과목명 : 시스템 프로그래밍

담당 교수명 : 김지환

<<Assignment 2>>

서강대학교 컴퓨터공학과

학번: 20171690

이름: 정유석

**목차**

1. 프로그램 개요
2. 프로그램 설명
   1. 프로그램 흐름도
   2. SIC/XE Assembler 흐름도
3. 분할 c 파일
   1. 20171690.c
   2. shell.c
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : type()
            1. 기능
            2. 사용 변수
   3. memory.c
   4. opcode.c
   5. assembler.c
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : assemble()
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : pass1()
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : pass2()
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : getInstructionSize()
            1. 기능
            2. 사용 변수
         5. 모듈 이름 : getObjCode()
            1. 기능
            2. 사용 변수
         6. 모듈 이름 : printObjCode()
            1. 기능
            2. 사용 변수
         7. 모듈 이름 : tokenizeAsmFile()
            1. 기능
            2. 사용 변수
         8. 모듈 이름 : removeSpaceAroundComma()
            1. 기능
            2. 사용 변수
   6. assembler\_functions.c
      1. 설명
      2. 모듈 정의
         1. 모듈 이름 : getRegNum()
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : LDB()
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : printLineinLST()
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : isWhiteSpace()
            1. 기능
            2. 사용 변수
         5. 모듈 이름 : isBlankLine()
            1. 기능
            2. 사용 변수
         6. 모듈 이름 : isDirective()
            1. 기능
            2. 사용 변수
         7. 모듈 이름 : isComma()
            1. 기능
            2. 사용 변수
         8. 모듈 이름 : nameToListing()
            1. 기능
            2. 사용 변수
         9. 모듈 이름 : nameToObj()
            1. 기능
            2. 사용 변수
         10. 모듈 이름 : toUpper()
             1. 기능
             2. 사용 변수
         11. 모듈 이름 : toUpperCase()
             1. 기능
             2. 사용 변수
         12. 모듈 이름 : byteSize()
             1. 기능
             2. 사용 변수
         13. 모듈 이름 : wordSize()
             1. 기능
             2. 사용 변수
         14. 모듈 이름 : resbSize()
             1. 기능
             2. 사용 변수
         15. 모듈 이름 : reswSize()
             1. 기능
             2. 사용 변수
         16. 모듈 이름 : addNum()
             1. 기능
             2. 사용 변수
         17. 모듈 이름 : freeNums()
             1. 기능
             2. 사용 변수
         18. 모듈 이름 : printNums()
             1. 기능
             2. 사용 변수
         19. 모듈 이름 : enqueue()
             1. 기능
             2. 사용 변수
         20. 모듈 이름 : dequeue()
             1. 기능
             2. 사용 변수
         21. 모듈 이름 : freeQueue()
             1. 기능
             2. 사용 변수
   7. symbol.c
      1. 설명
      2. 모듈 설명
         1. 모듈 이름 : symbol()
            1. 기능
            2. 사용 변수
         2. 모듈 이름 : addSym()
            1. 기능
            2. 사용 변수
         3. 모듈 이름 : freeSymTab()
            1. 기능
            2. 사용 변수
         4. 모듈 이름 : symHashFunc()
            1. 기능
            2. 사용 변수
         5. 모듈 이름 : findSym()
            1. 기능
            2. 사용 변수
         6. 모듈 이름 : isStr()
            1. 기능
            2. 사용 변수
         7. 모듈 이름 : getBiggerStr()
            1. 기능
            2. 사용 변수
         8. 모듈 이름 : getMaxofFour()
            1. 기능
            2. 사용 변수
         9. 모듈 이름 : getMaxofThree()
            1. 기능
            2. 사용 변수
         10. 모듈 이름 : printSymbol()
             1. 기능
             2. 사용 변수
4. 구조체 정의
   1. HISTORY
   2. opNode
   3. symNode
   4. numNode
   5. tRecord
5. 전역 변수 정의
   1. HISTORY\* hisHead
   2. opNode\*\* opTable
   3. unsigned char\* MEMORY
   4. int END\_ADDR
   5. symNode\*\* SYMTAB
   6. numNode\* numHead
   7. tRecord\* tRHead
   8. tRTail
6. 코드

