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 : help()
            1. 기능
            2. 사용 변수
         2. 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. tRecord\* tRTail
6. 코드

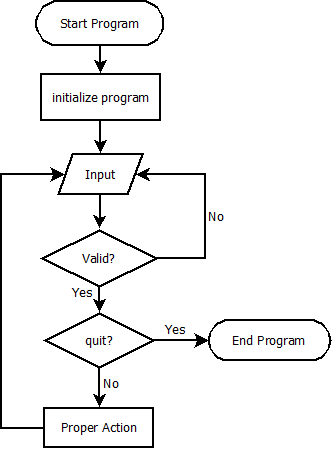
1. 프로그램 개요

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. help 명령어에서 새로운 명령어 목록을 업데이트하였고, type filename 기능을 추가하였다. 또한 이번 프로젝트의 목적인 어셈블러 기능을 구현한 assemble.c, 어셈블러 기능을 추가하기 위해 작성한 함수들을 모아 놓은 assembler\_functions.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. 이번 프로젝트에서 새로 구현한 명령어 (type, assemble, symbol) 목록을 추가하였다.

3.2.2.1.2 사용 변수

없음

3.2.2.2 모듈 이름 : **type(char\* filename)**

3.2.2.2.1 기능

parameter로 전달받은 filename에 해당하는 파일을 현재 디렉터리에서 읽어 화면에 출력한다. 이 때 파일이 없으면 에러 메시지를 출력한다.

3.2.2.2.2 사용 변수

**char\* filename** – 읽을 파일의 이름이다.

**FILE\* fp** – 읽어올 파일 포인터를 가리킨다.

3.3 memory.c

3.4 opcode.c

**3.5 assembler.c**

3.5.1 설명

3.5.2 모듈 정의

3.5.2.1 모듈 이름 : **assemble(char\* filename)**

3.5.2.1.1 기능

3.5.2.1.2 사용 변수

3.5.2.2 모듈 이름 : **pass1(FILE\* fp)**

3.5.2.2.1 기능

3.5.2.2.2 사용 변수

3.5.2.3 모듈 이름 : **pass2(FILE\* fp, char\* filename)**

3.5.2.3.1 기능

3.5.2.3.2 사용 변수

3.5.2.5 모듈 이름 : **getInstructionSize(char\*\* token, int lineNum, int isLabel)**

3.5.2.5.1 기능

3.5.2.5.2 사용 변수

3.5.2.6 모듈 이름 : **getObjCode(char\*\* token, int\* format, int type, numNode\* pCurrent)**

3.5.2.6.1 기능

3.5.2.6.2 사용 변수

3.5.2.7 모듈 이름 : **tokenizeAsmFile(char\*\*\* token, char\* input)**

3.5.2.7.1 기능

3.5.2.7.2 사용 변수

3.5.2.8 모듈 이름 : **removeSpaceAroundComma(char\* input)**

3.5.2.8.1 기능

3.5.2.8.2 사용 변수

**3.6 assembler\_functions.c**

3.6.1 설명

3.6.2 모듈 정의

3.6.2.1 모듈 이름 : **getRegNum(char\* reg)**

3.6.2.1.1 기능

3.6.2.1.2 사용 변수

3.6.2.2 모듈 이름 : **LDB(char\*\* token, int lineNum)**

3.6.2.2.1 기능

3.6.2.2.2 사용 변수

3.6.2.3 모듈 이름 : **printLineinLST(numNode\* pCurrent, char\*\* token, int size, int tokenNum, int objCode, FILE\* LF, int isLabel)**

