과목 명: 시스템프로그래밍

담당 교수 명: 소 정 민

<<Assignment 1>>

**서강대학교 컴퓨터공학과**

**[20141515]**

**[김연후]**

목 차

1. 프로그램 개요 2-3

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: addnode() 5

3.2.1 기능 5

3.2.2 사용 변수 5

3.3 모듈 이름: help() 5

3.3.1 기능 5

3.3.2 사용변수 5

3.4 모듈 이름: dir() 5

3.4.1 기능 5

3.4.2 사용변수 5

3.5 모듈 이름: history() 5

3.5.1 기능 5

3.5.2 사용변수 6

3.6 모듈 이름: charTohex() 6

3.6.1 기능 6

3.6.2 사용변수 6

3.7 모듈 이름: TwoInputsString() 6

3.7.1 기능 6

3.7.2 사용변수 6

3.8 모듈 이름: dump() 6

3.8.1 기능 6

3.8.2 사용변수 6

3.9 모듈 이름: ThreeInputsString() 6

3.9.1 기능 6

3.9.2 사용변수 6

3.10 모듈 이름: hashfunc() 7

3.10.1 기능 7

3.10.2 사용변수 7

3.11 모듈 이름: getOpcode() 7

3.11.1 기능 7

3.11.2 사용변수 7

3.12 모듈 이름: makeHashTable() 7

3.12.1 기능 7

3.12.2 사용변수 7

4. 전역 변수 정의 7

4.1 Node \*head 7

4.2 Node \*rear 7

# 프로그램 개요

여러 가지 명령어를 입력 받으며, 각각의 명령에 대한 기능을 수행하고, 잘못 된 명령어를 입력 받았을 땐, 예외처리를 하여 실행하지 않는다.

명령어로는 h[elp], d[ir], q[uit], hi[story], du[mp], e[dit], f[ill], opcode, opcodelist 가 있다.