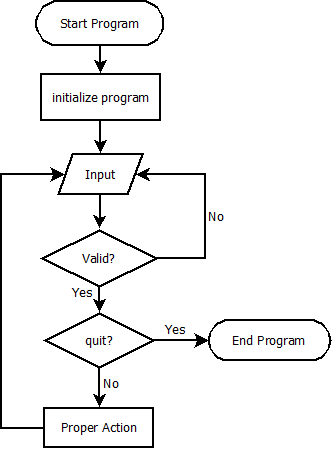
1. 프로그램 개요

프로젝트 1에서 Shell을 구현하여 메모리를 할당하고, opcode table 등을 생성하였다. 이번 프로젝트에서는 구현한 shell에서 실행할 SIC/XE Machine 어셈블러를 구현하였다. assembly program source 파일을 입력받아 object file과 listing file을 생성한다. 이 때 어셈블 과정에서 symbol table을 생성한다.

보고서에는 변경된 내용이나 새로 추가된 내용에 대해서만 기술하였다.

2. 프로그램 설명

2.1 프로그램 흐름도



< SIC Shell Program Flow Chart >

3. 분할 c파일

프로젝트 1에서 수정된 c 파일은 shell.c 파일이다. type() 함수를 추가하였다. 또한 이번 프로젝트의 목적인 어셈블러 기능을 구현한 assemble.c, 어셈블러 기능을 추가하기 위해 작성한 함수들을 모아 놓은 assembler\_function.c, 그리고 assemble 과정에서 생성하는 symbol table과 관련된 기능을 구현한 symbol.c 파일이 새로 추가되었다.

**3.1 20171690.c**

**3.2 shell.c**

3.2.1 설명

Shell 관련 명령어들과 관련된 필요한 기능들을 구현한 파일이다. 이번 프로젝트에서 구현해야 할 shell 관련 명령어는 h[elp], d[ir], hi[story], q[uit] 으로 4개이다.

3.2.2 모듈 정의

3.2.2.1 모듈 이름 : **help()**

3.2.2.1.1 기능

Shell에서 실행 가능한 모든 명령어들의 리스트를 화면에 출력한다. 단순 printf를 활용하여 출력하였다.

3.2.2.1.2 사용 변수

없음

3.2.2.2 모듈 이름 : **directory()**

3.2.2.2.1 기능

현재 디렉터리에 있는 모든 파일을 출력한다. 이 때 디렉터리는 ‘/’ 표시를, 실행파일은 ‘\*’ 표시를 한다.

3.2.2.2.2 사용 변수

**struct dirent\* dirEntry** – 현재 디렉터리에 속하는 파일들의 정보를 저장하는 dirent 구조체 포인터이다. 구조체의 정보 중 파일의 이름을 출력하기 위해 d\_name 변수를 사용한다.

**struct stat fileInfo** – 파일의 상태를 나타내는 stat 구조체 변수이다. 읽어온 파일이 디렉터리이거나 실행파일인 경우, 특별한 표시를 하기 위해 사용하였다.

**DIR\* dr** – 디렉터리 검색을 위한 DIR 구조체 포인터 변수이다. opendir(), readdir, closedir() 함수와 함께 사용한다.

3.2.2.3 모듈 이름 : **addHistory()**

3.2.2.3.1 기능

main 함수에서 입력을 받을 때, valid한 input이라면 해당 input을 원형 그대로 history에 저장한다. HISTORY 구조체와 전역 변수에 대한 자세한 설명은 각각 4.1, 5.1에 기술하였다.

