시스템 프로그래밍 프로젝트

20142385 엄재식

출석번호 19

1. 목적

- ControlSection 방식으로 생성된 ObjectCode(프로젝트#2의 결과물)를 실행하고 시뮬레이션할 수 있는 시뮬레이터 만들기

- 시뮬레이션 과정이 Step-by-Step으로 Visual하게 보여주는 Java GUI 프로그램

- GUI를 위한 모듈, 연산 모듈, 가상 장치(메모리, 레지스터) 모듈, 로더를 통하여 시뮬레이터 구현

1. 설계/구현 아이디어
2. ResourceManager.class

Device클래스를 따로 만들어서 해쉬배열에 넣음 key는 디바이스의 정수값

device클래스는 저장공간 char 배열과 읽을 index 정수 값을 멤버로 갖고 버퍼를 읽는 메소드와 쓰는 메소드를 추가함

나머지는 제공한 모듈의 함수들을 목적에 맞게 채워 놓음

1. TokenTable.class

어셈블러를 파싱하여 토큰으로 만듬 토큰은 extdef, extref, modification, instruction별로 테이블을 따로 생성해 구분한다.

1. SicLoader.class

어셈블러를 코드의 첫 문자 별로 방법을 달리하여 코드를 파싱한다. 파싱한 코드는 token으로 생성되어 목적에 맞게 생성된 테이블에 넣어진다.

첫 문자가 H인 코드로 프로그램의 주소를 찾는다.

첫 문자가 D일 경우 토큰을 만들어 extdef 테이블에 넣는다

첫 문자가 R일 경우 토큰을 만들어 extref 테이블에 넣는다

첫 문자가 T일 경우 시작주소와 길이를 뺀 나머지 부분을 메모리에 로드 시켜준다. instruction마다 토큰을 만들어 inst 테이블에 넣는다.

1. SicSimulator.class

instCnt 변수를 만들어 SicLoader에서 만든 inst 테이블 인덱스로 사용한다.

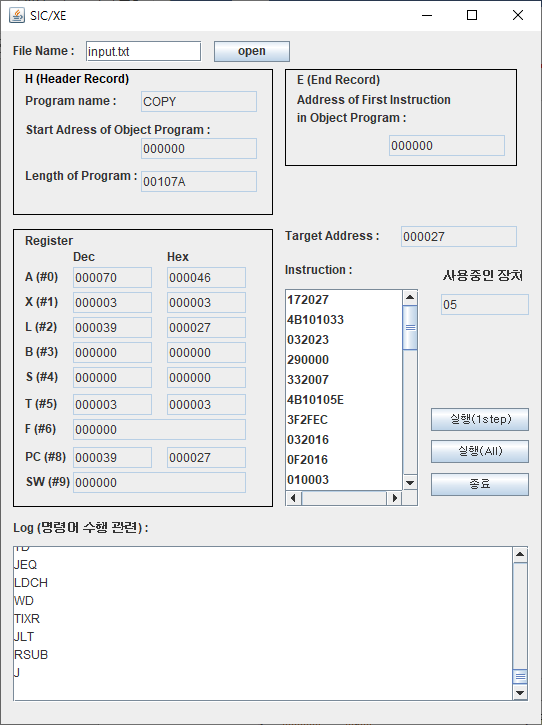
Onestep 메소드에서 inst 테이블의 instCnt번째 instruction을 수행하는 동작을 한다. Appendix에 맞게 Instruct의 이름별로 동작을 구현함. 구현상 어려운 instruction은 inst.data에서 삭제함

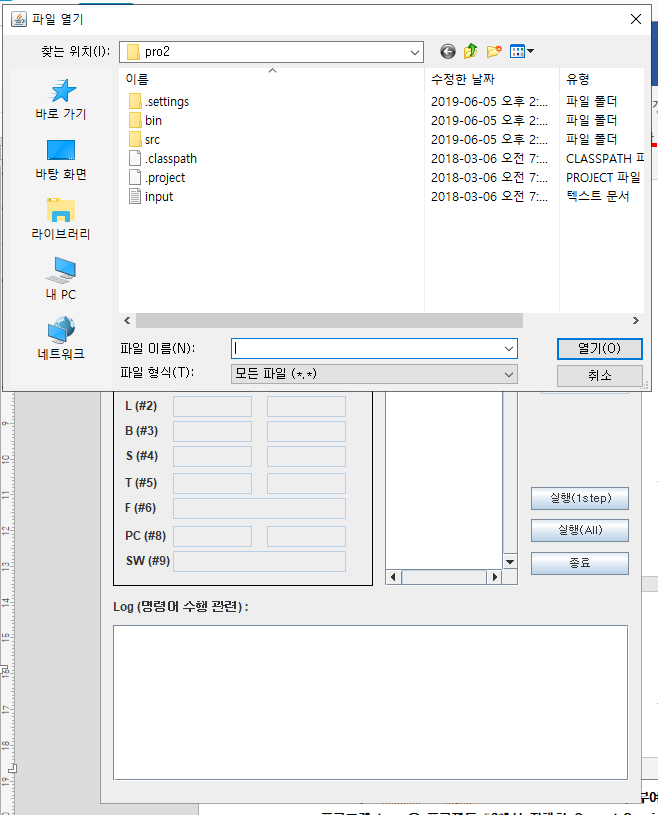
Allstep은 onestep을 반복하면 되기 때문에 따로 구현 안함

1. VisualSimulator.class

Gui를 구현하고 적절한 버튼 이벤트를 등록함 맨밑의 텍스트에리어는 실행한 instruction의 이름을 출력시킴

1. 수행결과





1. 결론 및 보충할 점

프로젝트 1을 이용해 만든 코드를 메모리에 적재하고 시뮬레이션 시켜보는 프로그램을 자바 GUI 환경을 통해 구현했다. 코드를 파싱할 때 instruction인지 데이터인지 구분하는 것이 굉장히 힘들었다. 이 과제를 통해 익숙하지 않은 자바 gui를 더 알게 되었고 문자열을 더 잘 다룰 수 있게 되었다. 프로그램이 복잡하여 제대로 동작하는지는 잘 모르겠지만 배운 것이 많은 과제이다.

1. 소스코드

InstTable.java

package SP18\_simulator;

import java.util.HashMap;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

/\*\*

\* 모든 instruction의 정보를 관리하는 클래스. instruction data들을 저장한다. <br>

\* 또한 instruction 관련 연산, 예를 들면 목록을 구축하는 함수, 관련 정보를 제공하는 함수 등을 제공 한다.

\*/

public class InstTable {

/\*\*

\* inst.data 파일을 불러와 저장하는 공간.

\* 명령어의 이름을 집어넣으면 해당하는 Instruction의 정보들을 리턴할 수 있다.

\*/

private HashMap<Integer, Instruction> instMap;

/\*\*

\* 클래스 초기화. 파싱을 동시에 처리한다.

\* @param instFile : instuction에 대한 명세가 저장된 파일 이름

\*/

public InstTable(String instFile) {

instMap = new HashMap<Integer, Instruction>();

openFile(instFile);

}

/\*\*

\* 입력받은 이름의 파일을 열고 해당 내용을 파싱하여 instMap에 저장한다.

\*/

public void openFile(String fileName) {

//...

Instruction newInstruction;

String str;

try {

File file = new File(fileName);

FileReader filereader = new FileReader(file);

BufferedReader bufReader = new BufferedReader(filereader);

while((str = bufReader.readLine()) != null){

newInstruction = new Instruction(str);

instMap.put(newInstruction.getOpcode(), newInstruction);

}

bufReader.close();

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public Instruction getInstruction(int op) {

return instMap.get(op);

}

//get, set, search 등의 함수는 자유 구현

}

/\*\*

\* 명령어 하나하나의 구체적인 정보는 Instruction클래스에 담긴다.

