시스템 프로그래밍

Project #2

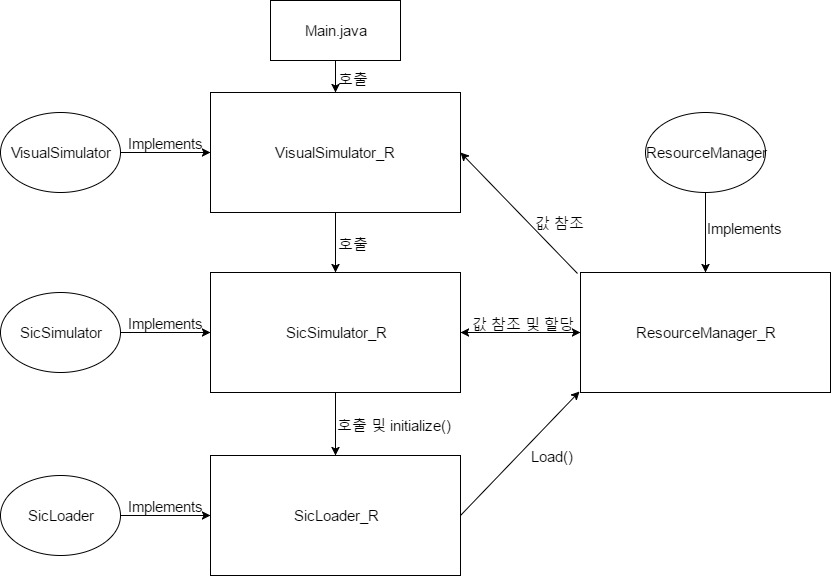
|  |  |
| --- | --- |
| 소속학과 | 컴퓨터학부 |
| 학번 | 20122366 |
| 성명 | 김홍석 |
| 출석번호 | 139(컴학 수업) |
| 담당교수명 | 최재영 교수님 |
| 일자 | 2017-06-09 |

목차

1. 동기 / 목적
2. 설계 / 구현 아이디어
3. 소스 코드
4. 수행 결과
5. 결론 및 보충할 점
6. 동기 / 목적

어셈블러를 통해 소스코드를 변환시키면 Object code로 구성 된 Object Program이 생성된다. 이 Object Program을 실행시킨다는 것은 컴퓨터의 메인 메모리에 Object Program을 올린다는 것을 뜻한다. 이 메모리에 올릴 때 사용하는 것이 바로 Loader 이다. 이 Loader를 직접 설계하고 구성하며 프로그래밍 하는 것이 이번 과제의 목적이다. 이러한 과정에서 Loader의 동작 원리, 각 어셈블리 명령어들이 어떻게 레지스터에 값 할당 및 참조를 하는지에 대해 이해한다. 이번 과제의 Object Program은 ControlSection 방식으로 생성되었기 때문에 이에 대한 이해도 할 수 있다. 최종적으로 구현한 Loader를 JAVA GUI로 나타내는 시뮬레이터 프로그램을 만드는 것이 목적이다.

1. 설계 / 구현 아이디어



* 1. VisualSimulator\_R

Main.java에서 호출하는 클래스이다. 전체적인 GUI를 구성하고 유저가 발생시킨 이벤트를 주로 처리한다. oneStep 혹은 allStep 으로 실행되면 그 때 마다 ResourceManger\_R에 저장된 각종 값들을 가져와서 GUI에 뿌려준다. 또한 initialize() 함수를 통해 SicSimulator\_R 클래스를 호출하며 ResourceManager\_R 클래스도 이 클래스에서 인스턴스를 생성해준다.

* 1. SicSimultor\_R

VisualSimulator\_R 에 의해서 호출되며 SicLoader\_R을 호출한다. 그리고 ResourceManager\_R 인스턴스의 각 레지스터, 메모리 값을 초기화해준다. SicLoader\_R을 호출한 후 메모리에 직접 load까지 실행시킨다. 또한 메모리에 올라간 코드를 파싱하여 분석하고 각 명령어들을 실행시킨다. 이 과정에서 메모리에 설정한 값이나 레지스터 값들은 ResourceManager\_R 클래스를 통해서 참조 및 할당을 해준다. 즉, 시뮬레이터의 전반적인 기능이 모두 이 클래스에 구현을 해 놓았다.