3.2.2.3.2 사용 변수

**HISTORY\* pNew** – 새로운 history 구조체를 저장할 노드의 포인터이다. 새로운 노드는 linked list의 끝에 연결되며, head는 제일 먼저 들어온 node를 가리킨다.

3.2.2.4 모듈 이름 : **history()**

3.2.2.4.1 기능

shell 명령어 hi[story]를 입력받았을 때 실행된다. 현재까지 사용한 모든 명령어들은 addHistory 함수를 통해 순서대로 번호와 함께 HISTORY linked list에 저장된다. 이 list에 있는 모든 정보를 순차적으로 출력한다.

3.2.2.4.2 사용 변수

**HISTORY\* pMove** – 노드를 순차적으로 탐색하며 정보를 출력한다. 이 때 순차적으로 노드를 가리키는 구조체 포인터이다.

3.2.2.5 모듈 이름 : **quit()**

3.2.2.5.1 기능

shell 명령여 q[uit]를 입력받았을 때 실행된다. 프로그램을 종료하는 명령어에 해당되며, 따라서 할당하였던 모든 메모리를 해제한다. 이 때 해제하는 메모리는 init() 함수에서 생성했던 opcode table, 1024 byte가 할당된 MEMORY, 그리고 history가 저장된 history linked list이다.

3.2.2.5.2 사용 변수

**HISTORY\* pFree**

**opNode\* opFree, \*optmp**

세 변수 모두 linked list를 탐색하며 해당 노드를 free하기 위해 임시로 선언한 변수들이다.

**3.3 memory.c**

**3.4 opcode.c**

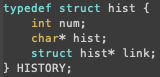
**3.5 assembler.c**

**3.6 assembler\_functions.c**

**3.7 symbol.c**

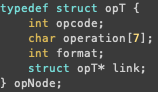
4. 구조체 정의

4.1 **HISTORY**



사용한 명령어를 저장하는 node로 구조는 위와 같다. 명령어와 그 명령어의 순서를 저장하는 변수인 num과 hist, 그리고 linked list로 구현하기 위한 노드 포인터 link로 구성되어있다.

4.2 **opNode**



opcode.txt 파일에서 읽어온 operation에 관한 정보를 저장하는 node이다. opcode와 format을 저장하는 int형 변수, operation 이름을 저장할 operation, 그리고 linked list로 구현하기 위한 node pointer인 link로 구성되어있다.

5. 전역 변수 정의

5.1 **HISTORY\* hisHead**

수행한 명령어들을 저장하는 linked list의 Head를 나타낸다. hisHead를 기준으로 명령어들이 순차적으로 저장되어있다.

5.2 **opNode\*\* opTable**

opcode의 hash table 시작주소를 나타낸다.

5.3 **unsigned char\* MEMORY**

구현할 1MB Memory의 시작주소를 나타낸다.

5.4 **END\_ADDR**

dump명령어를 사용하여 출력한 마지막 값의 주소를 나타낸다. dump 명령어에서 start parameter를 사용하지 않을 경우, END\_ADDR + 1번째 주소부터 값을 출력한다.END\_ADDR이 마지막 주소를 가리킬 경우, 0번지 값부터 출력한다.

**6. 코드**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 20171690.h \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ~ 190325 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include <sys/types.h>

// MACROs

#define MEMORY\_SIZE 1<<20

#define COMMAND\_SIZE 100

#define OPCODE "opcode.txt"

#define ONE\_BYTE (unsigned char)0xFF

// STRUCTUREs

typedef struct hist {

int num;

char\* hist;

struct hist\* link;

} HISTORY;

typedef struct opT {

int opcode;

char operation[7];

int format;

struct opT\* link;

} opNode;