\* instruction과 관련된 정보들을 저장하고 기초적인 연산을 수행한다.

\*/

class Instruction {

/\*

\* 각자의 inst.data 파일에 맞게 저장하는 변수를 선언한다.

\*

\* ex)

\* String instruction;

\* int opcode;

\* int numberOfOperand;

\* String comment;

\*/

/\*\* instruction이 몇 바이트 명령어인지 저장. 이후 편의성을 위함 \*/

private int format;

private int opcode;

private String instruction;

private int numberOfOperand;

/\*\*

\* 클래스를 선언하면서 일반문자열을 즉시 구조에 맞게 파싱한다.

\* @param line : instruction 명세파일로부터 한줄씩 가져온 문자열

\*/

public Instruction(String line) {

parsing(line);

}

/\*\*

\* 일반 문자열을 파싱하여 instruction 정보를 파악하고 저장한다.

\* @param line : instruction 명세파일로부터 한줄씩 가져온 문자열

\*/

public void parsing(String line) {

// TODO Auto-generated method stub

String[] array = line.split(" ");

if (array.length != 4) {

System.err.println("inst error " + array.length);

System.exit(-1);

}

instruction = array[0];

format = Integer.valueOf(array[1]);

opcode = Integer.valueOf(array[2], 16);

numberOfOperand = Integer.valueOf(array[3]);

}

public int getFormat() {

return format;

}

public int getOpcode() {

return opcode;

}

public int getNumberOfOperand() {

return numberOfOperand;

}

public String getInstruction() {

return instruction;

}

//그 외 함수 자유 구현

}

TokenTable.java

import java.util.ArrayList;

public class TokenTable {

ArrayList<Token> tokenList;

TokenTable() {

tokenList = new ArrayList<Token>();;

}

public void putToken(Token token) {

tokenList.add(token);

}

public Token getToken(int index) {

return tokenList.get(index);

}

public void modifyToken(Token token, int location) {

tokenList.set(location, token);

}

public int length() {

return tokenList.size();

}

}

class Token {

public String name;

public int addr;

public int sub;

public int mod;

public int isOp; //+ = 1, - = 2

public int op;

public int format;

public int nixbpe;

public int target;

final int mask3 = 4095, mask4 = 65535;

Token(String str)

{

this.name=str;

}

void modify(int addr) {

String str;

int modInst = Integer.valueOf(name, 16);

if (format == 3) {

addr = addr & mask3;

modInst += addr;

str = String.format("%06X", modInst);

}

else {

addr = addr & mask4;

modInst += addr;

str = String.format("%08X", modInst);

}

this.target += addr;

name = str;

}

}

ResourceManager.java

import java.io.File;

import java.util.ArrayList;

import java.util.HashMap;

/\*\*

\* ResourceManager는 컴퓨터의 가상 리소스들을 선언하고 관리하는 클래스이다.

\* 크게 네가지의 가상 자원 공간을 선언하고, 이를 관리할 수 있는 함수들을 제공한다.<br><br>

\*

\* 1) 입출력을 위한 외부 장치 또는 device<br>

\* 2) 프로그램 로드 및 실행을 위한 메모리 공간. 여기서는 64KB를 최대값으로 잡는다.<br>

\* 3) 연산을 수행하는데 사용하는 레지스터 공간.<br>

\* 4) SYMTAB 등 simulator의 실행 과정에서 사용되는 데이터들을 위한 변수들.

\* <br><br>

\* 2번은 simulator위에서 실행되는 프로그램을 위한 메모리공간인 반면,

\* 4번은 simulator의 실행을 위한 메모리 공간이라는 점에서 차이가 있다.

\*/

public class ResourceManager{

/\*\*

\* 디바이스는 원래 입출력 장치들을 의미 하지만 여기서는 파일로 디바이스를 대체한다.<br>

\* 즉, 'F1'이라는 디바이스는 'F1'이라는 이름의 파일을 의미한다. <br>

\* deviceManager는 디바이스의 이름을 입력받았을 때 해당 이름의 파일 입출력 관리 클래스를 리턴하는 역할을 한다.

\* 예를 들어, 'A1'이라는 디바이스에서 파일을 read모드로 열었을 경우, hashMap에 <"A1", scanner(A1)> 등을 넣음으로서 이를 관리할 수 있다.

\* <br><br>

\* 변형된 형태로 사용하는 것 역시 허용한다.<br>

\* 예를 들면 key값으로 String대신 Integer를 사용할 수 있다.

\* 파일 입출력을 위해 사용하는 stream 역시 자유로이 선택, 구현한다.

\* <br><br>

\* 이것도 복잡하면 알아서 구현해서 사용해도 괜찮습니다.

\*/

HashMap<Integer,Object> deviceManager = new HashMap<Integer,Object>();

char[] memory = new char[65536]; // String으로 수정해서 사용하여도 무방함.

int[] register = new int[10];

Integer curdev = null;

SymbolTable symtabList;

public static final int aReg=0;

public static final int xReg=1;

public static final int lReg=2;

public static final int bReg=3;

public static final int sReg=4;

public static final int tReg=5;

public static final int fReg=6;

public static final int pcReg=7;

public static final int swReg=8;

// 이외에도 필요한 변수 선언해서 사용할 것.

/\*\*

\* 메모리, 레지스터등 가상 리소스들을 초기화한다.

\*/

public void initializeResource(){

for(int i = 0; i < memory.length; i++) {

memory[i] = 0;

}

for (int i = 0; i < register.length; i++) {

register[i] = 0;

}

}

/\*\*

\* deviceManager가 관리하고 있는 파일 입출력 stream들을 전부 종료시키는 역할.

\* 프로그램을 종료하거나 연결을 끊을 때 호출한다.

\*/

public void closeDevice() {

deviceManager.clear();

}

/\*\*

\* 디바이스를 사용할 수 있는 상황인지 체크. TD명령어를 사용했을 때 호출되는 함수.

\* 입출력 stream을 열고 deviceManager를 통해 관리시킨다.

\* @param devName 확인하고자 하는 디바이스의 번호,또는 이름

\*/

public void testDevice(Integer devName) {

if (deviceManager.get(devName) == null) {

deviceManager.put(devName, new Device());

register[swReg] = 0;

}

else {

register[swReg] = 1;

curdev = devName;

}

}

/\*\*

\* 디바이스로부터 원하는 개수만큼의 글자를 읽어들인다. RD명령어를 사용했을 때 호출되는 함수.

\* @param devName 디바이스의 이름

\* @param num 가져오는 글자의 개수

\* @return 가져온 데이터

\*/

public char[] readDevice(Integer devName, int num){

Device device = (Device) deviceManager.get(devName);

return device.read(num);

}

/\*\*

\* 디바이스로 원하는 개수 만큼의 글자를 출력한다. WD명령어를 사용했을 때 호출되는 함수.

\* @param devName 디바이스의 이름

\* @param data 보내는 데이터

\* @param num 보내는 글자의 개수

\*/

public void writeDevice(Integer devName, char[] data, int num){

Device device = (Device) deviceManager.get(devName);

device.write(data, num);

}

/\*\*

\* 메모리의 특정 위치에서 원하는 개수만큼의 글자를 가져온다.

