과목 명: 시스템프로그래밍

담당 교수 명: 김 지 환

<<Assignment 1>>

**서강대학교 커뮤니케이션학과**

**[20111166]**

**[윤재호]**

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: list\_init() 5

3.2.1 기능 5

3.2.2 사용 변수 5

3.3 모듈이름: list\_insert() 5

3.3.1 기능 5

3.3.2 사용변수 5

3.4 모듈이름: hash\_function() 6

3.4.1 기능 6

3.4.2 사용변수 6

3.5 모듈이름: make\_hashTable() 6

3.5.1 기능 6

3.5.2 사용변수 6

3.6 모듈이름: print\_list() 6

3.6.1 기능 6

3.6.2 사용변수 6

3.7 모듈이름: print\_op\_table() 6

3.7.1 기능 6

3.7.2 사용변수 6

3.8 모듈이름: free\_list() 6

3.8.1 기능 6

3.8.2 사용변수 6

3.9 모듈이름: free\_hashTable() 7

3.9.1 기능 7

3.9.2 사용변수 7

3.10 모듈이름: tokenizer() 7

3.10.1 기능 7

3.10.2 사용변수 7

3.11 모듈이름: count\_token\_with\_comma() 7

3.11.1 기능 7

3.11.2 사용변수 7

3.12 모듈이름: command() 7

3.12.1 기능 7

3.12.2 사용변수 7

3.13 모듈이름: copy\_str() 8

3.13.1 기능 8

3.13.2 사용변수 8

3.14 모듈이름: dir() 8

3.14.1 기능 8

3.14.2 사용변수 8

3.15 모듈이름: find\_opcode() 8

3.15.1 기능 8

3.15.2 사용변수 8

3.16 모듈이름: handle\_error() 8

3.16.1 기능 8

3.16.2 사용변수 8

3.17 모듈이름: memory\_dump() 9

3.17.1 기능 9

3.17.2 사용변수 9

3.18 모듈이름: memory\_start\_dump() 9

3.18.1 기능 9

3.18.2 사용변수 9

3.19 모듈이름: memory\_range\_dump() 9

3.19.1 기능 9

3.19.2 사용변수 9

3.20 모듈이름: memory\_edit() 9

3.20.1 기능 9

3.20.2 사용변수 9

3.21 모듈이름: memory\_fill() 10

3.21.1 기능 10

3.21.2 사용변수 10

3.22 모듈이름: memory\_reset() 10

3.22.1 기능 10

3.22.2 사용변수 10

4. 전역 변수 정의 4

5. 코드 11

*5.1* 헤더 파일 11

5.1.1 20111166.h 11

*5.2* C 파일 12

5.2.1 20111166.c 12

5.2.2 data\_structure.c 13

5.2.3 shell.c 17

5.2.4 memory\_management.c 20

5.2.5 etc\_function.c 23

# 프로그램 개요

SIC/XE 머신을 구현하기 위한 전 단계로써 어셈블러, 링크, 로더들을 실행하게 될 셸(shell)과 컴파일을 통해서 만들어진 object코드가 적재되고 실행될 메모리공간과 mnemonic (ADD, COMP, FLOAT, etc …)을 opcode값으로 변환하는 OPCODE 테이블과 관련 명령어들을 구현하는 프로그램이다.