# 프로그램 설명

## 프로그램 흐름도

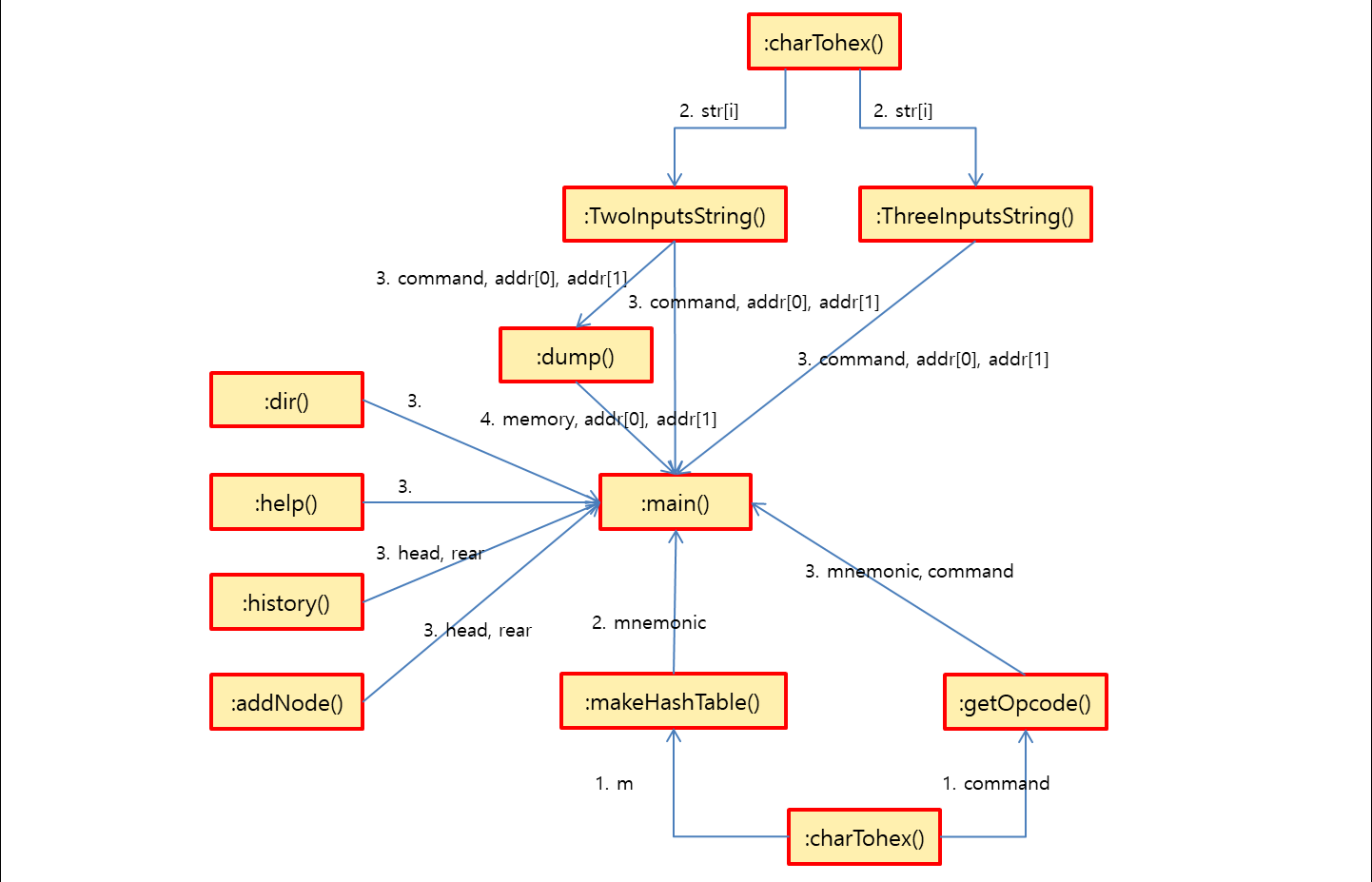


그림 1> 프로그램 흐름도

프로그램을 구성하는 모듈의 구조를 도식하고, 각 모듈간의 상호작용에 사용되는 데이터에 대해서 기술한다. 각각의 상호작용의 흐름은 번호를 부여하여 도식한다.

# 모듈 정의

## 모듈 이름 : main()

### 기능

명령어를 입력 받아서 해당 기능을 작동하거나 출력해주는 함수를 호출한다.

### 사용 변수

Unsigned char \* memory : dump 기능에서 사용 될 메모리 값들을 저장하는 배열

HashTable \*\* mnemonic : HashTable 기능을 이용한 opcode 와 mnemonic 이 저장된 구조체

Int sAddr : 시작 주소값

Int addr[3] : input 값들을 저장하는 변수

## 모듈 이름: addNode()

### 기능

Valid command 가 입력 될 시 history 기능에서 사용하기 위한 linked list 구조에 command string을 추가한다.

### 사용 변수

Node \*head : linked list 구조의 머리 부분에 해당된다.

Node \* rear : linked list 구조의 꼬리 부분에 해당된다.–

## 모듈이름: help()

### 기능

현재의 프로그램이 지원하는 모든 명령어 종류들을 출력하는 기능을 갖는다.

### 사용변수

없음

## 모듈이름: dir()

### 기능

현재의 directory에 있는 모든 파일들을 보여준다.

### 사용변수

Dir \*dir : Directory 변수

Struct dirent \*dire : 현재의 directory에 있는 파일 이름들의 정보를 갖는 구조체

Struct stat statbuf : 현재의 directory 에 있는 파일들의 권한 정보를 갖는 구조체

## 모듈이름: history()

### 기능

현재까지 입력되어 실행된 Valid Command 들을 출력한다.

### 사용변수

Node \*head : linked list 구조의 머리 부분에 해당된다.

## 모듈이름: charTohex()

### 기능

문자를 16진법의 숫자로 바꿔준다.

### 사용변수

Char c : 명령어에서 input에 해당되는 부분의 문자

## 모듈이름: TwoInputsString()

### 기능

입력 된 Command 에서 Valid 할 경우 dump or edit 에서 쓰일 2개의 input을 걸러내는 기능을 한다.

Invalid Command를 걸러내는 기능도 한다.

### 사용변수

Int startSize : 첫 번째 input의 자리 수

Int endSize : 두 번째 input 의 자리 수

Int comaFlag : 쉼표(,)의 포함 여부 (command string 에 있으면 1, 없으면 0)

Int spaceFlag : 첫 번째 input과 쉼표(,) 사이의 space(공백) 존재 여부 (있으면1, 없으면0)

Int f : 문자에서 정수로 변환 된 첫 번째 input

Int s : 문자에서 정수로 변환 된 두 번째 input

## 모듈이름: dump()

### 기능

Command dump 의 기능을 한다. 메모리 값들을 출력해준다.

### 사용변수

Int eAddr : 마지막 주소 값을 갖는다.