\* @param location 메모리 접근 위치 인덱스

\* @param num 데이터 개수

\* @return 가져오는 데이터

\*/

public char[] getMemory(int location, int num){

char[] buf = new char[num];

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

buf[i] = memory[location + i];

System.out.println("getMemory at " + String.format("%06X", location+i));

}

System.out.println(String.format("%06X", this.charToInt(buf)));

return buf;

}

/\*\*

\* 메모리의 특정 위치에 원하는 개수만큼의 데이터를 저장한다.

\* @param locate 접근 위치 인덱스

\* @param data 저장하려는 데이터

\* @param num 저장하는 데이터의 개수

\*/

public void setMemory(int locate, char[] data, int num){

for(int i = 0; i < num && i < data.length; i++) {

memory[locate + (num - data.length) + i] = data[i];

System.out.print("setMemory " +String.format("%06X", locate + (num - data.length) + i) + " " + " " + String.format("%02X", (int)memory[locate + (num - data.length) + i]) + " ");

}

System.out.println();

}

/\*\*

\* 번호에 해당하는 레지스터가 현재 들고 있는 값을 리턴한다. 레지스터가 들고 있는 값은 문자열이 아님에 주의한다.

\* @param regNum 레지스터 분류번호

\* @return 레지스터가 소지한 값

\*/

public int getRegister(int regNum){

return register[regNum];

}

/\*\*

\* 번호에 해당하는 레지스터에 새로운 값을 입력한다. 레지스터가 들고 있는 값은 문자열이 아님에 주의한다.

\* @param regNum 레지스터의 분류번호

\* @param value 레지스터에 집어넣는 값

\*/

public void setRegister(int regNum, int value){

register[regNum] = value;

}

/\*\*

\* 주로 레지스터와 메모리간의 데이터 교환에서 사용된다. int값을 char[]형태로 변경한다.

\* @param data

\* @return

\*/

public char[] intToChar(int data){

char buf[];

int i = 0, tmp = data;

for (; tmp / 256 > 0; tmp /= 256, i++) {}

i++;

buf = new char[i];

for (int j = 0; j < i; j++, data /= 256) {

buf[i-j-1] = (char) (data & 255);

}

return buf;

}

/\*\*

\* 주로 레지스터와 메모리간의 데이터 교환에서 사용된다. char[]값을 int형태로 변경한다.

\* @param data

\* @return

\*/

public int charToInt(char[] data){

int n = 0;

for (int i = 0; i < data.length; i++) {

n += data[i] << 8 \* (data.length - (i+1));

}

return n;

}

}

class Device {

char[] buf;

int index;

public Device() {

buf = new char[100];

for (int i = 0; i < 3; i++)

buf[i] = (char)i;

index = 0;

}

char[] read(int n) {

char[] buffer = new char[n];

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

buffer[i] = buf[index++];

}

return buffer;

}

void write(char[] buffer, int n) {

for (int i = 0; i < n && i < buffer.length; i++) {

buf[index++] = buffer[i];

}

}

}

SicLoader.java

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

/\*\*

\* SicLoader는 프로그램을 해석해서 메모리에 올리는 역할을 수행한다. 이 과정에서 linker의 역할 또한 수행한다.

\* <br><br>

\* SicLoader가 수행하는 일을 예를 들면 다음과 같다.<br>

\* - program code를 메모리에 적재시키기<br>

\* - 주어진 공간만큼 메모리에 빈 공간 할당하기<br>

\* - 과정에서 발생하는 symbol, 프로그램 시작주소, control section 등 실행을 위한 정보 생성 및 관리

\*/

public class SicLoader {

ResourceManager rMgr;

InstTable instTable;

BufferedReader read = null;

TokenTable extref = new TokenTable();

TokenTable extdef = new TokenTable();

TokenTable mod = new TokenTable();

TokenTable inst = new TokenTable();

int sub = 0, programLength = 0;

static String programName, programStart; //mod레코드에서 상대주소를 저장하는 리스트

static ArrayList<String> start = new ArrayList<String>(); //루틴의 시작주소

static ArrayList<String> rStart = new ArrayList<String>(); //루틴의 실제시작주소

static ArrayList<String> name = new ArrayList<String>(); //루틴의 이름

static ArrayList<String> length = new ArrayList<String>(); //루틴의 길이

static int loc=0; //loc값

public SicLoader(ResourceManager resourceManager) {

// 필요하다면 초기화

setResourceManager(resourceManager);

instTable = new InstTable("inst.data");

}

/\*\*

\* Loader와 프로그램을 적재할 메모리를 연결시킨다.

\* @param rMgr

\*/

public void setResourceManager(ResourceManager resourceManager) {

this.rMgr=resourceManager;

}

/\*\*

\* object code를 읽어서 load과정을 수행한다. load한 데이터는 resourceManager가 관리하는 메모리에 올라가도록 한다.

\* load과정에서 만들어진 symbol table 등 자료구조 역시 resourceManager에 전달한다.

\* @param objectCode 읽어들인 파일

\*/

public void load(File objectCode) {

String line;

int cnt = 0;

int op, nixbpe, rstart=0;

Instruction instBuf;

final int opMask = 252 , nixbpeMask = 63, niMask = 48, bpMask = 6;

char[] buf = null;

try {

read = new BufferedReader(new FileReader(objectCode));

while ((line = read.readLine()) != null) {

if (line.charAt(0) == 'H') {

sub++;

name.add(line.substring(1, 7));

length.add(line.substring(14, 19));

if (sub == 1) {

programName = line.substring(1, 7);

programStart = line.substring(7, 13);

rstart = Integer.valueOf(line.substring(7, 13), 16);

}

if (sub > 1) {

int n = 0, m = 0;

for (int i = 0; i < sub - 1; i++) {

n += Integer.parseInt(length.get(i), 16);

}

m = Integer.parseInt(line.substring(7, 13), 16);

String s = String.format("%06X", n + m);

System.out.println("start "+s);

rStart.add(s);

loc = n + m;

rstart = loc;

cnt = 0;

cnt += Integer.parseInt(s, 16);

Token tokenbuf =new Token(line.substring(1, 7));

tokenbuf.addr = loc;

tokenbuf.sub = sub;

extdef.putToken(tokenbuf);

}

}

else if (line.charAt(0) == 'D') {

Token tokenbuf = new Token(line.substring(1, 7));

tokenbuf.sub = sub;

if (sub > 1) {

tokenbuf.addr = Integer.parseInt(line.substring(7, 13), 16)

+ Integer.parseInt(rStart.get(sub - 2), 16);

}

else {

tokenbuf.addr = Integer.parseInt(line.substring(7, 13), 16);

}

extdef.putToken(tokenbuf);

if (line.length() > 14) {

tokenbuf = new Token(line.substring(13, 19));

tokenbuf.sub = sub;

if (sub > 1) {

tokenbuf.addr = Integer.parseInt(line.substring(19, 25), 16)

+ Integer.parseInt(rStart.get(sub - 2), 16);

}

else {

tokenbuf.addr = Integer.parseInt(line.substring(19, 25), 16);

}

extdef.putToken(tokenbuf);

}

if (line.length() > 26) {

tokenbuf = new Token(line.substring(25, 31));

tokenbuf.sub = sub;

if (sub > 1) {

tokenbuf.addr = Integer.parseInt(line.substring(31, line.length()), 16)

+Integer.parseInt(rStart.get(sub - 2), 16);

}

else {

tokenbuf.addr = Integer.parseInt(line.substring(31, line.length()), 16);

}

extdef.putToken(tokenbuf);

}

}

else if (line.charAt(0) == 'R')