* 1. SicLoader\_R

SicSimulator\_R에 의해서 호출되며 메모리에 인풋 파일의 Object Program을 올리는 기능을 수행한다. 또한 Object Program에서 M 레코드의 정보를 참조하여 M 레코드가 가르키는 위치에 실제 메모리 주소 값을 업데이트 해준다.

* 1. ResourceManager\_R

메모리 및 각종 레지스터들이 저장되어 있는 클래스이다. 이 클래스를 통해 메모리와 레지스터들을 관리하며 다른 클래스와의 값 전달 및 값을 받아서 업데이트 해준다.

1. 소스 코드

--------------------------------------Main.java------------------------------------------------

**import** javax.swing.JFrame;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

VisualSimulator\_R simulator = **new** VisualSimulator\_R();

simulator.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

simulator.setLocation(200, 50);

simulator.setVisible(**true**);

simulator.setResizable(**false**);

}

}

--------------------------------------ResourceManager.java-------------------------------------

**public** **interface** ResourceManager {

//메모리 영역을 초기화 하는 메소드

**public** **void** initializeMemory();

//각 레지스터 값을 초기화 하는 메소드

**public** **void** initializeRegister();

//디바이스 접근에 대한 메소드

//디바이스는 각 이름과 매칭되는 파일로 가정한다

//(F1 이라는 디바이스를 읽으면 F1 이라는 파일에서 값을 읽는다.)

//해당 디바이스(파일)를 사용 가능한 상태로 만드는 메소드

**public** **void** initialDevice();

//선택한 디바이스(파일)에 값을 쓰는 메소드. 파라미터는 변경 가능하다.

**public** **void** writeDevice(String devName, **int** data);

//선택한 디바이스(파일)에서 값을 읽는 메소드. 파라미터는 변경 가능하다.

**public** **int** readDevice(String devName);

//메모리 영역에 값을 쓰는 메소드

**public** **void** setMemory(**int** locate, String data, **int** size);

//레지스터에 값을 세팅하는 메소드. regNum은 레지스터 종류를 나타낸다.

**public** **void** setRegister(**int** regNum, **int** value);

//메모리 영역에서 값을 읽어오는 메소드

**public** String getMemory(**int** locate, **int** size);

//레지스터에서 값을 가져오는 메소드

**public** **int** getRegister(**int** regNum);

// 레지스터 -> 메모리, 디바이스 데이터 타입 변경. 필요한 경우 구현

// public byte[] getRegister(int data);

// 메모리, 디바이스 -> 레지스터 데이터 타입 변경. 필요한 경우 구현

// public int getRegister(byte[] data);

}

--------------------------------------SicLoader.java-------------------------------------------

**import** java.io.File;

**public** **interface** SicLoader {

//목적코드를 읽어 메모리에 로드한다.

//목적코드의 각 헤더(H, T, M 등)를 읽어 동작을 수행한다.

**public** **void** load(File objFile, ResourceManager\_R rMgr);

}

--------------------------------------SicSimulator.java----------------------------------------

**import** java.io.File;

**public** **interface** SicSimulator {

//시뮬레이터를 동작시키기 위한 세팅을 수행한다.

//메모리 작업 등 실질적인 초기화 작업을 수행한다.

**public** **void** initialize(File objFile, ResourceManager\_R rMgr);

//하나의 명령어만 수행한다. 해당 명령어가 수행되고 난 값의 변화를

//보여주고, 다음 명령어를 포인팅한다.

//실질적인 동작을 수행하는 메소드

**public** **void** oneStep();

//남은 명령어를 모두 수행하는 메소드.

//목적 코드를 모두 수행하고 난 값의 변화를 보여준다.

//실질적인 동작을 수행하는 메소드

**public** **void** allStep();

}

--------------------------------------VisualSimulaotr.java-------------------------------------

**import** java.io.File;

**public** **interface** VisualSimulator {

//시뮬레이터를 동작시키기 위한 세팅을 수행한다.

//sic 시뮬레이터를 통해 로더를 수행시키고, 로드된 값들을 읽어 보여주어

//스텝을 진행할 수 있는 상태로 만들어 놓는다.