# 프로그램 설명

## 프로그램 흐름도

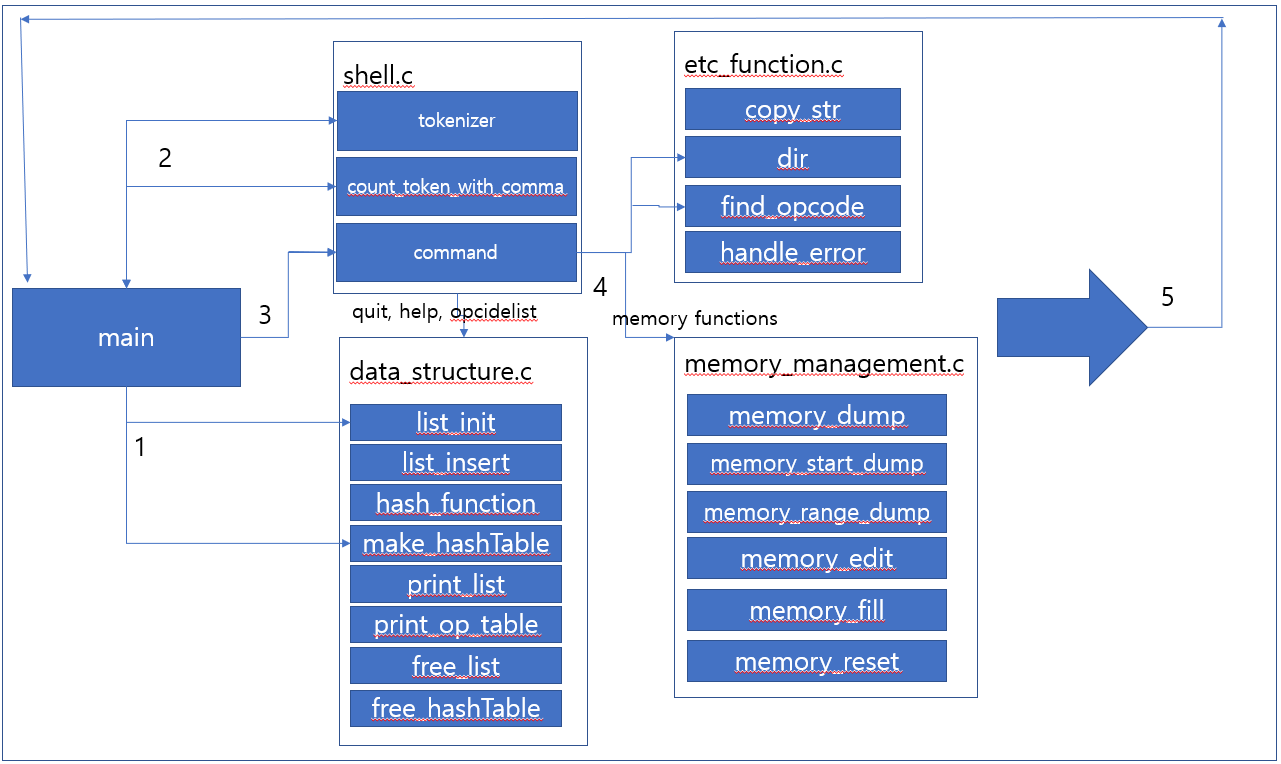


그림 1> 프로그램 흐름도

main 함수에서 opcode\_table과 history\_list를 초기화(list\_init(), make\_hashTable() )후, while loop을 통하여 프로그램을 반복한다. 명령어를 입력받은 명령어는 tokenizer()함수로 token으로 분리하고 token의 개수를 돌려받는다. count\_token\_with\_comma()함수를 통해서 제대로 된 입력인지 체크한다. 그 후, shell 모듈의 command()함수를 통하여 입력받은 명령을 수행한다. 각각의 다른 모듈들은 command()에서 필요한 함수들을 호출하여 명령을 수행한다. 하나의 명령이 끝난 후, main의 while loop으로 돌아가 종료가 될 때까지 반복 작업한다.

# 모듈 정의

## 모듈 이름 : main()

### 기능

필요한 자료구조를 정의한다. list\_init(), make\_hashTable()함수를 통하여 자료구조를 초기화한다. while\_loop을 통하여 명령을 입력받고 tokenizer, count\_token\_with\_comma 함수를 통해서 명령을 token으로 나누고 검사를 한다. command함수를 호출하여 명령을 수행하고 문제가 없다면 history\_list에 명령어를 입력하고 다시 작업을 반복한다.

### 사용 변수

List \*history\_list - 명령어 history을 저장하는 list

List \*op\_table[20] - opcode를 저장하는 hashtable

unsigned char memory[MEM\_COL][MEM\_ROW] - 16\*65536 = 1MB 크기 메모리

char command\_str[MAX\_STR] - 명령어를 입력받는 문자열

char \*token[MAX\_TOKEN] - 명령어를 token으로 나눠 저장

char \*temp - 명령어를 복사 후, tokenizer 함수에 전달

char \*comp\_temp - 명령어를 복사 후, count\_token\_with\_comma 함수에 전달

int token\_number - 나눈 토큰의 갯수

int comma\_token\_number - comma로 나눈 토큰의 개수

## 모듈 이름: list\_init()

### 기능

list를 초기화한다

.

### 사용 변수

List \*list - 초기화할 list

## 모듈이름: list\_insert()

### 기능

list에 node를 삽입한다.