Int start : 한 줄의 시작 기준 주소 값을 가지며, 계속 변동된다.

Int tmps : 명령어로 입력 받은 시작 주소 값을 가지며 변하지 않는다.

## 모듈이름: ThreeInputsString()

### 기능

입력 된 Command 에서 Valid 할 경우 fill 에서 쓰일 3개의 input을 잡아내는 기능을 한다.

Invalid Command를 걸러내는 기능도 한다.

### 사용변수

Int startSize : 첫 번째 input의 자리 수

Int endSize : 두 번째 input 의 자리 수

Int comaFlag : 쉼표(,)의 포함 여부 (command string 에 있으면 1, 없으면 0)

Int spaceFlag : 첫 번째 input과 쉼표(,) 사이의 space(공백) 존재 여부 (있으면1, 없으면0)

Int f : 문자에서 정수로 변환 된 첫 번째 input

Int s : 문자에서 정수로 변환 된 두 번째 input

Int t : 문자에서 정수로 변환 된 세 번째 input

## 모듈이름: hashfunc()

### 기능

Hashtable 에 쓰이는 저장방식을 함수화 하는 기능을 갖는다.

Opcode mnemonic의 각 문자의 ASCII code 값들의 합을 20으로 나눈 나머지를 hashtable의 index로 갖는다.

### 사용변수

Char \*str : main()에서 주는 mnemonic string

Int res : Opcode mnemonic의 각 문자의 ASCII code 값들의 합

## 모듈이름: getOpcode()

### 기능

HashTable 에서 해당 mnemonic이 포함되는 index를 찾아서 mnemonic string 비교를 통해 mnemonic이 갖는 index 값을 가져온다. 없을 시 -1을 return 한다.

### 사용변수

HashTable \*temp : 해당 index의 head 부분의 주소 값을 갖는다.

## 모듈이름: makeHashTable()

### 기능

Opcode mnemonic의 각 문자의 ASCII code 값들의 합을 20으로 나눈 나머지를 hashtable의 index로 하여 만든 HashTable을 만드는 기능을 한다.

### 사용변수

FILE \*fp : opcode.txt를 읽어오기 위한 파일 변수

Char m[10] : opcode.txt에서 mnemonic문자를 가져와 받는다.

Char tmp : opcode.txt에서 getChar()를 이용하여 한 문자씩 가져와서 받는 변수

Int code : opcode.txt에서 opcode 숫자를 가져와 받는다.

# 전역 변수 정의

## Node \*head

명령어 history가 저장 된 linked list 구조의 제일 앞 부분 주소 값을 저장한다.

## Node \*rear

명령어 history가 저장 된 linked list 구조의 제일 뒷 부분 주소 값을 저장한다.

.

# 코드

/\*헤더 파일\*/

/\*20141515.h\*/

#pragma once

#ifndef \_\_20141515\_\_

#define \_\_20141515\_\_

/\*포함되는 파일\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/stat.h>

#include"dirent.h"

/\*구조체\*/

typedef struct Node\_ {

char strcmd[500];

struct Node\_ \*link;

}Node;

typedef struct HashTable\_ {

char mne[10];

int opcode;

struct HashTable\_ \*link;

}HashTable;

/\*변수\*/

Node \*head;

Node \*rear;

/\*함수 원형\*/

void addNode(char \*cmd);

void help();

void dir();

void history();

int charTohex(char c);

int TwoInputsString(char \*str, int \*first, int \*second);

int dump(unsigned char \*mem, int s, int e);

int ThreeInputsString(char \*str, int \*first, int \*second, int \*third);

int hashfunc(char \*str);

int getOpcode(HashTable \*\*mne, char \*str, int index);

#endif

/\*20141515.c\*/

#include "20141515.h"

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

void addNode(char \*cmd) { // when valid command is excuted add the command string to the linked list (for history)

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

strcpy(n->strcmd, cmd);

n->link = NULL;

if (head == NULL)

head = n;

else

rear->link = n;

rear = n;

}

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

/\*함수 : addNode(char \*cmd)

/\*목적 : 사용 Valid command 가 입력 될 시 history 기능에서 사용하기 위한 linked list 구조에 command

string을 추가한다. \*/

/\*리턴값 : 없음\*/

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

void help() { // display all commands supported by this program

printf("\n");

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\n");