// GLOBAL VARIANTs

HISTORY\* hisHead;

opNode\*\* opTable;

unsigned char\* MEMORY;

int END\_ADDR;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void init();

int findCmd(char\*);

char\* removeSpace(char\*);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SHELL COMMANDS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void help(); // 0x00

void directory(); // 0x01

void history(); // 0x02

void quit(); // 0x03

void addHistory(char\*);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MEMORY COMMANDS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int dump(char\* start, char\* end, int type); // 0x10 ~ 0x12

int edit(char\* address, char \*value); // 0x13 ~ 0x14

int fill(char\* start, char\* end, char \*value); // 0x15

void reset(); // 0x16

int validAddr(int addr);

int validAddrRange(int start, int end);

int strToHex(char\* param);

int isHex(char ch);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\* OPCODE TABLE COMMANDS \*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int opcode(char\* mnemonic); // 0x20

int opcodeList(); // 0x21

int readOpTable();

int hashFunction(char\* op);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 20171690.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 190325 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "20171690.h"

int main() {

init();

char\* input;

char\* input\_formed;

while (1) {

printf("sicsim> ");

// allocate memories

input = (char\*)malloc(COMMAND\_SIZE \* sizeof(char));

input\_formed = (char\*)malloc(COMMAND\_SIZE \* sizeof(char));

// get input

fflush(stdin);

fgets(input, COMMAND\_SIZE, stdin);

if ((int)strlen(input) > COMMAND\_SIZE) {

printf("Too long command!\n");

continue;

}

// remove white spaces

strcpy(input\_formed, removeSpace(input));

char\* cmd = strtok(input\_formed, " \t");

char\* params;

char\* start, \*end, \*addr, \*val;

int before, after;

// if input is white spaces, continue;

if (cmd) {

switch (findCmd(cmd)) {

case 0x00: // h[elp] done

if ( (params = strtok(NULL, "\0")) ) {

printf("h[elp]: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

addHistory(input);

help();

break;

case 0x01: // d[ir] done

if ( (params = strtok(NULL, "\0")) ) {

printf("d[ir]: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

directory();

addHistory(input);

break;

case 0x02: // hi[story] done

if ( (params = strtok(NULL, "\0")) ) {

printf("hi[story]: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

addHistory(input);

history();

break;

case 0x03: // q[uit] done

if ( (params = strtok(NULL, "\0")) ) {

printf("q[uit]: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

free(input);

free(input\_formed);

quit();

return 0;

case 0x10: case 0x11: case 0x12: // du[mp] [start, end]

params = strtok(NULL, "\0");

if (!params) {

// "dump"

addHistory(input);

dump(NULL, NULL, 0);

break;

}

// parameter(s) follow(s)

before = (int)strlen(params);

start = removeSpace(strtok(params, ","));

after = (int)strlen(params);

end = strtok(NULL, "\0");

if (!end) {

if (before != after) {

// "dump start, "

printf("Invalid Syntax. See 'h[elp]'\n");

break;

}

if (!dump(start, NULL, 1)) {

// inavalid addr

break;

}

// "dump start"

addHistory(input);

break;

}

// "dump start, end"

end = removeSpace(end);

if (!dump(start, end, 2)) {

// invalid addr

break;

}

addHistory(input);

break;

case 0x13: case 0x14: // e[dit] address, value

params = strtok(NULL, "\0");

if (!params) {

// no parameters

printf("Syntax Error. See 'h[elp]'\n");

break;

}

addr = removeSpace(strtok(params, ","));

val = strtok(NULL, "\0");

if (!val) {

// no value + invalid sytax

printf("Syntax Error. See 'h[elp]'\n");

break;

}

val = removeSpace(val);

if (!edit(addr, val)) {

// invalid addr & val

break;

}

addHistory(input);

break;

case 0x15: // f[ill] start, end, value

params = strtok(NULL, "\0");

if (!params) {

// No parameters at all

printf("Syntax Error. See 'h[elp]'\n");

break;

}

start = removeSpace(strtok(params, ","));

end = strtok(NULL, ",");

if (!end) {

// "fill start, "

printf("Syntax Error. See 'h[elp]'\n");

break;

}

end = removeSpace(end);

val = strtok(NULL, "\0");

if (!val) {

// "fill start, end, "

printf("Syntax Error. See 'h[elp]'\n");

break;