### 사용변수

char \*str - 노드에 담을 string, 또는 명령어 history

int op\_code - 노드에 담을 opcode

## 모듈이름: hash\_function()

### 기능

hashtable에 저장하기 위한 hash값을 생성한다. 문자열의 각각의 ascii값을 더한다. 이 때, 같은 문자들이 순열을 이루어서 중복된 값을 막기 위해 g값을 이용하여 계산한다.

### 사용변수

char \*s - hashtable에 저장할 문자열

const int g - hash\_function()에서 사용. 순열 시 값이 중복되는 것을 막기 위해 곱하는 상수

## 모듈이름: make\_hashTable()

### 기능

hashtable을 초기화하고 opcode.txt 파일을 열어 그 안의 값들을 hashtable에 저장한다. .

### 사용변수

List \*\*hashTable - hashTable을 저장할 포인터

## 모듈이름: print\_list()

### 기능

list에 담긴 내용을 출력한다.

### 사용변수

List \*list - 출력할 list

## 모듈이름: print\_op\_table()

### 기능

hashtable에 담긴 optable을 출력한다.

### 사용변수

List \*\*op\_table - 출력할 optable

## 모듈이름: free\_list()

### 기능

list에 할당된 메모리를 free 시킨다.

### 사용변수

List \*list - free시킬 list

## 모듈이름: free\_hashTable()

### 기능

hashtable에 할당된 메모리를 free시킨다.

### 사용변수

List \*\*hashTable - free 시킬 hashtable

## 모듈이름: tokenizer()

### 기능

입력 받은 명령어를 blank, \t, \n, “,”를 기준으로 나눠 token에 저장한다..

### 사용변수

char \*delim - 나누는 기준이 되는 문자들

char \*str - 입력되는 명령어

char \*\*token - 나눠진 token들을 담을 token array

int i - 나눠진 token의 개수

## 모듈이름: count\_token\_with\_comma()

### 기능

tokenize로 나눈 토큰들은 comma에 따라서만 나눠지지 않기 때문에 dump AA, BB와 dump AA BB를 구분하지 못함. count\_token\_with\_comma()는 comma로 나눈 토큰의 개수를 세어서 토큰의 개수에 따라 제대로 된 input이 들어왔는 지 여부를 알 수 있다.

### 사용변수

char \*str - 입력받은 명령어

int i - comma로 나눈 token의 갯수

## 모듈이름: command()

### 기능

프로그램의 명령들을 실제로 수행하는 함수. token에 저장된 명령어에 따라 함수들을 호출하여 실행한다.

### 사용변수

unsigned char memory[][MEM\_ROW] - 프로그램에 할당된 메모리

char \*\*token - token으로 나뉘어진 명령어들

int token\_number - token의 개수

char \*history\_temp - history명령어가 수행되기 전에 명령어를 history\_list에 담기 위해 전달하는 명령어

List \*history\_list - history list

List \*\*op\_table - optable

static int address - dump를 위해 현재 가리키고 있는 address를 저장하는 static 변수

int start, end, value - 문자열로 저장되어진 값들을 hex값으로 저장하는 변수

char \*check1, \*check2, \*check3 - strtol함수를 사용할 때, 제대로 변환이 되었는지 확인하기 위한 포인터

## 모듈이름: copy\_str()

### 기능

from에서 to로 str을 복사한다.

### 사용변수

char \*from - 저장시킬 str

char \*\*to - 저장할 str

## 모듈이름: dir()

### 기능

현재 디렉토리의 모든 파일들을 출력한다.

### 사용변수

Dir \*dir - 현재 디렉토리에 대한 정보를 담는 struct

struct stat buf - 파일들의 상태 정보를 담는 struct

struct dirent \*dp - 현재 디렉토리의 파일링크를 담는 struct

## 모듈이름: find\_opcode()

### 기능

입력받은 mnemonic의 opcode를 optable에서 찾아 출력한다.

### 사용변수

List \*\*opTable - 찾아 볼 optable

char \*mnemonic - 입력받은 mnemonic

int hash - hashtable에서 찾기 위해 다시금 hash값을 찾아 저장

## 모듈이름: handle\_error()

### 기능

argument로 넘어오는 에러 상황에 따라 메시지를 출력

### 사용변수

error\_num num - typedef enum으로 정의한 에러 유형

## 모듈이름: memory\_dump()

### 기능

현재 보고 있는 address에서부터 160개의 메모리 공간을 형식에 맞게 출력한다.