{

Token tokenbuf = new Token(line.substring(1, 7));

tokenbuf.sub = sub;

extref.putToken(tokenbuf);

if (line.length() > 8) {

tokenbuf = new Token(line.substring(7,13));

tokenbuf.sub = sub;

extref.putToken(tokenbuf);

}

if(line.length()>14) {

tokenbuf = new Token(line.substring(13, line.length()));

tokenbuf.sub = sub;

extref.putToken(tokenbuf);

}

}

else if (line.charAt(0) == 'T') {

int tCnt, tLen = Integer.parseInt(line.substring(7, 9), 16);

tCnt = 9;

System.out.println(line.substring(9,line.length()));

for (int i = 11; i < line.length()+1; i+=2) {

buf = rMgr.intToChar(Integer.valueOf(line.substring(i-2,i),16));

rMgr.setMemory(rstart+Integer.valueOf(line.substring(1,7), 16) + (i - 11)/2, buf, 1);

}

while (true) {

if ((tCnt/2) > tLen + 3)

break;

Token tokenbuf;

if (line.length() < tCnt + 3) {

tCnt += 2;

break;

}

op = Integer.valueOf(line.substring(tCnt, tCnt + 2), 16) & opMask;

instBuf = instTable.getInstruction(op);

if (instBuf != null) {

System.out.println(instBuf.getInstruction());

int[] n = new int[3];

if (instBuf.getFormat() != 2) {

nixbpe = Integer.parseInt(line.substring(tCnt + 1, tCnt + 3), 16) & nixbpeMask;

}

else

nixbpe = 0;

//데이터

if((((nixbpe & niMask) == 0 && (((nixbpe & bpMask) == 0) && ((nixbpe & 1) == 0)))

&& (instBuf.getFormat()!=2)))

{

n[0]=Integer.parseInt(line.substring(tCnt,tCnt+2),16);

if (line.length() > (tCnt-9)\*2+5)

{

n[1] = Integer.parseInt(line.substring(tCnt + 2, tCnt + 4), 16);

}

if (line.length() > (tCnt-9)\*2 +9 )

{

n[2] = Integer.parseInt(line.substring(tCnt + 4, tCnt + 6), 16);

}

if(n[2]<=65 && n[1]<=65)

{

tCnt+=2;

}

else

{

cnt+=3;

tCnt+=6;

}

}

else {

if (instBuf.getFormat() == 2) {

tokenbuf = new Token(line.substring(tCnt, tCnt + 4));

tokenbuf.sub = sub;

tokenbuf.addr = loc;

tokenbuf.op = instBuf.getOpcode();

tokenbuf.format = 2;

inst.putToken(tokenbuf);

cnt += 2;

tCnt += 4;

loc += 2;

}

else if (((nixbpe & 1) == 0) && (instBuf.getFormat() == 3)) {

tokenbuf = new Token(line.substring(tCnt, tCnt + 6));

tokenbuf.sub = sub;

tokenbuf.addr = loc;

tokenbuf.op = instBuf.getOpcode();

tokenbuf.format = 3;

tokenbuf.nixbpe = nixbpe;

tokenbuf.target = Integer.valueOf(line.substring(tCnt+3, tCnt+6), 16);

inst.putToken(tokenbuf);

cnt += 3;

tCnt += 6;

loc += 3;

}

else if (nixbpe > 0) {

tokenbuf = new Token(line.substring(tCnt, tCnt + 8));

tokenbuf.sub = sub;

tokenbuf.addr = loc;

tokenbuf.op = instBuf.getOpcode();

tokenbuf.format = 4;

tokenbuf.nixbpe = nixbpe;

tokenbuf.target = Integer.valueOf(line.substring(tCnt+3, tCnt+8), 16);

inst.putToken(tokenbuf);

cnt +=4;

tCnt += 8;

loc += 4;

}

}

}

else {

System.out.println(buf.length);

cnt += 1;

tCnt += 2;

loc += 1;

}

}

}

else if (line.charAt(0)=='M') {

Token tokenbuf = new Token(line.substring(10, line.length()));;

if (sub > 1) {

tokenbuf.addr = Integer.parseInt(line.substring(1, 7), 16)

+Integer.parseInt(rStart.get(sub - 2), 16);

}

else {

tokenbuf.addr = Integer.parseInt(line.substring(1, 7), 16);

}

tokenbuf.mod = Integer.parseInt(line.substring(7, 9), 16);

if (line.substring(9, 10).equals("+")) {

tokenbuf.isOp = 1;

}

else

tokenbuf.isOp = 2;

mod.putToken(tokenbuf);

}

}

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

for (int i = 0; i < length.size(); i++) {

programLength += Integer.valueOf(length.get(i), 16);

}

Token modbuf=null, defbuf=null, instbuf=null;

for (int i = 0; i < mod.length(); i++) {

char[] c;

modbuf = mod.getToken(i);

String str = null;

defbuf = null;

instbuf = null;

for (int j = 0; j < extdef.length(); j++) {

defbuf = extdef.getToken(j);

if (defbuf.name.contains(modbuf.name)) {

break;

}

}

if (defbuf != null) {

for (int j = 0; j < inst.length(); j++) {

instbuf = inst.getToken(j);

if (instbuf.addr <= mod.getToken(i).addr

&& instbuf.addr + instbuf.format > mod.getToken(i).addr) {

instbuf.modify(defbuf.addr);

break;

}

if (j == inst.length() - 1)

instbuf = null;

}

}

if (instbuf != null) {

buf = rMgr.intToChar(Integer.valueOf(instbuf.name, 16));

rMgr.setMemory(mod.getToken(i).addr - 1, buf, 4);

}

else {

System.out.println(modbuf.name);

if (modbuf.mod == 5) {

c = rMgr.getMemory(modbuf.addr + 1, 3);

str = String.format("%06X", defbuf.addr);

buf = rMgr.intToChar(Integer.valueOf(str, 16) + rMgr.charToInt(c));

rMgr.setMemory(modbuf.addr + 1, buf, 3);

}

else {

c = rMgr.getMemory(modbuf.addr, 3);

str = String.format("%06X", defbuf.addr);

if (modbuf.isOp == 1)

buf = rMgr.intToChar(Integer.valueOf(str, 16) + rMgr.charToInt(c));

else

buf = rMgr.intToChar(-Integer.valueOf(str, 16) + rMgr.charToInt(c));

rMgr.setMemory(modbuf.addr, buf, 3);

}

System.out.println(String.format("%06X", rMgr.charToInt(rMgr.getMemory(modbuf.addr, 3))));

}

}

}

}

**sicSimulator.java**

**import** java.io.File;

/\*\*

\* 시뮬레이터로서의 작업을 담당한다. VisualSimulator에서 사용자의 요청을 받으면 이에 따라

\* ResourceManager에 접근하여 작업을 수행한다.

\*

\* 작성중의 유의사항 : <br>

\* 1) 새로운 클래스, 새로운 변수, 새로운 함수 선언은 얼마든지 허용됨. 단, 기존의 변수와 함수들을 삭제하거나 완전히 대체하는 것은 지양할 것.<br>

\* 2) 필요에 따라 예외처리, 인터페이스 또는 상속 사용 또한 허용됨.<br>

\* 3) 모든 void 타입의 리턴값은 유저의 필요에 따라 다른 리턴 타입으로 변경 가능.<br>

\* 4) 파일, 또는 콘솔창에 한글을 출력시키지 말 것. (채점상의 이유. 주석에 포함된 한글은 상관 없음)<br>

\*

\* <br><br>

\* + 제공하는 프로그램 구조의 개선방법을 제안하고 싶은 분들은 보고서의 결론 뒷부분에 첨부 바랍니다. 내용에 따라 가산점이 있을 수 있습니다.