**public** **void** initialize(File objFile, ResourceManager\_R rMgr);

//하나의 명령어만 수행하는 메소드로써 sic 시뮬레이터에게 작업을 전달한다.

**public** **void** oneStep();

//남은 명령어를 모두 수행하는 메소드로써 sic 시뮬레이터에 작업을 전달

**public** **void** allStep();

//작업이 완료되었을 때 변화된 결과를 화면에 업데이트

**public** **void** update();

//이외의 GUI 관련 메소드들

}

--------------------------------------ResourceManager\_R.java-----------------------------------

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.util.Vector;

public class ResourceManager\_R implements ResourceManager {

public String progName[] = new String[3];

public int startAddr[] = new int[3];

public String startAddr\_string[] = new String[3];

public int progLength[] = new int[3];

public String progLength\_string[] = new String[3];

public String firstInst;

public boolean end = false;

public String currentDevice = "";

public int sectionIndex=0;

public int beforeIndex=0;

public Vector<String> memoryField = new Vector<>(); //현재 메모리의 위치를 gui상을 나타내기 위해 vector자료구조 사용

public Vector<String> instField = new Vector<>(); //현재 명령어의 종류를 gui상을 나타내기 위해 vector자료구조 사용

public File inputfile = null;

public File outputfile = null;

public FileInputStream inputStream = null;

public FileOutputStream outputStream = null;

public int A;

public int X;

public int L;

public int PC;

public int SW; //비교를 하는데 참조하는 레지스터. > 일 때 2 ,== 1,< -1로 설정함

public int B;

public int S;

public int T;

public int F;

public int TA;

public final static int MEM\_SIZE = 0x100000 \* 2; // XE 머신의 경우 메모리사이즈 -> 1Mbyte =

// 2^20 byte 그런데 String으로 구현했으므로 String의 2바이트는 XE머신 메모리의 1바이트이다. 그러므로 2를 곱해줌.

StringBuffer memory = new StringBuffer(MEM\_SIZE); // 그러므로 모든 주소 관련 연산에는 \*2가 붙어있으며, 명시적으로 하기 위해 다항식을 풀어서 1\*2 등으로 표현하였다.

@Override //메모리 \*로 초기화

public void initializeMemory() {

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

memory.insert(i, '\*');

}

}

@Override //레지스터 값들 초기화

public void initializeRegister() {

A = 0;

X = 0;

L = 0;

PC = 0;

SW = 0;

B = 0;

S = 0;

T = 0;

F = 0;

TA = 0;

}

@Override //지정된 디바이스 'F1' 과 '05'로 입출력 스트림을 열어줌

public void initialDevice() {

try {

inputStream = new FileInputStream(new File("F1.txt"));

outputStream = new FileOutputStream(new File("05.txt"));

} catch (FileNotFoundException e) {

e.printStackTrace();

}

}

@Override

public void writeDevice(String devName, int data) {

if (devName.equals("05")) {

try {

outputStream.write(data); //data 변수의 하위 1바이트만 write함

outputStream.flush(); //스트림에 있는 버퍼를 파일로 보내줌.

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

@Override

public int readDevice(String devName) {

int read = -1;

if (devName.equals("F1")) {

try {

read = inputStream.read(); //파일의 1바이트만 read해옴

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

if(read == -1)

return 0;

return read;

}

return read;

}

@Override //큰 StringBuffer로 구현한 메모리의 값을 할당해주는 메소드. locate위치에 size 크기만큼 data를 입력해줌.

public void setMemory(int locate, String data, int size) {

memory.replace(locate, size, data);

}

// 각 레지스터 번호에 맞게 스위치로 분기하며

// 그때의 각 레지스터 값을 리턴해줌.