### 사용변수

int \*address - 현재 보고 있는 address

unsigned char memory[][MEM\_ROW] - 메모리 공간

## 모듈이름: memory\_start\_dump()

### 기능

입력 받은 address로부터 160개의 메모리 공간을 형식에 맞게 출력한다.

### 사용변수

int \*address - 현재 보고 있는 address를 끝난 위치 다음 지점을 보게 하기 위해 전달

unsigned char memory[][MEM\_ROW] - 메모리 공간

int start - 입력 받은 출력을 시작할 address

## 모듈이름: memory\_range\_dump()

### 기능

시작 지점과 끝 지점 사이의 메모리를 형식에 맞게 출력한다.

### 사용변수

int \*address - 현재 보고 있는 address를 끝난 위치 다음 지점을 보게 하기 위해 전달

unsigned char memory[][MEM\_ROW] - 메모리 공간

int start - 입력 받은 출력을 시작할 address

int end - 입력을 마치는 address

int rows, cols, end\_rows - 시작하는 col, row와 끝마치는 row의 값을 계산하여 저장하는 변수

int range - 입력할 메모리의 cols의 범위

## 모듈이름: memory\_edit()

### 기능

입력 받은 address에 value를 저장한다.

### 사용변수

int address - 메모리 값을 수정하기 위한 주소

unsigned char memory[][MEM\_ROW] - 메모리 공간

int value - 수정할 value

## 모듈이름: memory\_fill()

### 기능

start, end 주소 사이에 value값으로 메모리를 채운다.

### 사용변수

unsigned char memory[][MEM\_ROW] - 메모리 공간

int start - 시작 address

int end - 끝 address

int value - 채워 넣을 value

int rows, cols, end\_rows - 시작하는 col, row와 끝마치는 row의 값을 계산하여 저장하는 변수

int range - 입력할 메모리의 cols의 범위

## 모듈이름:memory\_reset()

### 기능

메모리 공간을 0으로 채워넣는다.

### 사용변수

unsigned char memory[][MEM\_ROW] - 메모리 공간

# 전역 변수 정의

전역 변수는 사용하지 않음.

# 코드

## 헤더파일

### 20111166.h

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

/\*정의되는 상수\*/

#define MAX\_TOKEN 6

#define MAX\_STR 256

#define MEM\_ROW 16

#define MEM\_COL 65536

/\*정의되는 구조체\*/

typedef struct \_Node {

int op\_code;

char \*content;

struct \_Node \*next;

} Node;

typedef struct \_List {

Node \*head;

Node \*tail;

} List;

/////////linked-list

typedef enum {

out\_of\_boundary,

wrong\_input,

not\_found,

not\_number,

not\_allocate,

} error\_num;

///////error enum type

/\*함수 원형\*/

void list\_init(List \*list);

int list\_insert(List \*list, char \*str, int op\_code);

int hash\_function(char \*s);

void make\_hashTable(List \*\*hashTable);

void print\_list(List \*list);

void print\_op\_table(List \*\*hashTable);

void free\_list(List \*list);

void free\_hashTable(List \*\*hashTable);

///////////data\_structure

int tokenizer(char \*str, char \*\*token);

int count\_token\_with\_comma(char \*str);

int command(unsigned char memory[][MEM\_ROW], List \*history\_list, List \*\*op\_table, char \*\*token, int token\_number, char \*history\_temp);

/////////////shell

void copy\_str(char \*from, char \*\*to);

int dir(void);

int find\_opcode(List \*\*opTable, char \*mnemonic);

int handle\_error(error\_num num);

//////etc\_function

int memory\_dump(unsigned char memory[][MEM\_ROW], int \*address);

int memory\_start\_dump(unsigned char memory[][MEM\_ROW], int start, int \*address);

int memory\_range\_dump(unsigned char memory[][MEM\_ROW], int start, int end, int \*address);

int memory\_edit(unsigned char memory[][MEM\_ROW], int address, int value);

int memory\_fill(unsigned char memory[][MEM\_ROW], int start, int end, int value);

void memory\_reset(unsigned char memory[][MEM\_ROW]);

///memory\_management

## C 파일

### 20111166.c

#include "20111166.h"

/\*프로그램 시작\*/

int main() {

/\*변수\*/

List \*history\_list;