\*/

**public** **class** SicSimulator {

ResourceManager rMgr;

**int** instCnt = 0;

**final** **int** nmask = 32, imask = 16, xmask = 8, bmask = 4, pmask = 2, emask = 1, chmask = 255;

**final** **int** minus = 2048;

**final** **int** sim = 0, ind = 1, imm = 2;

**public** SicSimulator(ResourceManager resourceManager) {

// 필요하다면 초기화 과정 추가

**this**.rMgr = resourceManager;

}

/\*\*

\* 레지스터, 메모리 초기화 등 프로그램 load와 관련된 작업 수행.

\* 단, object code의 메모리 적재 및 해석은 SicLoader에서 수행하도록 한다.

\*/

**public** **void** load(File program) {

/\* 메모리 초기화, 레지스터 초기화 등\*/

//rMgr.initializeResource();

}

/\*\*

\* 1개의 instruction이 수행된 모습을 보인다.

\*/

**public** String oneStep(SicLoader loader) {

Token inst;

**if** (instCnt < loader.inst.length()) {

inst = loader.inst.getToken(instCnt++);

}

**else** {

**return** "END";

}

String name = loader.instTable.getInstruction(inst.op).getInstruction();

**int** data = -1, x = 0, loc = 0;

**int** mode = -1;

**int** reg1, reg2;

**int** target;

**char**[] buf;

System.***out***.println(name);

**if** (instCnt < loader.inst.length()) {

rMgr.setRegister(rMgr.***pcReg***, loader.inst.getToken(instCnt).addr);

}

**if** (inst.format == 3 || inst.format == 4) {

**if** ((inst.nixbpe & nmask) == nmask && (inst.nixbpe & imask) == imask)

mode = sim;

**else** **if** ((inst.nixbpe & nmask) == nmask)

mode = ind;

**else** **if** ((inst.nixbpe & imask) == imask)

mode = imm;

**if** ((inst.nixbpe & xmask) == xmask)

x = rMgr.getRegister(rMgr.***xReg***);

**else**

x = 0;

**if** ((inst.nixbpe & pmask) == pmask)

loc = rMgr.getRegister(rMgr.***pcReg***);

**else** **if** ((inst.nixbpe & bmask) == bmask)

loc = rMgr.getRegister(rMgr.***bReg***);

**else**

loc = 0;

**if** ((inst.target & minus) == minus) {

inst.target = (-4096) | inst.target;

}

}

**if** (name.equals("ADD")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***aReg***, rMgr.getRegister(rMgr.***aReg***) + data);

}

**else** **if** (name.equals("ADDF")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 5));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 5));

}

rMgr.setRegister(rMgr.***fReg***, rMgr.getRegister(rMgr.***fReg***) + data);

}

**else** **if** (name.equals("ADDR")) {

reg1 = Integer.*valueOf*(inst.name.substring(2,3), 16);

reg2 = Integer.*valueOf*(inst.name.substring(3,4), 16);

data = rMgr.getRegister(reg1) + rMgr.getRegister(reg2);

rMgr.setRegister(reg2, data);

}

**else** **if** (name.equals("AND")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***aReg***, rMgr.getRegister(rMgr.***aReg***) & data);

}

**else** **if** (name.equals("CLEAR")) {

reg1 = Integer.*valueOf*(inst.name.substring(2,3), 16);

rMgr.setRegister(reg1, 0);

}

**else** **if** (name.equals("COMP")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 5));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 5));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***swReg***, rMgr.getRegister(rMgr.***aReg***) - data);

}

**else** **if** (name.equals("COMPF")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 5));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 5));

}

rMgr.setRegister(rMgr.***swReg***, rMgr.getRegister(rMgr.***fReg***) - data);

}

**else** **if** (name.equals("COMPR")) {

reg1 = Integer.*valueOf*(inst.name.substring(2,3), 16);

reg2 = Integer.*valueOf*(inst.name.substring(3,4), 16);

rMgr.setRegister(rMgr.***swReg***, rMgr.getRegister(reg1) - rMgr.getRegister(reg2));

}

**else** **if** (name.equals("DIV")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***aReg***, rMgr.getRegister(rMgr.***aReg***) / data);

}

**else** **if** (name.equals("ADDF")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 5));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 5));

}

rMgr.setRegister(rMgr.***fReg***, rMgr.getRegister(rMgr.***fReg***) / data);

}

**else** **if** (name.equals("ADDR")) {

reg1 = Integer.*valueOf*(inst.name.substring(2,3), 16);

reg2 = Integer.*valueOf*(inst.name.substring(3,4), 16);

data = rMgr.getRegister(reg2) / rMgr.getRegister(reg1);

rMgr.setRegister(reg2, data);

}

**else** **if** (name.equals("J")) {

**if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else**

data = inst.target + x + loc;

rMgr.setRegister(rMgr.***pcReg***, data);

**for** (**int** i = 0; i < loader.inst.length(); i++) {

**if** (loader.inst.getToken(i).addr == data) {

instCnt = i;

**break**;

}

}

}

**else** **if** (name.equals("JEQ")) {

**if** (rMgr.getRegister(rMgr.***swReg***) == 0) {

**if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else**

data = inst.target + x + loc;

rMgr.setRegister(rMgr.***pcReg***, data);

**for** (**int** i = 0; i < loader.inst.length(); i++) {

**if** (loader.inst.getToken(i).addr == data) {

instCnt = i;

**break**;

}

}

}

}

**else** **if** (name.equals("JGT")) {

**if** (rMgr.getRegister(rMgr.***swReg***) > 0) {

**if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else**

data = inst.target + x + loc;

rMgr.setRegister(rMgr.***pcReg***, data);

**for** (**int** i = 0; i < loader.inst.length(); i++) {

**if** (loader.inst.getToken(i).addr == data) {

instCnt = i;

**break**;

}

}

}

}

**else** **if** (name.equals("JLT")) {

**if** (rMgr.getRegister(rMgr.***swReg***) < 0) {

**if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else**

data = inst.target + x + loc;

rMgr.setRegister(rMgr.***pcReg***, data);

**for** (**int** i = 0; i < loader.inst.length(); i++) {

**if** (loader.inst.getToken(i).addr == data) {

instCnt = i;

**break**;

}

}

}

}

**else** **if** (name.equals("JSUB")) {

rMgr.setRegister(rMgr.***lReg***, rMgr.getRegister(rMgr.***pcReg***));

**if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else**

data = inst.target + x + loc;

rMgr.setRegister(rMgr.***pcReg***, data);

**for** (**int** i = 0; i < loader.inst.length(); i++) {

**if** (loader.inst.getToken(i).addr == data) {

instCnt = i;

**break**;

}

}

}

**else** **if** (name.equals("LDA")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***aReg***, data);

}

**else** **if** (name.equals("LDB")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***bReg***, data);

}

**else** **if** (name.equals("LDCH")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 1));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 1));

}

data &= chmask;

rMgr.setRegister(rMgr.***aReg***, data);

}

**else** **if** (name.equals("LDF")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 5));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 5));

}

rMgr.setRegister(rMgr.***fReg***, data);

}

**else** **if** (name.equals("LDL")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***lReg***, data);

}

**else** **if** (name.equals("LDS")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***sReg***, data);

}

**else** **if** (name.equals("LDT")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***tReg***, data);

}

**else** **if** (name.equals("LDX")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***xReg***, data);

}

**if** (name.equals("MUL")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***aReg***, rMgr.getRegister(rMgr.***aReg***) \* data);

}

**else** **if** (name.equals("MULF")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 5));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 5));