@Override

public void setRegister(int regNum, int value) {

switch (regNum) {

case 0:

A = value;

break;

case 1:

X = value;

break;

case 2:

L = value;

break;

case 3:

B = value;

break;

case 4:

S = value;

break;

case 5:

T = value;

break;

case 6:

F = value;

break;

case 7:

TA = value;

break;

case 8:

PC = value;

break;

case 9:

SW = value;

break;

}

}

@Override

public String getMemory(int locate, int size) {

return memory.substring(locate, size);

}

//레지스터의 값을 반환해준다

@Override

public int getRegister(int regNum) {

switch(regNum){

case 0:

return A;

case 1:

return X;

case 2:

return L;

case 3:

return B;

case 4:

return S;

case 5:

return T;

case 6:

return F;

case 7:

return TA;

case 8:

return PC;

case 9:

return SW;

}

return -1; // 에러발생

}

}

--------------------------------------SicLoader\_R.java-----------------------------------------

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.Scanner;

class Modify {

String name[] = new String[15];

int addr[] = new int[15];

int count[] = new int[15];

char sign[] = new char[15];

int section[] = new int[15];

}

class Define {

String name[] = new String[6];

int addr[] = new int[6];

}

public class SicLoader\_R implements SicLoader {

ResourceManager\_R rMgr;

// public BufferedReader reader;

Define define = new Define(); //D 레코드와 M 레코드에 쓸 자료구조

Modify modify = new Modify();

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

\* 설명 : 가상으로 잡은 rMgr.memory 에 인풋으로 들어온 Object Program을 로드시키는 함수.

\* 각 섹션을 참고하여 순서대로 로드해준다. 그 중에 M레코드가 들어오면 실제 메모리에는 수정된 주소를 업데이트 해줘야 하므로

\* 이 또한 처리하였다.

\* 매개 : -

\* 반환 : -

\* ----------------------------------------------------------------------------------

\*/

@Override

public void load(File objFile, ResourceManager\_R rMgr) {

this.rMgr = rMgr;

String line;

int section\_num = 0, d\_num = 0, m\_num = 0;

try {

Scanner scanner = new Scanner(objFile);

while (scanner.hasNextLine()) { //인풋인 오브젝트 파일을 끝까지 읽어들이면서 메모리에 로드함

line = scanner.nextLine();

if (line.equals(""))

continue;

if (line.charAt(0) == 'H') { //H태그의 각종 정보들을 처리.

rMgr.progName[section\_num] = line.substring(1, 7);

rMgr.progLength[section\_num] = Integer.parseInt(line.substring(13, 19), 16) \* 2; //StringBuffer의 1바이트엔 half\_byte짜리 캐릭터가 할당되므로 \*2를 해줌.

rMgr.progLength\_string[section\_num] = String.format("%06X", rMgr.progLength[section\_num]);

if (section\_num > 0) { // 첫번째 섹션이 아니라면 startAddr[0] + 각 프로그램의 길이를 더해준다.

int total = 0;

for (int x = 0; x < section\_num; x++)

total = total + rMgr.progLength[x];

rMgr.startAddr[section\_num] = total;

//StartAddr을 이용하여 뒤에 따라오는 컨트롤 섹션들의 실제 메모리 값 할당.

rMgr.startAddr\_string[section\_num] = String.format("%06X", rMgr.startAddr[section\_num]/2);

define.name[d\_num] = line.substring(1, 7);

define.addr[d\_num++] = rMgr.startAddr[section\_num]/2;

//define의 주소는 StringBuffer의 번지수를 찾아가는 게 아니라 그 안에 들어갈 내용물. \*2를 해줄 필요가 없다

} else { //첫번째 섹션이라면 startAddr의 값을 H레코드의 시작 주소로 초기화

rMgr.startAddr[section\_num] = Integer.parseInt(line.substring(7, 13), 16) \* 2;

rMgr.startAddr\_string[section\_num] = String.format("%06X", rMgr.startAddr[section\_num]/2);

}

} else if (line.charAt(0) == 'E') { //E 태그의 뒤에따라오는 인자는 첫번째 실행가능한 명령어. 이를 처리.

if (line.length() != 1)

rMgr.firstInst = line.substring(1, 7);

section\_num++;

} else if (line.charAt(0) == 'D') { //D태그를 만나면 define 클래스의 변수들에 저장해줌. 뒤에서 Modificaion record가 가르키는 메모리 위치에 맵핑함

define.name[d\_num] = line.substring(1, 7);

define.addr[d\_num++] = Integer.parseInt(line.substring(7, 13), 16);

define.name[d\_num] = line.substring(13, 19);

define.addr[d\_num++] = Integer.parseInt(line.substring(19, 25), 16);

define.name[d\_num] = line.substring(25, 31);

define.addr[d\_num++] = Integer.parseInt(line.substring(31), 16);