List \*op\_table[20];

unsigned char memory[MEM\_COL][MEM\_ROW]; //1MB memory

char command\_str[MAX\_STR];

char \*token[MAX\_TOKEN]; //max number of tokens is 4

char \*temp, \*comp\_temp;

int token\_number, comma\_token\_number;

history\_list = (List \*)malloc(sizeof(List));

list\_init(history\_list);

make\_hashTable(op\_table);

while(1) {

printf("sicsim> ");

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

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

copy\_str(command\_str, &temp);

copy\_str(command\_str, &comp\_temp);

token\_number = tokenizer(temp, token); //tokenize the command and return the number of tokens

comma\_token\_number = count\_token\_with\_comma(comp\_temp); //check that input is right //or not(about comma)

if(token\_number != comma\_token\_number) { //check if num of comma is not same

handle\_error(wrong\_input);

continue;

}

if(token\_number == 0 || token\_number >4) { //check if num of token is too small or too much

handle\_error(wrong\_input);

continue;

}

if(!command(memory, history\_list, op\_table, token, token\_number, command\_str)) { //process the command

list\_insert(history\_list, command\_str, 0); //if there's no error, put the last command //into the history list

}

}

return 0;

}

### data\_structure.c

#include "20111166.h"

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

/\*함수 : list\_init\*/

/\*목적 : list를 초기화한다\*/

/\*리턴값: 없음\*/

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

void list\_init(List \*list)

{

list->head = NULL;

list->tail = NULL;

}

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

/\*함수 : list\_insert\*/

/\*목적 : list에 노드를 추가한다\*/

/\*리턴값: 0 = 정상, -1: error \*/

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

int list\_insert(List \*list, char \*str, int op\_code)

{

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

if(temp == NULL) {

return handle\_error(not\_allocate);

}

copy\_str(str, &temp->content);

temp->op\_code = op\_code;

temp->next = NULL;

if(list->head == NULL) {

list->head = temp;

list->tail = temp;

} else {

list->tail->next = temp;

list->tail = temp;

}

return 0;

}

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

/\*함수 : print\_list\*/

/\*목적 : list를 출력한다\*/

/\*리턴값: 없음\*/

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

void print\_list(List \*list)

{

Node \*temp;

int num;

temp = list->head;

num = 1;

if(temp == NULL) return;

while(temp->next!=NULL) {

printf("%d %s\n", num++, temp->content);

temp = temp->next;

}

printf("%d %s\n", num++, temp->content); //print tail

return;

}

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

/\*함수 : hash\_function\*/

/\*목적 : hash값을 생성\*/

/\*리턴값: hash\*/

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

int hash\_function(char \*s)

{

int i;

const int g = 31; //hash constant.

int hash=0; //hash index

int length = strlen(s);

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

hash = g \* hash + s[i]; //add each char digit. and multiply by g. so it doensn't allow //any permutation overlap

hash = hash%20;

}

return hash;

}

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

/\*함수 : make\_hashTable\*/

/\*목적 : hashtable을 초기화하고 구성한다\*/

/\*리턴값: 없\*/

/\*----------음-----------\*/

void make\_hashTable(List \*\*hashTable)

{

FILE \*fp;

unsigned int op\_code;

char mnemonic[8];

char format[5];

int i;

fp = fopen("opcode.txt", "r");

if(fp == NULL) {

printf("file can not open\n");

return;

}

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

hashTable[i] = (List \*)malloc(sizeof(List));

list\_init(hashTable[i]);

}

while(fscanf(fp, "%x %s %s", &op\_code, mnemonic, format) != EOF) {

int hash = hash\_function(mnemonic);

list\_insert(hashTable[hash], mnemonic, op\_code); //insert to the hashtable by //hash\_index which is from hash\_function

}

fclose(fp);

return;

}

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

/\*함수 : print\_op\_table\*/

/\*목적 : optable을 출력한다\*/

/\*리턴값: 없음\*/

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

void print\_op\_table(List \*\*op\_table) //print table

{

int i;

Node \*temp;

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

temp = op\_table[i]->head;

if(temp == NULL) { //nothing in the list

printf("%d : empty\n", i);

continue;

}

if(temp->next != NULL) { //case1: theres' next node

printf("%d : [%s, %02X] ->", i, temp->content, temp->op\_code);