}

rMgr.setRegister(rMgr.***fReg***, rMgr.getRegister(rMgr.***fReg***) \* data);

}

**else** **if** (name.equals("MULR")) {

reg1 = Integer.*valueOf*(inst.name.substring(2,3), 16);

reg2 = Integer.*valueOf*(inst.name.substring(3,4), 16);

data = rMgr.getRegister(reg1) \* rMgr.getRegister(reg2);

rMgr.setRegister(reg2, data);

}

**else** **if** (name.equals("OR")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***aReg***, rMgr.getRegister(rMgr.***aReg***) | data);

}

**else** **if** (name.equals("RD")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 1));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 1));

}

rMgr.setRegister(rMgr.***aReg***, rMgr.charToInt(rMgr.readDevice(data, 3)));

}

**else** **if** (name.equals("RMO")) {

reg1 = Integer.*valueOf*(inst.name.substring(2,3), 16);

reg2 = Integer.*valueOf*(inst.name.substring(3,4), 16);

rMgr.setRegister(reg2, rMgr.getRegister(reg1));

}

**else** **if** (name.equals("RSUB")) {

rMgr.setRegister(rMgr.***pcReg***, rMgr.getRegister(rMgr.***lReg***));

data = rMgr.getRegister(rMgr.***lReg***);

**for** (**int** i = 0; i < loader.inst.length(); i++) {

**if** (loader.inst.getToken(i).addr == data) {

instCnt = i;

**break**;

}

}

}

**else** **if** (name.equals("RMO")) {

reg1 = Integer.*valueOf*(inst.name.substring(2,3), 16);

reg2 = Integer.*valueOf*(inst.name.substring(3,4), 16);

rMgr.setRegister(reg2, rMgr.getRegister(reg1));

}

**else** **if** (name.equals("SHIFTL")) {

reg1 = Integer.*valueOf*(inst.name.substring(2,3), 16);

reg2 = Integer.*valueOf*(inst.name.substring(3,4), 16);

rMgr.setRegister(rMgr.***swReg***, rMgr.getRegister(reg1) << reg2);

}

**else** **if** (name.equals("SHIFTL")) {

reg1 = Integer.*valueOf*(inst.name.substring(2,3), 16);

reg2 = Integer.*valueOf*(inst.name.substring(3,4), 16);

rMgr.setRegister(rMgr.***swReg***, rMgr.getRegister(reg1) >> reg2);

}

**else** **if** (name.equals("STA")){

**if** (mode == sim) {

data = inst.target + x + loc;

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

rMgr.setMemory(data, rMgr.intToChar(rMgr.getRegister(rMgr.***aReg***)), 3);

}

**else** **if** (name.equals("STB")){

**if** (mode == sim) {

data = inst.target + x + loc;

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

rMgr.setMemory(data, rMgr.intToChar(rMgr.getRegister(rMgr.***bReg***)), 3);

}

**else** **if** (name.equals("STCH")){

**if** (mode == sim) {

data = inst.target + x + loc;

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

rMgr.setMemory(data, rMgr.intToChar(rMgr.getRegister(rMgr.***aReg***)&chmask), 3);

}

**else** **if** (name.equals("STF")){

**if** (mode == sim) {

data = inst.target + x + loc;

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

rMgr.setMemory(data, rMgr.intToChar(rMgr.getRegister(rMgr.***fReg***)&chmask), 5);

}

**else** **if** (name.equals("STL")){

**if** (mode == sim) {

data = inst.target + x + loc;

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

rMgr.setMemory(data, rMgr.intToChar(rMgr.getRegister(rMgr.***lReg***)), 3);

}

**else** **if** (name.equals("STS")){

**if** (mode == sim) {

data = inst.target + x + loc;

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

rMgr.setMemory(data, rMgr.intToChar(rMgr.getRegister(rMgr.***sReg***)), 3);

}

**else** **if** (name.equals("STSW")){

**if** (mode == sim) {

data = inst.target + x + loc;

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

rMgr.setMemory(data, rMgr.intToChar(rMgr.getRegister(rMgr.***swReg***)), 3);

}

**else** **if** (name.equals("STT")){

**if** (mode == sim) {

data = inst.target + x + loc;

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

rMgr.setMemory(data, rMgr.intToChar(rMgr.getRegister(rMgr.***tReg***)), 3);

}

**else** **if** (name.equals("STX")){

**if** (mode == sim) {

data = inst.target + x + loc;

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

rMgr.setMemory(data, rMgr.intToChar(rMgr.getRegister(rMgr.***xReg***)), 3);

}

**if** (name.equals("SUB")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***aReg***, rMgr.getRegister(rMgr.***aReg***) - data);

}

**else** **if** (name.equals("SUBF")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 5));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 5));

}

rMgr.setRegister(rMgr.***fReg***, rMgr.getRegister(rMgr.***fReg***) - data);

}

**else** **if** (name.equals("SUBR")) {

reg1 = Integer.*valueOf*(inst.name.substring(2,3), 16);

reg2 = Integer.*valueOf*(inst.name.substring(3,4), 16);

data = rMgr.getRegister(reg2) - rMgr.getRegister(reg1);

rMgr.setRegister(reg2, data);

}

**else** **if** (name.equals("TD")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 1));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 1));

}

rMgr.testDevice(data);

}

**else** **if** (name.equals("TIX")) {

rMgr.setRegister(rMgr.***xReg***, rMgr.getRegister(rMgr.***xReg***)+1);

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 3));

}

rMgr.setRegister(rMgr.***swReg***, data-rMgr.getRegister(rMgr.***xReg***));

}

**else** **if** (name.equals("TIXR")) {

reg1 = Integer.*valueOf*(inst.name.substring(2,3), 16);

rMgr.setRegister(rMgr.***xReg***, rMgr.getRegister(rMgr.***xReg***)+1);

rMgr.setRegister(rMgr.***swReg***, rMgr.getRegister(rMgr.***xReg***)-rMgr.getRegister(reg1));

}

**else** **if** (name.equals("WD")) {

**if** (mode == sim) {

data = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 1));

}

**else** **if** (mode == imm) {

data = inst.target;

}

**else** **if** (mode == ind) {

target = rMgr.charToInt(rMgr.getMemory(inst.target + x + loc, 3));

data = rMgr.charToInt(rMgr.getMemory(target, 1));

}

rMgr.writeDevice(data, rMgr.intToChar(rMgr.getRegister(rMgr.***aReg***)), 3);

}

**return** name;

}

/\*\*

\* 남은 모든 instruction이 수행된 모습을 보인다.

\*/

**public** **void** allStep(SicLoader loader) {

}

/\*\*

\* 각 단계를 수행할 때 마다 관련된 기록을 남기도록 한다.