} else if (line.charAt(0) == 'M') { //M태그를 따로 modify 클래스 변수에 저장. 메모리에 일단 T 태그의 레코드를 다 올린 후, 추후 메모리 접근하여 수정함.

modify.addr[m\_num] = Integer.parseInt(line.substring(1, 7), 16) \* 2;

modify.count[m\_num] = Integer.parseInt(line.substring(7, 9), 16);

if (line.contains("-"))

modify.sign[m\_num] = '-';

else

modify.sign[m\_num] = '+';

modify.name[m\_num] = line.substring(10);

modify.section[m\_num] = section\_num;

m\_num++;

} else if (line.charAt(0) == 'T') { //T 레코드의 값들을 넣어줌. StringBuffer의 1바이트엔 half\_byte짜리 캐릭터가 할당되므로 \*2를 해줌.

int locate = Integer.parseInt(line.substring(1, 7), 16) \* 2 + rMgr.startAddr[section\_num];

int size = Integer.parseInt(line.substring(7, 9), 16) \* 2;

rMgr.setMemory(locate, line.substring(9, 9 + size), locate + size);

}

}

for (int i = 0; i < m\_num; i++) //modify와 define을 각각 모두 비교하면서 주소값을 갱신해줌

for (int j = 0; j < d\_num; j++) {

if (modify.name[i].equals(define.name[j])) {

int start = rMgr.startAddr[modify.section[i]]; //각 M 레코드 마다 어느 section에 속해있는지를 기준으로 주소계산.

if (modify.count[i] == 5) { //count에 따라서 주소할당의 방식이 다르다. 5라면 half-byte로 인해서 +1을 해줘서 계산한다.

rMgr.setMemory(start + modify.addr[i] + 1, String.format("%05X", define.addr[j]),

start + modify.addr[i] + 1 + modify.count[i]);

} else if (modify.count[i] == 6) {

if (modify.sign[i] == '-') { //음수라면, 먼저 있던 값 - 찾은 주소 값을 해서 넣어준다.

int original\_value = Integer.parseInt(rMgr.memory.substring(start + modify.addr[i],

start + modify.addr[i] + modify.count[i]), 16);

rMgr.setMemory(start + modify.addr[i],

String.format("%06X", original\_value - define.addr[j]),

start + modify.addr[i] + modify.count[i]);

} else if (modify.sign[i] == '+') { //양수라면 바로 넣어준다.

rMgr.setMemory(start + modify.addr[i], String.format("%06X", define.addr[j]),

start + modify.addr[i] + modify.count[i]);

}

}

}

}

scanner.close(); //스트림 닫아줌

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

--------------------------------------SicSimulator\_R.java--------------------------------------

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Scanner;

import java.util.StringTokenizer;

class inst\_unit{

String operator;

String type;

String opcode;

int operandAmount;

}

public class SicSimulator\_R implements SicSimulator {

public SicLoader\_R loader;

public ResourceManager\_R rMgr;

int inst\_index = 0;

inst\_unit inst\_table[] = new inst\_unit[256];

@Override

public void initialize(File objFile, ResourceManager\_R rMgr) {

this.rMgr = rMgr;

rMgr.initializeMemory(); //rMgr의 메모리 및 레지스터, 디바이스 등을 초기화해준다

rMgr.initializeRegister();

rMgr.initialDevice();

loader = new SicLoader\_R();

loader.load(objFile, rMgr);

if(init\_inst\_file("inst.data")==-1) //오브젝트 코드의 명령어를 비교하기 위한 명령어 테이블 만드는 메소드 호출

{

System.out.println("Inst file Init failed!!");

System.exit(0);

}

}

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

\* 설명 : 메모리에 올라간 오브젝트 코드를 파싱하고 분석하여 명령어를 수행시키는 메소드. 메소드 명 그대로 한 개의 명령어만 수행한다.

\* 매개 : -

\* 반환 : -

\* 주의 : 주소 관련 연산들이 \*2 가 되어있는 부분이 있어서 혼동할 가능성이 있다.