} else { //case2: no next node

printf("%d : [%s, %02X]\n", i, temp->content, temp->op\_code);

continue;

}

temp = temp->next;

while(temp->next != NULL) {

printf(" [%s, %02X] ->", temp->content, temp->op\_code);

temp = temp->next;

}

if(op\_table[i]->head != NULL) { //print last node

printf(" [%s, %02X]\n", temp->content, temp->op\_code);

}

}

}

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

/\*함수 :free\_list\*/

/\*목적 : list 할당된 메모리를 해제한다\*/

/\*리턴값: 없음\*/

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

void free\_list(List \*list) //free list

{

if(list == NULL) return;

if(list->head == NULL) { //there's nothing in the list

free(list);

return;

}

Node \*temp;

temp = list->head;

while(temp->next != NULL) { //free content, node

temp = list->head;

free(temp->content);

list->head = list->head->next;

free(temp);

}

free(list);

return;

}

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

/\*함수 : free\_hashTable\*/

/\*목적 : hashtable 할당된 메모리를 해제한다.\*/

/\*리턴값: 없음\*/

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

void free\_hashTable(List \*\*hashTable)

{

int i;

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

free\_list(hashTable[i]);

}

return;

}

### shell.c

#include "20111166.h"

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

/\*함수 : tokenizer\*/

/\*목적 : str를 나눠 token에 저장한다\*/

/\*리턴값: token의 갯수\*/

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

int tokenizer(char \*str, char \*\*token)

{

int i=0;

char \*delim = " ,\t\n"; //first delimeter. tokenize except , \t, \n.

token[i++] = strtok(str, delim); //save the token into the token[i]

while(token[i-1] != NULL) {

if(i >= 5) return 5;

token[i++] = strtok(NULL, delim);

}

if(token[i-2] && token[i-2][strlen(token[i-2])-1] == '\n') { //remove end of token '\n'

token[i-2][strlen(token[i-2])-1] = '\0';

}

return (i-1); //return number of token

}

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

/\*함수 : count\_token\_with\_comma\*/

/\*목적 : comma에 따라 토큰의 갯수를 센다\*/

/\*리턴값: 토큰의 갯수\*/

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

int count\_token\_with\_comma(char \*str) //check that input is right or not about comma

{

int i=0;

char \*token;

strsep(&str, " ,\t\n");

i++;

while ((token = strsep(&str, ",")) != NULL) {

i++;

}

return i;

}

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

/\*함수 : command\*/

/\*목적 : 저장된 토큰에 따라 명령을 수행한다\*/

/\*리턴값: 0 = 정상, -1 = error\*/

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

int command(unsigned char memory[][MEM\_ROW], List \*history\_list, List \*\*op\_table, char \*\*token, int token\_number, char \*history\_temp) //history\_temp : to handle history. if input is "hi" or "history",

{ //input the command into the history list before print it.

static int address = 0; //internal address pointer

int start, end, value; //change into hex numbers

char \*check1, \*check2, \*check3; //check whether change is ok

int status=0;

//distinguish command by token number

if(token\_number == 1) { //help, dir, quit, history, dump, reset, opcodelist

if(!strcmp(token[0], "h") || !strcmp(token[0], "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");

} else if (!strcmp(token[0], "d") || !strcmp(token[0], "dir")) {

dir();

} else if (!strcmp(token[0], "q") || !strcmp(token[0], "quit")) {

//before exit, free all assigned memory

free\_list(history\_list);

free\_hashTable(op\_table);

exit(0);

} else if (!strcmp(token[0], "hi") || !strcmp(token[0], "history")) {

list\_insert(history\_list, history\_temp, 0);

status = -1; //to not write history again in the history list

print\_list(history\_list);

} else if (!strcmp(token[0], "du") || !strcmp(token[0], "dump")) {

status = memory\_dump(memory, &address);

} else if (!strcmp(token[0], "reset")) {

memory\_reset(memory);

} else if (!strcmp(token[0], "opcodelist")) {

print\_op\_table(op\_table);

} else {

status = handle\_error(wrong\_input);

}

} else if (token\_number == 2) { //opcode mnemonic, dump start

int op\_code;

if(!strcmp(token[0], "opcode")) {

op\_code = find\_opcode(op\_table, token[1]);

if(op\_code == -1) {

status = handle\_error(not\_found);

} else {

printf("opcode is %02X\n", op\_code);