\*/

**public** **void** addLog(String log) {

}

}

VisualSimulator.java

import java.awt.BorderLayout;

import java.awt.EventQueue;

import java.awt.FileDialog;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.border.EmptyBorder;

import javax.swing.JLabel;

import javax.swing.JTextField;

import javax.swing.DefaultListModel;

import javax.swing.JButton;

import javax.swing.JTabbedPane;

import javax.swing.JDesktopPane;

import javax.swing.JLayeredPane;

import javax.swing.JEditorPane;

import javax.swing.JSeparator;

import java.awt.Color;

import javax.swing.JTextArea;

import javax.swing.SwingConstants;

import java.awt.event.ActionListener;

import java.io.File;

import java.awt.event.ActionEvent;

import javax.swing.border.CompoundBorder;

import javax.swing.border.LineBorder;

import javax.swing.JComboBox;

import javax.swing.JList;

import javax.swing.JScrollPane;

import java.awt.Font;

import javax.swing.JScrollBar;

import javax.swing.ScrollPaneConstants;

public class VisualSimulator extends JFrame {

ResourceManager resourceManager = new ResourceManager();

SicLoader sicLoader = new SicLoader(resourceManager);

SicSimulator sicSimulator = new SicSimulator(resourceManager);

private JPanel contentPane;

private JTextField fileName;

private JTextField programName;

private JTextField startAdress;

private JTextField lengthOfProgram;

private JTextField adressOfFirstInstruction;

private JTextField target;

private JTextField decA;

private JTextField hexA;

private JTextField decX;

private JTextField hexX;

private JTextField decL;

private JTextField hexL;

private JTextField decB;

private JTextField hexB;

private JTextField decS;

private JTextField hexS;

private JTextField device;

private JTextField decT;

private JTextField hexT;

private JTextField regF;

private JTextField decPC;

private JTextField hexPC;

private JTextField regSW;

private JList<?> list;

private JTextArea textArea;

/\*\*

\* 프로그램 로드 명령을 전달한다.

\*/

public static void main(String[] args) {

VisualSimulator frame = new VisualSimulator();

frame.setTitle("SIC/XE");

frame.setVisible(true);

}

public void load(File string){

//...

sicLoader.load(string);

};

/\*\*

\* 하나의 명령어만 수행할 것을 SicSimulator에 요청한다.

\*/

public void oneStep(){

textArea.append(sicSimulator.oneStep(sicLoader) + "\n");

update();

};

/\*\*

\* 남아있는 모든 명령어를 수행할 것을 SicSimulator에 요청한다.

\*/

public void allStep(){

int before=0;

while (sicSimulator.instCnt <= sicLoader.inst.length()) {

before = sicSimulator.instCnt;

oneStep();

if (before == sicSimulator.instCnt)

break;

update();

}

};

/\*\*

\* 화면을 최신값으로 갱신하는 역할을 수행한다.

\*/

public void update(){

programName.setText(sicLoader.programName);

DefaultListModel model = new DefaultListModel();

for (int i = 0; i < sicLoader.inst.length(); i++) {

model.addElement(sicLoader.inst.getToken(i).name);

}

list.setModel(model);

lengthOfProgram.setText(String.format("%06X",sicLoader.programLength));

startAdress.setText(sicLoader.programStart);

decA.setText(String.format("%06d", sicLoader.rMgr.getRegister(0)));

hexA.setText(String.format("%06X", sicLoader.rMgr.getRegister(0)));

decX.setText(String.format("%06d", sicLoader.rMgr.getRegister(1)));

hexX.setText(String.format("%06X", sicLoader.rMgr.getRegister(1)));

decL.setText(String.format("%06d", sicLoader.rMgr.getRegister(2)));

hexL.setText(String.format("%06X", sicLoader.rMgr.getRegister(2)));

decB.setText(String.format("%06d", sicLoader.rMgr.getRegister(3)));

hexB.setText(String.format("%06X", sicLoader.rMgr.getRegister(3)));

decS.setText(String.format("%06d", sicLoader.rMgr.getRegister(4)));

hexS.setText(String.format("%06X", sicLoader.rMgr.getRegister(4)));

decT.setText(String.format("%06d", sicLoader.rMgr.getRegister(5)));

hexT.setText(String.format("%06X", sicLoader.rMgr.getRegister(5)));

regF.setText(String.format("%06X", sicLoader.rMgr.getRegister(6)));

hexPC.setText(String.format("%06X", sicLoader.rMgr.getRegister(7)));

decPC.setText(String.format("%06d", sicLoader.rMgr.getRegister(7)));

regSW.setText(String.format("%06X", sicLoader.rMgr.getRegister(8)));

adressOfFirstInstruction.setText(String.format("%06X",sicLoader.inst.getToken(0).addr));

if (sicSimulator.instCnt < sicLoader.inst.length())

target.setText(String.format("%06X",sicLoader.inst.getToken(sicSimulator.instCnt).addr));

if (resourceManager.curdev != null)

device.setText(String.format("%02X", resourceManager.curdev));

};

/\*\*

\* Create the frame.

\*/

public VisualSimulator() {

setTitle("SIC/XE Simulator");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setBounds(100, 100, 556, 732);

contentPane = new JPanel();

contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

setContentPane(contentPane);

contentPane.setLayout(null);

JLabel File = new JLabel("File Name :");

File.setBounds(12, 13, 76, 15);

contentPane.add(File);

fileName = new JTextField();

fileName.setBounds(85, 10, 116, 21);

contentPane.add(fileName);

fileName.setColumns(10);

JButton btnOpen = new JButton("open");

btnOpen.setBounds(213, 10, 76, 21);

FileDialog dialog = new FileDialog(this, "파일 열기", FileDialog.LOAD);

btnOpen.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

dialog.setVisible(true);

File file = new File(dialog.getDirectory()+dialog.getFile());

fileName.setText(dialog.getFile());

load(file);

update();

}

});

contentPane.add(btnOpen);

JLayeredPane layeredPane = new JLayeredPane();

layeredPane.setBounds(12, 38, 260, 146);

layeredPane.setBorder(new CompoundBorder(new LineBorder(new Color(0, 0, 0)), null));

contentPane.add(layeredPane);

JLabel lblHheaderRecord = new JLabel("H (Header Record)");

lblHheaderRecord.setForeground(Color.BLACK);

lblHheaderRecord.setBounds(12, 0, 112, 20);

layeredPane.add(lblHheaderRecord);

JLabel lblProgramName = new JLabel("Program name :");

lblProgramName.setBounds(12, 22, 104, 20);

layeredPane.add(lblProgramName);

programName = new JTextField();

programName.setEditable(false);

programName.setBounds(128, 22, 116, 21);

layeredPane.add(programName);

programName.setColumns(10);

JLabel lblNewLabel\_1 = new JLabel("Start Adress of \r\nObject Program :");

lblNewLabel\_1.setVerticalAlignment(SwingConstants.TOP);

lblNewLabel\_1.setBounds(12, 52, 191, 20);

layeredPane.add(lblNewLabel\_1);

startAdress = new JTextField();

startAdress.setEditable(false);

startAdress.setBounds(128, 69, 116, 21);

layeredPane.add(startAdress);

startAdress.setColumns(10);

JLabel lblNewLabel\_2 = new JLabel("Length of Program : ");

lblNewLabel\_2.setBounds(12, 100, 116, 15);

layeredPane.add(lblNewLabel\_2);

lengthOfProgram = new JTextField();

lengthOfProgram.setEditable(false);

lengthOfProgram.setBounds(128, 102, 116, 21);

layeredPane.add(lengthOfProgram);

lengthOfProgram.setColumns(10);

JPanel panel = new JPanel();

panel.setBounds(284, 38, 232, 97);

panel.setBorder(new LineBorder(new Color(0, 0, 0)));

contentPane.add(panel);

panel.setLayout(null);

JLabel lblEendRecord = new JLabel("E (End Record)");

lblEendRecord.setBounds(12, 0, 88, 23);

panel.add(lblEendRecord);

JLabel lblNewLabel\_3 = new JLabel("<html>Address of First Instruction<br> in Object Program :\r\n");

lblNewLabel\_3.setBounds(12, 20, 208, 36);

panel.add(lblNewLabel\_3);

adressOfFirstInstruction = new JTextField();

adressOfFirstInstruction.setEditable(false);

adressOfFirstInstruction.setBounds(104, 66, 116, 21);

panel.add(adressOfFirstInstruction);

adressOfFirstInstruction.setColumns(10);

