정(특정 값 입력, 범위 내에서 특정 값으로 채우기, 초기화)하고 출력할 수 있다. 또한 opcode 값을 hash table에 저장하여 opcode list를 보거나 특정 mnemonic을 검색할 수 있다. 이 프로그램은 나중에 어셈블러, 링커, 로더를 실행하여 SIC/XE 머신을 구현할 수 있도록 업그레이드한다.

# 프로그램 설명

## 프로그램 흐름도

그림 1> 프로그램 흐름도

# 모듈 정의

## 모듈 이름 : main()

### 기능 - opcode hash table 만드는 함수를 호출한다 - 커맨드를 입력 받고 입력 받은 커맨드에 해당하는 함수를 호출한다. 해당 하는 함수가 없을 시에는 에러 메시지를 출력한다. - 프로그램이 끝나기 전에 동적 할당했던 history linked list와 opcode linked list를 모두 할당 해제한다.

### 사용 변수 - char oricmd[100] : 처음 입력 받은 커맨드 - char cmd[100] : 명령어 부분만 잘라낸 char 배열 - char idx[100] : 명령어 뒤의 파라미터를 저장한 char 배열 - hiList \*hifree, \*tmp1 : history linked list를 할당 해제하기 위한 변수 - OPlist \*opfree, \*tmp2 : opcode linked list를 할당 해제하기 위한 변수 - int i : 반복문에 사용할 변수

## 모듈 이름: editcmd(char \*cmd, char \*new, char \*idx)

### 기능

입력 받은 string을 명령어 뒤에 있는 ‘ ‘ 또는 ‘\t’을 기준으로 하여 두 개의 string으로 나눈다. 첫 번째 string은 명령어만 저장되고 두 번째 string에는 명령어의 뒷부분이 저장된다.

### 사용 변수 - int i, j : char 배열의 index를 표현하고 반복문에서도 사용 - char \*cmd (입력 파라미터) : 프로그램 시작 시에 입력 받은 전체 커맨드 - char \*new (입력 파라미터) : 명령어 부분만 잘라서 저장되는 변수 - char \*idx (입력 파라미터) : 명령어 뒷부분이 저장되는 변수

## 모듈이름: help()

### 기능 ‘h’ 또는 ‘help’가 입력되었을 때 프로그램 상에서 실행 가능한 명령어들을 출력한다

### 사용변수 없음

## 모듈이름: dir()

### 기능 현재 디렉토리 내에 있는 파일들을 출력한다. 실행파일 뒤에는 ‘\*’를 붙이고 디렉토리 뒤에는 ‘/’를 붙인다.

### 사용변수 - DIR \*ptrDir : 현재 디렉토리를 가리키는 디렉토리 포인터 - struct dirent \*ptrFile : 현재 디렉토리에 있는 파일의 정보에 접근할 수 있는 포인터 - struct stat fileStat : 파일이 디렉토리인지 실행파일인지 구분할 수 있도록 한다 - int num : 디렉토리 내 파일 개수

## 모듈이름: history()

### 기능 history linked list 내에 저장된 command 들을 차례로 출력한다.

### 사용변수 - hiList \*ptr : linked list 탐색에 사용되는 포인터 - int num : linked list에 저장된 노드 개수

## 모듈이름: add\_list(char \*cmd)

### 기능 history linked list에 입력 받은 command를 추가한다.

### 사용변수 - hiList \*newNode : 입력 받은 값이 저장될 새로운 노드 - hiList \*ptr : linked list를 탐색할 때 사용되는 포인터 - char \*cmd (입력 파라미터) : linked list에 추가될 명령어

## 모듈이름: dump(char \*idx)

### 기능 - 입력된 string에서 start와 end 값을 추출한다 (알맞지 않은 input은 에러 처리) - start와 end 값을 확인하고 범위를 벗어나면 에러 메시지 출력한다 - 왼쪽 칼럼엔 address(16진수)를 표시하고 가운데 칼럼에는 메모리 값(16진수)을 나타내고 오른쪽 칼럼에는 메모리 값에 해당하는 ASCII 코드를 나타낸다. - 에러가 발생하면 0 return, 정상 종료되면 1 return

### 사용변수 - int cmflag : 콤마의 인덱스를 표시 - int adflag : end address를 저장할지 결정 (dump 뒤에 인자가 없을 때 1) - int i, j : 배열의 인덱스와 반복문에 사용 - int start, end : dump 실행 시 start와 end에 해당하는 값 - char tmp[100] : 입력이 올바른지 확인할 때 사용하는 임시 변수 - char \*idx (입력 파라미터) : 명령어 뒤에 입력된 string

## 모듈이름: edit(char \*idx)

### 기능 - 입력된 string에서 address 값과 value 값을 추출한다 (올바르지 않은 input은 에러 처리) - address와 value가 허용된 범위 밖이면 에러 메시지 출력한다 - 메모리의 address에 입력된 value 값을 저장한다 - 에러 발생하면 0 return, 정상 종료되면 1 return