}

} else if(!strcmp(token[0], "du") || !strcmp(token[0], "dump")) {

start = (int)strtol(token[1], &check1, 16);

if(\*check1 == '\0') {

status = memory\_start\_dump(memory, start, &address);

} else {

status = handle\_error(not\_number);

}

} else {

status = handle\_error(wrong\_input);

}

} else if (token\_number == 3) { //(dump start, end), (edit address, value)

if(!strcmp(token[0], "du") || !strcmp(token[0], "dump")) {

start = (int)strtol(token[1], &check1, 16);

end = (int)strtol(token[2], &check2, 16);

if(\*check1 == '\0' && \*check2 == '\0') {

status = memory\_range\_dump(memory, start, end, &address);

} else {

status = handle\_error(not\_number);

}

} else if(!strcmp(token[0], "e") || !strcmp(token[0], "edit")) {

address = (int)strtol(token[1], &check1, 16);

value = (int)strtol(token[2], &check2, 16);

if(\*check1 == '\0' && \*check2 == '\0') {

status = memory\_edit(memory, address, value);

} else {

status = handle\_error(not\_number);

}

} else {

status = handle\_error(wrong\_input);

}

} else if (token\_number == 4) { //fill start, end, value

if(!strcmp(token[0], "f") || !strcmp(token[0], "fill")) {

start = (int)strtol(token[1], &check1, 16);

end = (int)strtol(token[2], &check2, 16);

value = (int)strtol(token[3], &check3, 16);

if(\*check1 == '\0' && \*check2 == '\0' && \*check3 == '\0') {

status = memory\_fill(memory, start, end, value);

} else {

status = handle\_error(not\_number);

}

} else {

status = handle\_error(wrong\_input);

}

}

return status;

}

### memory\_management.c

#include "20111166.h"

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

/\*함수 : memory\_range\_dump\*/

/\*목적 : 범위 안의 메모리 값을 출력한다\*/

/\*리턴값: 0= 정상, -1 = 비정상\*/

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

int memory\_range\_dump(unsigned char memory[][MEM\_ROW], int start, int end, int \*address) { //dump memory from start, to end

int rows, cols, end\_rows;

int range;

int i,j;

if(start > 0xfffff || end > 0xfffff || (start > end) || start < 0 || end < 0) { //error handling(check the start, end, address are in the memory boundary)

return handle\_error(out\_of\_boundary);

}

cols = start/16; //start col

rows = start%16; //start row

end\_rows = end%16; //end col

range = end/16 - start/16 + 1;

for(i=cols; i<range + cols; i++) {

printf("%05X ", i\*16); //memory address

for(j=0; j<16; j++) { //memory content

if((i==cols && j < rows) || (i == range + cols - 1 && j > end\_rows)) { //handle the exception case(first, end last row)

printf(" ");

} else {

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

}

}

printf("; ");

for(j=0; j<16; j++) { //print ascii code

if((i==cols && j<rows) || (i == range + cols - 1 && j > end\_rows)) { //handle the exception case(first, end last row)

printf(".");

} else {

if(memory[i][j] >= 0x20 && memory[i][j] <= 0x7E) {

printf("%c", memory[i][j]);

} else {

printf(".");

}

}

}

printf("\n");

}

\*address = end + 1;

return 0;

}

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

/\*함수 : memory\_dump\*/

/\*목적 : address부터 160개의 메모리값을 출력한다\*/

/\*리턴값: 0= 정상, -1 = 비정상\*/

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

int memory\_dump(unsigned char memory[][MEM\_ROW], int \*address)

{

if(\*address + 159 > 0xFFFFF) {

return handle\_error(out\_of\_boundary);

}

memory\_range\_dump(memory, \*address, \*address+159, address);

return 0;

}

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

/\*함수 : memory\_start\_dump\*/

/\*목적 : start address부터 160개의 메모리 값을 출력한다\*/

/\*리턴값: 0= 정상, -1 = 비정상\*/

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

int memory\_start\_dump(unsigned char memory[][MEM\_ROW], int start, int \*address)

{

if(start + 159 > 0xFFFFF) {

return memory\_range\_dump(memory, start, 0xFFFFF, address);

} else {

return memory\_range\_dump(memory, start, start+159, address);

}

}

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

/\*함수 : memory\_edit\*/

/\*목적 : address의 메모리 값을 수정한다\*/

/\*리턴값: 0= 정상, -1 = 비정상\*/

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

int memory\_edit(unsigned char memory[][MEM\_ROW], int address, int value)