}

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

/\*함수 : help()\*/

/\*목적 : 현재의 프로그램이 지원하는 모든 명령어 종류들을 출력하는 기능을 갖는다. \*/

/\*리턴값 : 없음\*/

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

void dir() { // shows the files in the current directory

DIR \*dir;

struct dirent \*dire;

struct stat statbuf;

dir = opendir("./");

printf("\n");

if (dir != NULL) {

readdir(dir);

readdir(dir);

while ((dire = readdir(dir)) != NULL) {

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

if (stat(dire->d\_name, &statbuf) < 0) {

perror("stat error\n");

return;

}

if (S\_ISDIR(statbuf.st\_mode)) // if it's a directory

printf("/");

else if ((statbuf.st\_mode & S\_IXUSR) != 0 || (statbuf.st\_mode & S\_IXGRP) != 0 || (statbuf.st\_mode & S\_IXOTH) != 0) // if file has execute permission (which means it's executable file)

printf("\*");

printf("\n");

}

closedir(dir);

}

else

perror("\nerror on directory output...(d[ir])\n\n");

printf("\n");

}

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

/\*함수 : dir()\*/

/\*목적 : 현재의 directory에 있는 모든 파일들을 보여준다. \*/

/\*리턴값 : 없음\*/

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

void history() { // print out history of all valid commands excuted

Node \*temp = head;

int i = 1;

printf("\n");

while (temp != NULL) {

printf("%d\t%s\n", i, temp->strcmd);

temp = temp->link;

i++;

}

printf("\n");

}

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

/\*함수 : history()\*/

/\*목적 : 현재까지 입력되어 실행된 Valid Command 들을 출력한다. \*/

/\*리턴값 : 없음\*/

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

int charTohex(char c) { // change character to hexadecimal

if (c >= '0'&&c <= '9')

return c - '0';

else if (c >= 'A'&&c <= 'F')

return c - 'A' + 10;

else if (c >= 'a'&&c <= 'f')

return c - 'a' + 10;

else

return -1;

}

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

/\*함수 : charTohex(char c) \*/

/\*목적 : 문자를 16진법의 숫자로 바꿔준다. \*/

/\*리턴값 : 16진수 or -1\*/

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

int TwoInputsString(char \*str, int \*first, int \*second) { // get two inputs from str (used for du[mp] and e[dit])

int i = 0, startSize = 0, comaFlag = 0, endSize = 0, spaceFlag = 0, f = 0, s = 0;

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

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

if ((str[i] >= '0'&&str[i] <= '9') || (str[i] >= 'A'&&str[i] <= 'F') || (str[i] >= 'a'&&str[i] <= 'f') || (str[i] == ',') || (str[i] == '0' && (str[i + 1] == 'X' || str[i + 1] == 'x'))) {

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

if (startSize == 0) { // command , ...

printf("\nWrong Command!!!\n\n");

return 0;

}

else { // command start, ...

if (str[i + 1] == '\0') { // command start, (error)

printf("\nWrong Command!!!\n\n");

return 0;

}

comaFlag++;

spaceFlag = 0;

if(comaFlag >= 2){ // when there are more inputs (error)

printf("\nWrong Command!!!\n\n");

return 0;

}

}

}

else if (str[i] == '0' && (str[i + 1] == 'X' || str[i + 1] == 'x') && str[i-1] != '0') {

if (!((str[i + 2] >='0' && str[i + 2] <= '9') || (str[i + 2] >= 'A' && str[i + 2] <= 'F') || (str[i + 2] >= 'a' && str[i + 2] <= 'f'))) { // command start, 0x (end error)

printf("\nWrong Command!!!\n\n");

return 0;

}

i++;

}

else if ((str[i] >= '0'&&str[i] <= '9') || (str[i] >= 'A'&&str[i] <= 'F') || (str[i] >= 'a'&&str[i] <= 'f')) {

if (str[i + 1] == '\0' && comaFlag == 0) // command start

s = -1;

if (comaFlag == 0) {

f = 16 \* f + charTohex(str[i]);

startSize++;

}

if (comaFlag == 1) {

s = 16 \* s + charTohex(str[i]);

if (s > 0xFFFFF) {

s = 0xFFFFF;

break;

}

endSize++;

}

if (spaceFlag == 1 && (startSize != 0 || endSize != 0)) { // command s tart... or dump stard end (errors)

printf("\nWrong Command!!! (Input error)\n\n");

return 0;

}

}

}

else {

printf("\nWrong Command!!! (Input error)\n\n"); // other words in string (error) com mand (error)

return 0;

}

}

else {

if (startSize != 0 && comaFlag == 0)

spaceFlag = 1;

if (endSize != 0) { // command start, e nd (error)

printf("\nWrong Command!!! (Input error)\n\n");

return 0;

}

}

i++;

}

\*first = f;

if (s != -1)

\*second = s;

else

\*second = -1;

return 1;

}

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

/\*함수 : TwoInputsString(char \*str, int \*first, int \*second) \*/

/\*목적 : 입력 된 Command 에서 Valid 할 경우 dump or edit 에서 쓰일 2개의 input을 걸러내는 기능을 한다.