}

val = removeSpace(val);

if (!fill(start, end, val)) {

// invalid addr & val

break;

}

addHistory(input);

break;

case 0x16: // reset

if ( (params = strtok(NULL, "\0")) ) {

printf("reset: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

reset();

addHistory(input);

printf("Reset Successfully\n");

break;

case 0x20: // opcode mnemonic //done

params = strtok(NULL, "\0");

if (!params) {

// no mnemonic

printf("Invalid Syntax. See 'h[elp]'\n");

break;

}

params = removeSpace(params);

if (opcode(params))

addHistory(input);

else

printf("%s: No matching mnemonic\n", params);

break;

case 0x21: // opcodelist done

if ( (params = strtok(NULL, "\0")) ) {

printf("opcodelist: '%s' is an invalid option. See 'h[elp]'\n", removeSpace(params));

break;

}

if (!opcodeList())

printf("There's no opcode.txt in the directory\n");

else

addHistory(input);

break;

case 0x30:

printf("command not found: %s\n", cmd);

break;

}

}

free(input);

free(input\_formed);

}

return 0;

}

void init() {

// allocate memories

hisHead = NULL;

opTable = (opNode\*\*)malloc(20 \* sizeof(opNode\*));

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

opTable[i] = NULL;

MEMORY = (unsigned char\*)calloc(MEMORY\_SIZE, sizeof(unsigned char));

END\_ADDR = 0;

readOpTable();

}

int findCmd(char\* cmd) {

// get command as a parameter, return proper cmd

if (strcmp(cmd, "help") == 0 || strcmp(cmd, "h") == 0) return 0x00;

if (strcmp(cmd, "dir") == 0 || strcmp(cmd, "d") == 0) return 0x01;

if (strcmp(cmd, "history") == 0 || strcmp(cmd, "hi") == 0) return 0x02;

if (strcmp(cmd, "quit") == 0 || strcmp(cmd, "q") == 0) return 0x03;

if (strcmp(cmd, "dump") == 0 || strcmp(cmd, "du") == 0) return 0x10;

if (strcmp(cmd, "edit") == 0 || strcmp(cmd, "e") == 0) return 0x13;

if (strcmp(cmd, "fill") == 0 || strcmp(cmd, "f") == 0) return 0x15;

if (strcmp(cmd, "reset") == 0) return 0x16;

if (strcmp(cmd, "opcode") == 0) return 0x20;

if (strcmp(cmd, "opcodelist") == 0) return 0x21;

return 0x30;

}

char\* removeSpace(char\* input) {

// remove Spaces at front & at the end

if (!input) return input;

int i;

for (i = 0; input[i] == ' ' || input[i] == '\t' || input[i] == '\n'; i++) ;

input = input + i;

i = 0;

while (input[strlen(input) - 1 - i] == ' ' || input[strlen(input) - 1 - i] == '\n' || input[strlen(input) - 1 - i] == '\t') i++;

input[strlen(input) - i] = '\0';

return input;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* shell.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 190325 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "20171690.h"

void help() {

printf("\t\th[elp]\n\t\td[ir]\n\t\tq[uit]\n\t\thi[story]\n");

printf("\t\tdu[mp] [start, end]\n");

printf("\t\te[dit] address, value\n");

printf("\t\tf[ill] start, end, value\n");

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

printf("\t\topcode mnemonic\n\t\topcodelist\n");

}

void directory() {

struct dirent\* dirEntry; // directory entry Pointer

struct stat fileInfo;

DIR\* dr = opendir(".");

while ( (dirEntry = readdir(dr)) != NULL) {

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

stat(dirEntry->d\_name, &fileInfo);

if (S\_ISDIR(fileInfo.st\_mode)) printf("/");

else if (fileInfo.st\_mode & S\_IXUSR) printf("\*");

printf("\n");

}

closedir(dr);