\* 메모리를 String으로 구현했으므로 String의 2바이트는 XE머신 메모리의 1바이트이다. 그러므로 2를 곱해준다.

\* ----------------------------------------------------------------------------------

\*/

@Override

public void oneStep() {

if(rMgr.end == true) //오브젝트의 모든 명령어를 수행했음을 표시하는 boolean 변수 rMgr.end를 통해 프로그램이 끝났는지, 안끝났는지 판별. 끝났으면 추후의 로직을 실행하지 않는다.

return ;

int index = 0;

int currentAddr = rMgr.getRegister(8);

int opcode = Integer.parseInt(rMgr.memory.substring(currentAddr , currentAddr + 2),16); //PC ~ PC+2 만큼 짤라서 확인

int ni\_bit = opcode & 3; //opcode의 하위 2비트만 ni\_bit로 가져옴

int xbpe\_bit = Integer.parseInt(rMgr.memory.substring(currentAddr + 2 , currentAddr + 3),16);

opcode = opcode - ni\_bit; //opcode - ni\_bit를 수행함으로써 optab과 비교할 값을 얻음.

String opcode\_string = String.format("%02X", opcode);

if ((index = search\_opcode(opcode\_string)) > -1) //opcode가 있을때 각 토큰들에선 현재 얼마까지 읽었는지만 기록

{

if (inst\_table[index].type.equals("1")) // format 1 일 때

{

rMgr.setRegister(8, rMgr.getRegister(8)+ 1 \* 2); //다음 PC값 증가

}

else if (inst\_table[index].type.equals("2")) // format 2 일 때 operand로 오는 레지스터에 따라 구분

{

rMgr.setRegister(8, rMgr.getRegister(8)+ 2 \* 2); //다음 PC값 증가

rMgr.memoryField.add(rMgr.memory.substring(currentAddr, currentAddr+4)); //insturction GUI 부분에 나타낼 정보 저장

switch(opcode) {

case 0xB4 : //CLEAR

rMgr.setRegister(Integer.parseInt(rMgr.memory.substring(currentAddr+2, currentAddr+3),16), 0);

rMgr.setRegister(7, 0); //2형식은 타겟이 따로 없으므로 0으로 설정해둠

rMgr.instField.add("CLEAR"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0xA0: // COMPR

int comp1 = rMgr.getRegister(Integer.parseInt(rMgr.memory.substring(currentAddr + 2, currentAddr + 3), 16));

int comp2 = rMgr.getRegister(Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 4), 16)); //레지스터 두개 안에 있는 값들 비교

rMgr.setRegister(7, 0); //2형식은 타겟이 따로 없으므로 0으로 설정해둠

if (comp1 > comp2)

rMgr.setRegister(9, 2);

if (comp1 == comp2)

rMgr.setRegister(9, 1);

if (comp1 < comp2)

rMgr.setRegister(9, -1);

rMgr.instField.add("COMPR"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0xB8 : //TIXR

rMgr.setRegister(1, rMgr.getRegister(1) + 1 \* 2); // X도 주소 관련해서 쓰기 때문에 2를 곱해서 더해줌

rMgr.setRegister(7, 0); //2형식은 타겟이 따로 없으므로 0으로 설정해둠

if(rMgr.getRegister(1) > rMgr.getRegister(Integer.parseInt(rMgr.memory.substring(currentAddr + 2, currentAddr + 3), 16)))

rMgr.setRegister(9, 2);

if(rMgr.getRegister(1) == rMgr.getRegister(Integer.parseInt(rMgr.memory.substring(currentAddr + 2, currentAddr + 3), 16)))

rMgr.setRegister(9, 1);

if(rMgr.getRegister(1) < rMgr.getRegister(Integer.parseInt(rMgr.memory.substring(currentAddr + 2, currentAddr + 3), 16)))

rMgr.setRegister(9, -1);

rMgr.instField.add("TIXR"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

}

}

else if (inst\_table[index].type.equals("3/4"))

{

int targetAddr=0;

int x\_flag=0,b\_flag=0,p\_flag=0,e\_flag=0;

int temp\_sum = 0;

int sign=0;

// target 주소를 구하기 위해 x, b, p, e flag를 설정.