Invalid Command를 걸러내는 기능도 한다. \*/

/\*리턴값 : 1 or 0\*/

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

int dump(unsigned char \*mem, int s, int e) { // outputs for command d[ump]

int eAddr, start, tmps;

printf("\n");

if (s <= e || e == -1) { // print address from start to end (when e == -1 there is no end address)

int i = 0;

if (e == -1)

e = s + 159;

if (e > 0xFFFFF)

e = 0xFFFFF;

eAddr = e;

start = s;

while (start >= 0) {

start -= 16;

i++;

}

i--;

start = 0;

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

start += 16;

tmps = start;

for (int j = 0; j < (e - tmps) / 16 + 1; j++) {

if (tmps <= s && s <= e && e <= tmps + 15) {

printf("%05X ", start);

for (int k = tmps; k < s; k++)

printf(" ");

for (int k = s; k <= e; k++)

printf("%02X ", mem[k]);

for (int k = e + 1; k <= tmps + 15; k++)

printf(" ");

printf("; ");

for (int k = 0; k < 16; k++)

printf(".");

printf("\n");

break;

}

if (j < (e - tmps) / 16) {

printf("%05X ", start);

for (int k = 0; k < 16; k++) {

if (j == 0 && k < s - (s / 16) \* 16)

printf(" ");

else

printf("%02X ", mem[tmps + 16 \* j + k]);

}

printf("; ");

for (int k = 0; k < 16; k++) {

if (mem[tmps + 16 \* j + k] < 0x20 || mem[tmps + 16 \* j + k]>0x7E)

printf(".");

else

printf("%c", mem[tmps + 16 \* j + k]);

}

printf("\n");

start += 16;

}

if (j == (e - tmps) / 16 && e - (e / 16) \* 16 >= 0) {

printf("%05X ", start);

for (int k = 0; k < 16; k++) {

if (k <= e - (e / 16) \* 16)

printf("%02X ", mem[tmps + 16 \* j + k]);

else

printf(" ");

}

printf("; ");

for (int k = 0; k < 16; k++) {

if (k <= e - (e / 16) \* 16) {

if (mem[tmps + 16 \* j + k] < 0x20 || mem[tmps + 16 \* j + k]>0x7E)

printf(".");

else

printf("%c", mem[tmps + 16 \* j + k]);

}

else

printf(".");

}

printf("\n");

}

}

}

else {

eAddr = 0;

printf("Wrong Command!!!(start address must be smaller than end address...)\n");

}

printf("\n");

return eAddr;

}

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

/\*함수 : dump(unsigned char \*mem, int s, int e)\*/

/\*목적 : Command dump 의 기능을 한다. 메모리 값들을 출력해준다. \*/

/\*리턴값 : 1 or 0\*/

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

int ThreeInputsString(char \*str, int \*first, int \*second, int \*third) { // get three inputs from str (used for f[ill])

int i = 0, firstSize = 0, secondSize = 0, thirdSize = 0, comaFlag = 0, spaceFlag = 0, f = 0, s = 0, t = 0;

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

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

if ((str[i] >= '0'&&str[i] <= '9') || (str[i] >= 'A'&&str[i] <= 'F') || (str[i] >= 'a'&&str[i] <= 'f') || (str[i] == ',') || (str[i] == '0' && (str[i + 1] == 'X' || str[i + 1] == 'x'))) {

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

if (firstSize == 0) { // command , ...

printf("\nWrong Command!!!\n\n");

return 0;

}

else { // command start, ...

if (str[i + 1] == '\0') { // command start, (error)

printf("\nWrong Command!!!\n\n");

return 0;

}

comaFlag++;

spaceFlag = 0;

if (comaFlag >= 3){

printf("\nWrong Command!!!\n\n");

return 0;