{

int rows, cols;

if(address < 0 || address > 0xfffff || value < 0 || value > 0xff) {

return handle\_error(out\_of\_boundary);

}

cols = address/16; //start col

rows = address%16; //start row

memory[cols][rows] = value;

return 0;

}

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

/\*함수 : memory\_fill\*/

/\*목적 : 범위 안의 메모리 값을 value로 채운다\*/

/\*리턴값: 0= 정상, -1 = 비정상\*/

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

int memory\_fill(unsigned char memory[][MEM\_ROW], int start, int end, int value)

{

int rows, cols, end\_rows, range;

int i, j;

if(start > 0xfffff || end > 0xfffff || (start > end) || start < 0 || end < 0 || value < 0 || value > 0xff) {

return handle\_error(out\_of\_boundary);

}

cols = start/16; //start col

rows = start%16; //start row

end\_rows = end%16; //end row

range = end/16 - start/16 + 1;

for(i=cols; i<range + cols; i++) {

for(j=0; j<16; j++) { //memory content

if((i==cols && j < rows) || (i == range + cols - 1 && j > end\_rows)) {

continue;

} else {

memory[i][j] = value;

}

}

}

return 0;

}

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

/\*함수 : memory\_reset\*/

/\*목적 : 메모리 값을 전부 0으로 초기화한다\*/

/\*리턴값: 없음\*/

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

void memory\_reset(unsigned char memory[][MEM\_ROW])

{

int i, j;

for(i=0; i<MEM\_COL; i++) {

for(j=0; j<MEM\_ROW; j++) {

memory[i][j] = 0;

}

}

return;

}

### etc\_function.c

#include "20111166.h"

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

/\*함수 : copy\_str\*/

/\*목적 : str을 카피한다\*/

/\*리턴값: 없음\*/

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

void copy\_str(char \*from, char \*\*to)

{

int length = strlen(from);

\*to = (char \*)malloc(sizeof(char)\*length+1);

strncpy(\*to, from, length+1);

}

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

/\*함수 : dir\*/

/\*목적 : 현재 디렉토리의 모든 파일을 ㅊ ㅜㄹ력한다\*/

/\*리턴값: 0 = 정상, -1 = error\*/

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

int dir(void) // TODO: check error

{

DIR \*dir = NULL;

struct stat buf;

struct dirent \*dp = NULL;

dir = opendir("."); //open current directory stream

if(dir) {

while((dp = readdir(dir))) { //point to the next file

lstat(dp->d\_name, &buf);

if( S\_ISDIR(buf.st\_mode)) { //is it directory?

printf("%s\\ ", dp->d\_name);

} else if (S\_IXUSR & buf.st\_mode) { //or is it executable file?

printf("%s\* ", dp->d\_name);

} else { //else

printf("%s ", dp->d\_name);

}

}

closedir(dir);

printf("\n");

} else {

printf("can not open directory\n");

return -1;

}

return 0;

}

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

/\*함수 : find\_opcode\*/

/\*목적 : mnemonic에 맞는 opcode를 찾는다\*/

/\*리턴값: opcode, -1 = error\*/

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

int find\_opcode(List \*\*opTable, char \*mnemonic)

{

Node \*temp;

int hash;

hash = hash\_function(mnemonic); //find where the mnemonic is saved in the hashTable

temp = opTable[hash]->head;

while(temp != NULL) {

if(!strcmp(temp->content, mnemonic)) {

return temp->op\_code; //return the opcode of the mnemonic

}

temp = temp->next;

}

return -1; //can't find

}

//

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

/\*함수 : handle\_error\*/

/\*목적 : error에 맞는 메세지를 출력한다\*/

/\*리턴값: -1 = error\*/

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

int handle\_error(error\_num num) { //error handling by num which is defined enum error type

switch(num) {

case out\_of\_boundary :

printf("out of memory boundary\n");

break;

case wrong\_input :

printf("wrong input. please try again\n");

break;

case not\_found :

printf("can't find mnemonic. try again\n");

break;

case not\_number :

printf("you have to input number arguments. try again\n");

break;

case not\_allocate :

printf("memory allocation fail\n");

break;

default :

printf("something wrong. try again\n");

break;

}

return -1; //return -1, so do not put the command into the history list

}