}

void addHistory(char\* input) {

HISTORY\* pNew = (HISTORY\*)malloc(sizeof(HISTORY));

pNew->hist = (char\*)malloc(COMMAND\_SIZE \* sizeof(char));

strcpy(pNew->hist, input);

pNew->link = NULL;

if (!hisHead) {

hisHead = pNew;

pNew->num = 1;

return ;

}

HISTORY\* pMove;

for (pMove = hisHead; pMove->link; pMove = pMove->link) ;

pMove->link = pNew;

pNew->num = pMove->num + 1;

}

void history() {

HISTORY\* pMove;

for (pMove = hisHead; pMove; pMove = pMove->link) {

printf("\t%d\t%s\n", pMove->num, pMove->hist);

}

}

void quit() {

// free memories

HISTORY\* pFree;

while (hisHead) {

pFree = hisHead;

hisHead = hisHead->link;

free(pFree);

}

opNode\* opFree, \*optmp;

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

optmp = opTable[i];

while (optmp) {

opFree = optmp;

optmp = optmp->link;

free(opFree);

}

}

free(MEMORY);

printf("exit shell\n");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* memory.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 190325 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "20171690.h"

int dump(char\* start, char\* end, int type) {

// du[mp] [start, end]

int s, e;

int tmp = END\_ADDR;

switch (type) {

case 0: // du[mp]

if (END\_ADDR > 0xFFFFF) END\_ADDR = 0;

s = END\_ADDR;

e = ((END\_ADDR + 0x9F) > 0xFFFFF) ? 0xFFFFF : END\_ADDR + 0x9F;

END\_ADDR = e + 0x1;

break;

case 1: // du[mp] start

s = strToHex(start);

e = ((s + 0x9F) > 0xFFFFF) ? 0xFFFFF : s + 0x9F;

END\_ADDR = e + 0x1;

break;

case 2: // du[mp] start, end

s = strToHex(start);

e = strToHex(end);

END\_ADDR = e + 0x1;

break;

}

// incorrect input or addr and val are not hex

if (s == -1 || e == -1) {

printf("Incorrect input. Please text again\n");

END\_ADDR = tmp;

return 0;

}

if (!validAddrRange(s, e)) {

END\_ADDR = tmp;

return 0;

}

int i, j;

int startLineAddr = s / 16 \* 16;

int endLineAddr = e / 16 \* 16;

int line = (endLineAddr - startLineAddr) / 16 + 1;

int addr;

char ASCIIcode[17];

ASCIIcode[16] = '\0';

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

printf("%05X ", startLineAddr + i \* 16);

for (j = 0; j < 16; j++) {

addr = startLineAddr + i \* 16 + j;

if (addr < s) {

printf(" ");

ASCIIcode[j] = '.';

}

else if (addr > e) {

printf(" ");

ASCIIcode[j] = '.';

}

else {

printf("%02X ", \*(MEMORY + addr));

if (\*(MEMORY + addr) >= 0x20 && \*(MEMORY + addr) <= 0x7E) ASCIIcode[j] = \*(MEMORY + addr);

else ASCIIcode[j] = '.';

}

}

printf("; %s\n", ASCIIcode);

}

return 1;

}

int edit(char\* address, char\* value) {

// e[dit] address, value

int addr = strToHex(address);

int val = strToHex(value);

// incorrect input or addr and val are not hex

if (addr == -1 || val == -1) {

printf("Incorrect input. Please text again\n");

return 0;

}

if (val > 0xFF) {

printf("Invalid value. Too big!\n");

return 0;

}

// then edit memory

unsigned char realValue = val & ONE\_BYTE;

memcpy(MEMORY + addr, &realValue, 1);

return 1;

}

int fill(char\* start, char\* end, char\* value) {

// f[ill] start, end, value

int s, e, v;

s = strToHex(start);

e = strToHex(end);

v = strToHex(value);

// incorrect input or addr and val are not hex

if (s == -1 || e == -1 || v == -1) {

printf("Incorrect input. Please text again\n");

return 0;

}

if (!validAddrRange(s, e)) {

return 0;