}

}

}

else if (str[i] == '0' && (str[i + 1] == 'X' || str[i + 1] == 'x')&& str[i-1] != '0') {

if (!((str[i + 2] >= '0' && str[i + 2] <= '9') || (str[i + 2] >= 'A' && str[i + 2] <= 'F') || (str[i + 2] >= 'a' && str[i + 2] <= 'f'))) { // command start, 0x (end error)

printf("\nWrong Command!!!\n\n");

return 0;

}

i++;

}

else if ((str[i] >= '0'&&str[i] <= '9') || (str[i] >= 'A'&&str[i] <= 'F') || (str[i] >= 'a'&&str[i] <= 'f')) {

if (comaFlag == 0) {

f = 16 \* f + charTohex(str[i]);

firstSize++;

}

if (comaFlag == 1) {

s = 16 \* s + charTohex(str[i]);

secondSize++;

}

if (comaFlag == 2) {

t = 16 \* t + charTohex(str[i]);

thirdSize++;

}

if (spaceFlag == 1 && (firstSize != 0 || secondSize != 0 || thirdSize != 0)) { // command s tart... or dump stard end (errors)

printf("\nWrong Command!!! (Input error)\n\n");

return 0;

}

}

}

else {

printf("\nWrong Command!!! (Input error)\n\n"); // other words in string (error) com mand (error)

return 0;

}

}

else {

if ((firstSize != 0 && comaFlag == 0) || (secondSize != 0 && comaFlag == 1))

spaceFlag = 1;

if (thirdSize != 0) { // command start, e nd (error)

printf("\nWrong Command!!! (Input error)\n\n");

return 0;

}

}

i++;

}

if (firstSize == 0 || secondSize == 0 || thirdSize == 0) {

printf("\nWrong Command!!!(Need to write 3 inputs)\n\n");

return 0;

}

\*first = f;

\*second = s;

\*third = t;

if(f>s){

printf("\nWrong Command!!!(start address must be smaller than end address...)\n\n");

return 0;

}

return 1;

}

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

/\*함수 : ThreeInputsString(char \*str, int \*first, int \*second, int \*third) \*/

/\*목적 : 입력 된 Command 에서 Valid 할 경우 fill 에서 쓰일 3개의 input을 잡아내는 기능을 한다.

Invalid Command를 걸러내는 기능도 한다. \*/

/\*리턴값 : 1 or 0\*/

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

int hashfunc(char \*str) { // hashfunction : Returns the remainder value of res(sum of values of each characters ASCII code in string) divided by 20

int i = 0, res = 0;

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

res += (int)str[i];

i++;

}

return res % 20;

}

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

/\*함수 : hashfunc(char \*str) \*/

/\*목적 : Hashtable 에 쓰이는 저장방식을 함수화 하는 기능을 갖는다.

Opcode mnemonic의 각 문자의 ASCII code 값들의 합을 20으로 나눈 나머지를 hashtable의 index로 갖는다. \*/

/\*리턴값 : res%20\*/

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

int getOpcode(HashTable \*\* mne, char \*str, int index) { // get (str)'s opcode from hashtable

HashTable \*temp = mne[index];

while (temp != NULL) {

if (strcmp(temp->mne, str) == 0)

return temp->opcode;

temp = temp->link;

}

return -1;

}

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

/\*함수 : getOpcode(HashTable \*\* mne, char \*str, int index) \*/

/\*목적 : HashTable 에서 해당 mnemonic이 포함되는 index를 찾아서 mnemonic string 비교를 통해

mnemonic이 갖는 index 값을 가져온다. 없을 시 -1을 return 한다.\*/

/\*리턴값 : temp->opcode or -1\*/

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

void makeHashTable(HashTable \*\*mne) { // make hashtable of opcode

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

char m[10], tmp;

int code, i, j;

tmp = fgetc(fp);

while (tmp != EOF) {

i = 0; j = 0; code = 0;

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

m[k] = 0;

while (tmp != '\n') {

if (i < 2) {

if (tmp >= '0'&&tmp <= '9')

code = code \* 16 + tmp - '0';

if (tmp >= 'A'&&tmp <= 'F')

code = code \* 16 + tmp - 'A' + 10;

i++;

}

else {

if (tmp >= 'A'&&tmp <= 'Z') {

m[j] = tmp;

j++;

}

}

tmp = fgetc(fp);

}

HashTable \*temp, \*node;

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

strcpy(temp->mne, m);

temp->opcode = code;

temp->link = NULL;

if (mne[hashfunc(m)] == NULL)

mne[hashfunc(m)] = temp;

else {

node = mne[hashfunc(m)];

while (node->link != NULL)

node = node->link;

node->link = temp;