### 사용변수 - int cmflag : 콤마의 인덱스를 표시 - int i : 배열의 인덱스와 반복문에 사용 - int adr, val : edit 실행 시 사용되는 address와 value에 해당하는 값 - char tmp[100] : 입력이 올바른지 확인할 때 사용하는 임시 변수 - char \*idx (입력 파라미터) : 명령어 뒤에 입력된 string

## 모듈이름: fill(char \*idx)

### 기능 - 입력된 string에서 start, end, value 값을 추출한다 (올바르지 않은 input은 에러 처리) - start, end, value 값이 허용된 범위 밖이면 에러 메시지 출력한다 - 메모리의 start address 부터 end address 까지의 값의 value로 채운다 - 에러 발생하면 0 return, 정상 종료되면 1 return

### 사용변수 - int cm1, cm2 : 각각 콤마의 인덱스 표현 - int i : 배열의 인덱스 표현과 반복문에 사용 - int start, end, value : fill 실행 시 사용되는 start, end, value 값을 각각 저장 - char tmp[100] : 입력이 올바른지 확인할 때 사용하는 임시 변수 - char \*idx (입력 파라미터) : 명령어 뒤에 입력된 string

## 모듈이름: reset()

### 기능 메모리에 저장된 모든 값을 0으로 초기화 한다

### 사용변수 없음

## 모듈이름: makeOPlist()

### 기능 - ‘opcode.txt’ 파일로부터 opcodelist를 불러와서 opcode linked list 로 구성 - opcode linked list는 hash table로 구현

### 사용변수 - FILE \*fp : “opcode.txt”를 불러오는 파일포인터 - char inst[10], form[5] : 각각 mnemonic과 format을 저장하고 있다 - int op : mnemonic에 해당하는 opcode 값이 저장되어있다 - int tmp : hash function을 거친 hash table 인덱스 값 - OPlist \*newNode : opcode가 저장된 linked list 포인터 (새 노드) - OPlist \*ptr : newNode를 linked list에 연결할 때 사용되는 포인터

## 모듈이름: hashFunction(char \*inst)

### 기능 임의의 hash function을 사용하여 입력 instruction을 0~19의 값으로 암호화한 후 리턴

### 사용변수 - int i : 암호화된 값을 저장 - int len : instruction의 길이를 저장 - char\* inst (입력 파라미터) : 입력으로 들어온 instruction

## 모듈이름: opcode(char \*idx)

### 기능 - 입력 받은 idx에서 mnemonic 추출 하여 해당하는 opcode 값 출력 (올바르지 않은 입력은 에러 처리) - 에러 발생 시 return 0, 정상 종료 시 return 1

### 사용변수 - char tmp[100] : 입력이 올바른지 확인할 때 사용하는 임시 변수 - char inst[10] : mnemonic 이름을 저장 - int i : hash function을 거쳐 암호화된 값 저장 -> hash table 인덱스 - OPlist \*ptr : linked list 탐색을 위한 포인터 - char \*idx (입력 파라미터) : 명령어 뒤의 string

## 모듈이름: opcodelist()

### 기능 hash table에 저장된 opcodelist를 차례로 출력한다

### 사용변수 - int i : 배열의 인덱스와 반복문에 사용 - OPlist \*ptr : linked list를 탐색할 때 사용되는 포인터

# 구조체 정의

### **4.1 struct hiList**

### 기능 history 를 저장한 linked list에 사용되는 구조체

### 사용변수 - char cmd\_name[100] : 입력된 커맨드를 저장 - struct hiList \*link : linked list에서 다음 노드를 가리키는 포인터

### **4.2 struct OPlist**

### 4.2.1 기능 opcodelist가 저장된 linked list에 사용되는 구조체

### 4.2.2 사용변수 - int opcode : mnemonic에 해당하는 opcode 값 - char inst\_name[10] : mnemonic의 이름 - char form[5] : mnemonic의 형식 - struct OPlist \*link : linked list에서 다음 노드를 가리키는 포인터

# 전역 변수 정의

## hiList \*hiHead history linked list의 head가 저장된 포인터

## OPlist \*OPhead[20] opcode 가 저장된 linked list의 head 포인터의 집합

## unsigned char memory[1048576] 1Mbyte (16 X 65536)의 가상 메모리 공간

## int dumpend dump 실행 시 마지막으로 출력된 end 주소를 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;

/\* global 변수 \*/

hiList \*hiHead = NULL; // hiList head

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

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

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

/\* 함수 원형 \*/

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

/\* 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();

## 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

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

}

// 그 이외의 입력 출력하지 않음

}

// 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);

}

}

return 0;

}

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");

}

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;

adflag = 1;

}

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

// 읽어들인 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,%X]", ptr->inst\_name, ptr->opcode);

}

printf("\n");

}

}

## Makefile

OBJECTS = 20171640.o #생성할 오브젝트파일명

SRC = 20171640.c #소스파일명

CC = gcc #gcc로 컴파일

CFLAGS = -g -Wall -W #옵션을 -g -Wall -W로 걸어준다 (강한 warning)

TARGET = 20171640.out #생성할 실행파일명

$(TARGET) : $(OBJECTS)

$(CC) -o $(TARGET) $(OBJECTS) $(CFLAGS)

#make clean하면 object파일과 target파일을 지운다

clean:

rm $(OBJECTS) $(TARGET)