3.6.2.3.1 기능

3.6.2.3.2 사용 변수

3.6.2.4 모듈 이름 : **isWhiteSpace(char ch)**

3.6.2.4.1 기능

3.6.2.4.2 사용 변수

3.6.2.5 모듈 이름 : **isBlankLine(char\* input)**

3.6.2.5.1 기능

3.6.2.5.2 사용 변수

3.6.2.6 모듈 이름 : **isDirective(char\*token)**

3.6.2.6.1 기능

3.6.2.6.2 사용 변수

3.6.2.7 모듈 이름 : **isComma(char\* input)**

3.6.2.7.1 기능

3.6.2.7.2 사용 변수

3.6.2.8 모듈 이름 : **nameToListing(char\* filename)**

3.6.2.8.1 기능

3.6.2.8.2 사용 변수

3.6.2.9 모듈 이름 : **nameToObj(char\* filename)**

3.6.2.9.1 기능

3.6.2.9.2 사용 변수

3.6.2.10 모듈 이름 : **toUpper(char ch)**

3.6.2.10.1 기능

3.6.2.10.2 사용 변수

3.6.2.11 모듈 이름 : **toUpperCase(char\* input)**

3.6.2.11.1 기능

3.6.2.11.2 사용 변수

3.6.2.12 모듈 이름 : **byteSize(char\* input)**

3.6.2.12.1 기능

3.6.2.12.2 사용 변수

3.6.2.13 모듈 이름 : **wordSize(char\* input)**

3.6.2.13.1 기능

3.6.2.13.2 사용 변수

3.6.2.14 모듈 이름 : **resbSize(char\* input)**

3.6.2.14.1 기능

3.6.2.14.2 사용 변수

3.6.2.15 모듈 이름 : **reswSize(char\* input)**

3.6.2.15.1 기능

3.6.2.15.2 사용 변수

3.6.2.16 모듈 이름 : **addNum(int lineNum, int LOC, numNode\* pLast, int flagType)**

3.6.2.16.1 기능

3.6.2.16.2 사용 변수

3.6.2.17 모듈 이름 : **freeNums()**

3.6.2.17.1 기능

3.6.2.17.2 사용 변수

3.6.2.18 모듈 이름 : **printNums()**

3.6.2.18.1 기능

3.6.2.18.2 사용 변수

3.6.2.19 모듈 이름 : **enqueue(int objCode, int size, int LOC, FILE\* OF)**

3.6.2.19.1 기능

3.6.2.19.2 사용 변수

3.6.2.20 모듈 이름 : **dequeue(FILE\* OF)**

3.6.2.21.1 기능

3.6.2.21.2 사용 변수

3.6.2.22 모듈 이름 : **freeQueue()**

3.6.2.22.1 기능

3.6.2.22.2 사용 변수

**3.7 symbol.c**

3.7.1 설명

3.7.2 모듈 정의

3.7.2.1 모듈 이름 : symbol()

3.7.2.1.1 기능

3.7.2.1.2 사용 변수

3.7.2.2 모듈 이름 : addSym(char\* label, int LOC)

3.7.2.2.1 기능

3.7.2.2.2 사용 변수

3.7.2.3 모듈 이름 : freeSymTab()

3.7.2.3.1 기능

3.7.2.3.2 사용 변수

3.7.2.4 모듈 이름 : symHashFunc(char\* label)

3.7.2.4.1 기능

3.7.2.4.2 사용 변수

3.7.2.5 모듈 이름 : findSym(char\* label)

3.7.2.5.1 기능

3.7.2.5.2 사용 변수

3.7.2.6 모듈 이름 : isStr(char\* Str)

3.7.2.6.1 기능

3.7.2.6.2 사용 변수

3.7.2.7 모듈 이름 : getBiggetStr(char\* str1, char\* str2)

3.7.2.7.1 기능

3.7.2.7.2 사용 변수

3.7.2.8 모듈 이름 : getMaxofFour(char\* str1, char\* str2, char\* str3, char\* str4)

3.7.2.8.1 기능

3.7.2.8.2 사용 변수

3.7.2.9 모듈 이름 : getMaxofThree(char\* str1, char\* str2, chat\* str3)

3.7.2.9.1 기능

3.7.2.9.2 사용 변수

3.7.2.10 모듈 이름 : printSymbol()

3.7.2.10.1 기능

3.7.2.10.2 사용 변수

4. 구조체 정의

4.1 HISTORY

4.2 opNode

**4.3 symNode**

**4.4 numNode**

**4.5 tRecord**

5. 전역 변수 정의

5.1 HISTORY\* hisHead

5.2 opNode\*\* opTable

5.3 unsigned char\* MEMORY

5.4 END\_ADDR

**5.5 symNode\*\* SYMTAB**

**5.6 numNode\* numHead**

**5.7 tRecord\* tRHead**

**5.8 tRecord\* tRTail**

**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