}

tmp = fgetc(fp);

}

}

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

/\*함수 : makeHashTable(HashTable \*\*mne) \*/

/\*목적 : Opcode mnemonic의 각 문자의 ASCII code 값들의 합을 20으로 나눈 나머지를 hashtable의 index로 하여 만든 HashTable을 만드는 기능을 한다.\*/

/\*리턴값 : 없음\*/

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

int main() {

char \*str;

unsigned char \*memory = (unsigned char\*)malloc(0x100000 \* sizeof(unsigned char));

int sAddr = 0, addr[3];

HashTable \*\*mnemonic = (HashTable\*\*)malloc(20 \* sizeof(HashTable\*));

for (int a = 0; a < 0x100000; a++)

memory[a] = 0x00;

for (int a = 0; a < 20; a++) {

mnemonic[a] = (HashTable\*)malloc(sizeof(HashTable));

mnemonic[a] = NULL;

}

makeHashTable(mnemonic);

head = NULL;

while (1) { // program progress

printf("sicsim> ");

char \*command = (char\*)malloc(500 \* sizeof(char));

scanf("%[^\n]", command);

str = command;

//////////////////////////////////////////////////////////// erasing indent, space character from both left and right side of the string

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

int i = 1;

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

i++;

command = &command[i];

}

if (command[strlen(command) - 1] == ' ' || command[strlen(command) - 1] == '\t') {

int i = (int)strlen(command) - 2;

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

i--;

command[i + 1] = '\0';

}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

if (strcmp(command, "h") == 0 || strcmp(command, "help") == 0) { // h[elp]

addNode(str);

help();

}

else if (strcmp(command, "q") == 0 || strcmp(command, "quit") == 0) // q[uit]

break;

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

addNode(str);

dir();

}

else if (strcmp(command, "hi") == 0 || strcmp(command, "history") == 0) { // hi[story]

addNode(str);

history();

}

else if (command[0] == 'd' && command[1] == 'u') { // du[mp]

if (command[2] == 'm' && command[3] == 'p') {

if (command[4] == ' ' || command[4] == '\0' || command[4] == '\t') {

if (command[4] == ' ' || command[4] == '\t') { // dump A,B

if (TwoInputsString(&command[5], &addr[0], &addr[1]) == 1) {

if (addr[0] >= 0x00000 && addr[0] <= 0xFFFFF) {

if (addr[1] > 0xFFFFF)

addr[1] = 0xFFFFF;

addNode(str);

sAddr = dump(memory, addr[0], addr[1]) + 1;

}

else

printf("\nWrong Command!!!(input is outside the bounds)\n\n");

}

}

else { // just dump

if (sAddr < 0x100000) {

addNode(str);

sAddr = dump(memory, sAddr, -1) + 1;

}

else

printf("\nWrong Command!!! (start address is out of range(0 ~ FFFFF))\n\n");

}

}

else

printf("\nWrong Command!!!\n\n");

}

else if (command[2] == ' ' || command[2] == '\0' || command[2] == '\t') {

if (command[2] == ' ' || command[2] == '\t') { // du A,B

if (TwoInputsString(&command[3], &addr[0], &addr[1]) == 1) {

if (addr[0] >= 0x00000 && addr[0] <= 0xFFFFF) {

if (addr[1] > 0xFFFFF)

addr[1] = 0xFFFFF;

addNode(str);

sAddr = dump(memory, addr[0], addr[1]) + 1;

}

else

printf("\nWrong Command!!!(input is outside the bounds)\n\n");

}

}

else { // just du

if (sAddr < 0x100000) {

addNode(str);

sAddr = dump(memory, sAddr, -1) + 1;

}

else

printf("\nWrong Command!!! (start address is out of range(0 ~ FFFFF))\n\n");

}

}

else

printf("\nWrong Command!!!\n\n");

}

else if (command[0] == 'e') { // e[dit]

if (command[1] == 'd' && command[2] == 'i' && command[3] == 't') {

if (command[4] == ' ' || command[4] == '\t') { // edit A,B

if (TwoInputsString(&command[5], &addr[0], &addr[1]) == 1) {

if (addr[0] >= 0x00000 && addr[0] <= 0xFFFFF && addr[1] >= 0x00 && addr[1] <= 0xFF) {

addNode(str);

memory[addr[0]] = addr[1];

printf("\n");

}

else

printf("\nWrong Command!!!(input is outside the bounds)\n\n");