JLabel lblTargetAddress = new JLabel("Target Address :");

lblTargetAddress.setBounds(284, 198, 97, 15);

contentPane.add(lblTargetAddress);

target = new JTextField();

target.setEditable(false);

target.setBounds(400, 195, 116, 21);

contentPane.add(target);

target.setColumns(10);

JPanel panel\_1 = new JPanel();

panel\_1.setBounds(12, 198, 260, 278);

panel\_1.setBorder(new LineBorder(new Color(0, 0, 0)));

contentPane.add(panel\_1);

panel\_1.setLayout(null);

JLabel lblNewLabel\_5 = new JLabel("Register");

lblNewLabel\_5.setBounds(12, 0, 57, 24);

panel\_1.add(lblNewLabel\_5);

JLabel lblNewLabel\_6 = new JLabel("A (#0)");

lblNewLabel\_6.setBounds(12, 41, 57, 15);

panel\_1.add(lblNewLabel\_6);

JLabel lblNewLabel\_7 = new JLabel("X (#1)");

lblNewLabel\_7.setBounds(12, 66, 57, 15);

panel\_1.add(lblNewLabel\_7);

JLabel lblNewLabel\_8 = new JLabel("L (#2)");

lblNewLabel\_8.setBounds(12, 91, 57, 15);

panel\_1.add(lblNewLabel\_8);

JLabel lblNewLabel\_9 = new JLabel("B (#3)");

lblNewLabel\_9.setBounds(12, 116, 57, 15);

panel\_1.add(lblNewLabel\_9);

JLabel lblNewLabel\_10 = new JLabel("S (#4)");

lblNewLabel\_10.setBounds(12, 141, 57, 15);

panel\_1.add(lblNewLabel\_10);

JLabel lblNewLabel\_11 = new JLabel("Dec");

lblNewLabel\_11.setBounds(60, 21, 57, 15);

panel\_1.add(lblNewLabel\_11);

JLabel lblNewLabel\_12 = new JLabel("Hex");

lblNewLabel\_12.setBounds(154, 21, 57, 15);

panel\_1.add(lblNewLabel\_12);

decA = new JTextField();

decA.setEditable(false);

decA.setBounds(60, 38, 79, 21);

panel\_1.add(decA);

decA.setColumns(10);

hexA = new JTextField();

hexA.setEditable(false);

hexA.setColumns(10);

hexA.setBounds(154, 38, 79, 21);

panel\_1.add(hexA);

decX = new JTextField();

decX.setEditable(false);

decX.setColumns(10);

decX.setBounds(60, 63, 79, 21);

panel\_1.add(decX);

hexX = new JTextField();

hexX.setEditable(false);

hexX.setColumns(10);

hexX.setBounds(154, 63, 79, 21);

panel\_1.add(hexX);

decL = new JTextField();

decL.setEditable(false);

decL.setColumns(10);

decL.setBounds(60, 88, 79, 21);

panel\_1.add(decL);

hexL = new JTextField();

hexL.setEditable(false);

hexL.setColumns(10);

hexL.setBounds(154, 88, 79, 21);

panel\_1.add(hexL);

decB = new JTextField();

decB.setEditable(false);

decB.setColumns(10);

decB.setBounds(60, 113, 79, 21);

panel\_1.add(decB);

hexB = new JTextField();

hexB.setEditable(false);

hexB.setColumns(10);

hexB.setBounds(154, 113, 79, 21);

panel\_1.add(hexB);

decS = new JTextField();

decS.setEditable(false);

decS.setColumns(10);

decS.setBounds(60, 138, 79, 21);

panel\_1.add(decS);

hexS = new JTextField();

hexS.setEditable(false);

hexS.setColumns(10);

hexS.setBounds(154, 138, 79, 21);

panel\_1.add(hexS);

JLabel lblF = new JLabel("F (#6)");

lblF.setBounds(12, 193, 57, 15);

panel\_1.add(lblF);

JLabel lblT = new JLabel("T (#5)");

lblT.setBounds(12, 168, 57, 15);

panel\_1.add(lblT);

decT = new JTextField();

decT.setEditable(false);

decT.setColumns(10);

decT.setBounds(60, 165, 79, 21);

panel\_1.add(decT);

hexT = new JTextField();

hexT.setEditable(false);

hexT.setColumns(10);

hexT.setBounds(154, 165, 79, 21);

panel\_1.add(hexT);

regF = new JTextField();

regF.setEditable(false);

regF.setColumns(10);

regF.setBounds(60, 190, 173, 21);

panel\_1.add(regF);

JLabel label\_2 = new JLabel("SW (#9)");

label\_2.setBounds(12, 246, 57, 15);

panel\_1.add(label\_2);

JLabel label\_3 = new JLabel("PC (#8)");

label\_3.setBounds(12, 221, 57, 15);

panel\_1.add(label\_3);

decPC = new JTextField();

decPC.setEditable(false);

decPC.setColumns(10);

decPC.setBounds(60, 218, 79, 21);

panel\_1.add(decPC);

hexPC = new JTextField();

hexPC.setEditable(false);

hexPC.setColumns(10);

hexPC.setBounds(154, 218, 79, 21);

panel\_1.add(hexPC);

regSW = new JTextField();

regSW.setEditable(false);

regSW.setColumns(10);

regSW.setBounds(60, 243, 173, 21);

panel\_1.add(regSW);

JLabel lblNewLabel\_13 = new JLabel("Instruction :");

lblNewLabel\_13.setBounds(284, 232, 76, 15);

contentPane.add(lblNewLabel\_13);

JScrollPane scrollPane = new JScrollPane();

scrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL\_SCROLLBAR\_ALWAYS);

scrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL\_SCROLLBAR\_ALWAYS);

scrollPane.setBounds(284, 258, 134, 218);

contentPane.add(scrollPane);

list = new JList();

scrollPane.setViewportView(list);

JLabel lblNewLabel\_14 = new JLabel("\uC0AC\uC6A9\uC911\uC778 \uC7A5\uCE58");

lblNewLabel\_14.setBounds(442, 238, 86, 15);

contentPane.add(lblNewLabel\_14);

device = new JTextField();

device.setEditable(false);

device.setBounds(440, 263, 88, 21);

contentPane.add(device);

device.setColumns(10);

JButton btnNewButton = new JButton("\uC2E4\uD589(1step)");

btnNewButton.setFont(new Font("굴림", Font.PLAIN, 11));

btnNewButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

oneStep();

}

});

btnNewButton.setBounds(430, 377, 98, 23);

contentPane.add(btnNewButton);

JButton btnall = new JButton("\uC2E4\uD589(All)");

btnall.setFont(new Font("굴림", Font.PLAIN, 11));

btnall.setBounds(430, 409, 98, 23);

btnall.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

allStep();

}

});

contentPane.add(btnall);

JButton button = new JButton("\uC885\uB8CC");

button.setFont(new Font("굴림", Font.PLAIN, 11));

button.setBounds(430, 442, 98, 23);

contentPane.add(button);

JLabel lblNewLabel\_15 = new JLabel("Log (\uBA85\uB839\uC5B4 \uC218\uD589 \uAD00\uB828) :");

lblNewLabel\_15.setBounds(12, 490, 144, 15);

button.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

System.exit(0);

}

});

contentPane.add(lblNewLabel\_15);

textArea = new JTextArea();

JScrollPane scrollPane2 = new JScrollPane(textArea);

scrollPane2.setBounds(12, 515, 516, 156);

contentPane.add(scrollPane2);

}

}