}

if (v > 0xFF) {

printf("Invalid value. Too big!\n");

return 0;

}

unsigned char realValue = v & ONE\_BYTE;

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

memcpy(MEMORY + i, &realValue, 1);

return 1;

}

void reset() {

// reset

for (int i = 0; i < MEMORY\_SIZE; i++)

MEMORY[i] = 0x00;

}

int validAddr(int addr) {

// if given address is vaild, return 1, else return 0

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

return 1;

else

return 0;

}

int validAddrRange(int start, int end) {

if (!validAddr(start)) {

printf("Segmentation fault: Cannot access %05X\n", start);

return 0;

}

else if (!validAddr(end)) {

printf("Segmentation fault: Cannot access %05X\n", end);

return 0;

}

else if (end < start) {

printf("Incorrect Address Range. End address must be bigger than Start address.\n");

return 0;

}

return 1;

}

int strToHex(char\* param) {

// if parameter is not a hex, or incorrect parser return -1

for (int i = 0; i < (int)strlen(param); i++) {

if ( !isHex(param[i]) ) {

printf("Address is not a Hexadecimal!\n");

return -1;

}

}

// else, return proper hexa decimal number

int hex;

int res = sscanf(param, "%x", &hex);

if (res == 0) {

printf("error occured. Please type again\n");

return -1;

}

return hex;

}

int isHex(char ch) {

if ( (ch >= 'A' && ch <= 'F') || (ch >= 'a' && ch <= 'f') || (ch >= '0' && ch <= '9') )

return 1;

else

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\* System Programming \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* opcode.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Yuseok \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 190325 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "20171690.h"

int opcode(char\* mnemonic) {

// if found correct mnemonic, return 1, else return 0

int idx = hashFunction(mnemonic);

opNode\* pMove;

for (pMove = opTable[idx]; pMove; pMove = pMove->link) {

if (strcmp(pMove->operation, mnemonic) == 0) {

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

return 1;

}

}

printf("Wrong mnemonic\n");

return 0;

}

int opcodeList() {

if (!opTable) return 0;

opNode\* pMove;

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

printf("%d : ", i);

pMove = opTable[i];

if (pMove) {

for ( ; pMove->link; pMove = pMove->link)

printf("[%s,%X] -> ", pMove->operation, pMove->opcode);

printf("[%s,%X]", pMove->operation, pMove->opcode);

}

printf("\n");

}

return 1;

}

int readOpTable() {

FILE\* fp = fopen(OPCODE, "r");

if (!fp) {

printf("wrong input file\n");

return 0;

}

int idx;

int opcode;

char operation[7];

char formatStr[4];

while (fscanf(fp, "%X %s %s\n", &opcode, operation, formatStr) != EOF) {

opNode\* pNew = (opNode\*)malloc(sizeof(opNode));

pNew->opcode = opcode;

strcpy(pNew->operation, operation);

if (strcmp(formatStr, "3/4") == 0) pNew->format = 3;

else if (strcmp(formatStr, "1") == 0) pNew->format = 1;

else if (strcmp(formatStr, "2") == 0) pNew->format = 2;

else pNew->format = 0;

pNew->link = NULL;

idx = hashFunction(operation);

if (!opTable[idx])

opTable[idx] = pNew;

else {

opNode\* pMove;

for (pMove = opTable[idx]; pMove->link; pMove = pMove->link) ;

pMove->link = pNew;

}

}

fclose(fp);

return 1;

}

int hashFunction(char\* op) {

int idx = 0;

for (int i = 0; i < (int)strlen(op); i++)

idx += op[i];

return idx % 20;

}

# makefile

CC = gcc

CFLAGS = -Wall -W -g

OBJECTS = 20171690.o shell.o memory.o opcode.o

TARGET = 20171690.out

all: $(TARGET)

$(TARGET) : $(OBJECTS)

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

clean:

rm -rf $(OBJECTS) $(TARGET) $(TARGET).dSYM