}

}

else

printf("\nWrong Command!!!\n\n");

}

else if (command[1] == ' ' || command[1] == '\0' || command[1] == '\t') {

if (command[1] == ' ' || command[1] == '\t') { // e A,B

if (TwoInputsString(&command[2], &addr[0], &addr[1]) == 1) {

if (addr[0] >= 0x00000 && addr[0] <= 0xFFFFF && addr[1] >= 0x00 && addr[1] <= 0xFF) {

addNode(str);

memory[addr[0]] = addr[1];

printf("\n");

}

else

printf("\nWrong Command!!!(input is outside the bounds)\n\n");

}

}

else

printf("\nWrong Command!!!\n\n");

}

else

printf("\nWrong Command!!!\n\n");

}

else if (command[0] == 'f') { // f[ill]

if (command[1] == 'i' && command[2] == 'l' && command[3] == 'l') {

if (command[4] == ' ' || command[4] == '\t') { // fill A,B,C

if (ThreeInputsString(&command[5], &addr[0], &addr[1], &addr[2]) == 1) {

if (addr[0] >= 0x00000 && addr[0] <= 0xFFFFF && addr[1] >= 0x00000 && addr[1] <= 0xFFFFF && addr[2] >= 0x00 && addr[2] <= 0xFF) {

addNode(str);

for (int j = addr[0]; j <= addr[1]; j++)

memory[j] = addr[2];

printf("\n");

}

else

printf("\nWrong Command!!!(input is outside the bounds)\n\n");

}

}

else

printf("\nWrong Command!!!\n\n");

}

else if (command[1] == ' ' || command[1] == '\0' || command[1] == '\t') {

if (command[1] == ' ' || command[1] == '\t') { // f A,B,C

if (ThreeInputsString(&command[2], &addr[0], &addr[1], &addr[2]) == 1) {

if (addr[0] >= 0x00000 && addr[0] <= 0xFFFFF && addr[1] >= 0x00000 && addr[1] <= 0xFFFFF && addr[2] >= 0x00 && addr[2] <= 0xFF) {

addNode(str);

for (int j = addr[0]; j <= addr[1]; j++)

memory[j] = addr[2];

printf("\n");

}

else

printf("\nWrong Command!!!(input is outside the bounds)\n\n");

}

}

else

printf("\nWrong Command!!!\n\n");

}

else

printf("\nWrong Command!!!\n\n");

}

else if (strcmp(command, "reset") == 0) { // reset (make all values in the memory 0)

addNode(str);

for (int j = 0x00000; j <= 0xFFFFF; j++)

memory[j] = 0;

printf("\n");

}

else if (strcmp(command, "opcodelist") == 0) { // print out opcodelist by hashtable

addNode(str);

printf("\n");

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

printf("%d : ", i);

HashTable \*temp;

temp = mnemonic[i];

while (temp != NULL) {

printf("[%s, %02X]", temp->mne, temp->opcode);

if (temp->link != NULL)

printf(" -> ");

temp = temp->link;

}

printf("\n");

}

printf("\n");

}

else if (command[0] == 'o' && command[1] == 'p' && command[2] == 'c' && command[3] == 'o' && command[4] == 'd' && command[5] == 'e') { // opcode command

int i = 6;

//////////////////////////////////////////////////////////// erasing indent, space character from both left and right side of the string

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

i++;

command = &command[i];

if (command[strlen(command) - 1] == ' ' || command[strlen(command) - 1] == '\t') {

int i = (int)strlen(command) - 2;

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

i--;

command[i + 1] = '\0';

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

i = getOpcode(mnemonic, command, hashfunc(command));

if (i != -1) {

addNode(str);

printf("opcode is %02X.\n\n", i);

}

else

printf("Invalid mnemonic... opcode doesn't exist!\n\n");

}

else

printf("\nWrong Command!!!\n\n");

getchar();

free(str);

}

free(memory); // free

for (int a = 0; a < 20; a++) {

HashTable \*temp;

while (mnemonic[a] != NULL) {

temp = mnemonic[a];

mnemonic[a] = mnemonic[a]->link;

free(temp);

}

free(mnemonic[a]);

}

free(mnemonic);

while (head != NULL) {

Node \*temp = head;

head = head->link;

free(temp);

}

return 0;

}

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

/\*함수 : main()\*/

/\*목적 : command를 입력 받아 각각의 기능을 할 수 있도록 해당 모듈들을 호출하는 기능을 한다.\*/

/\*리턴값 : 0\*/

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