x\_flag = (xbpe\_bit & 8) >> 3;

b\_flag = (xbpe\_bit & 4) >> 2;

p\_flag = (xbpe\_bit & 2) >> 1;

e\_flag = xbpe\_bit & 1;

if( e\_flag == 0){

rMgr.setRegister(8, rMgr.getRegister(8)+ 3 \* 2); //다음 PC값 증가 모든 PC값들도 \*2를 해서 증가시켜줌

rMgr.memoryField.add(rMgr.memory.substring(currentAddr, currentAddr+6)); //insturction GUI 부분에 나타낼 정보 저장

}

else if(e\_flag == 1){

rMgr.setRegister(8, rMgr.getRegister(8)+ 4 \* 2); //다음 PC값 증가

rMgr.memoryField.add(rMgr.memory.substring(currentAddr, currentAddr+8)); //insturction GUI 부분에 나타낼 정보 저장

}

//Target address 설정

if (x\_flag == 1)

temp\_sum += rMgr.getRegister(1); //X를 증가시킬 때, 2를 곱해서 증가시켜놨음에 유의

if (e\_flag == 1) {

;

} else {

if (b\_flag == 1)

temp\_sum += rMgr.getRegister(3);

if (p\_flag == 1)

temp\_sum += rMgr.getRegister(8);

}

switch(opcode) {

case 0x00 : //LDA

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2; //모든 타겟 주소 역시 \*2상태로 처리.

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if (ni\_bit == 3) {

rMgr.setRegister(0, Integer.parseInt(rMgr.memory.substring(targetAddr, targetAddr + 6), 16));

} else if (ni\_bit == 2) {

;

} else if (ni\_bit == 1) { // immediate로 바로 A레지스터에 저장

rMgr.setRegister(0, targetAddr/2);

}

rMgr.instField.add("LDA"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x0C : //STA

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if (ni\_bit == 3) {

rMgr.memory.replace(targetAddr, targetAddr+6, String.format("%06X",rMgr.getRegister(0)));//메모리의 형태가 STring으로 구현했으므로 메모리에 String.format을 이용해 스트링 형태로 저장한다.

} else if (ni\_bit == 2) {

;

} else if (ni\_bit == 1) {

;

}

rMgr.instField.add("STA"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x10 : //STX

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if (ni\_bit == 3) {

rMgr.memory.replace(targetAddr, targetAddr+6, String.format("%06X",rMgr.getRegister(1)/2));//X를 2를 곱해서 증가시켰지만 실제 메모리에 저장할 땐, 2를 나눠서 저장시킴.

} else if (ni\_bit == 2) { // 추후 다른 레지스터에서 사용 할때 2곱해서 불러오도록 코딩함.

;

} else if (ni\_bit == 1) {

;

}

rMgr.instField.add("STX"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x14 : //STL

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if (ni\_bit == 3) {

rMgr.memory.replace(targetAddr, targetAddr+6, String.format("%06X",rMgr.getRegister(2)));//메모리의 형태가 STring으로 구현했으므로 메모리에 String.format을 이용해 스트링 형태로 저장한다.

} else if (ni\_bit == 2) {

;

} else if (ni\_bit == 1) {

;

}

rMgr.instField.add("STL"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x28 : //COMP

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if (ni\_bit == 3) { //값들을 비교하여 SW 레지스터에 논리결과를 저장시킴. 이를 이용해 다른 명령어 수행

int comp = Integer.parseInt(rMgr.memory.substring(targetAddr, targetAddr + 6), 16);

if (comp > rMgr.getRegister(0))

rMgr.setRegister(9, 2);

if (comp == rMgr.getRegister(0))

rMgr.setRegister(9, 1);

if (comp < rMgr.getRegister(0))

rMgr.setRegister(9, -1);

} else if (ni\_bit == 2) {

;

} else if (ni\_bit == 1) {

if(targetAddr > rMgr.getRegister(0))

rMgr.setRegister(9, 2);

if(targetAddr == rMgr.getRegister(0))

rMgr.setRegister(9, 1);

if(targetAddr < rMgr.getRegister(0))

rMgr.setRegister(9, -1);

}

rMgr.instField.add("COMP"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x30 : //JEQ

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

sign = targetAddr & 0x800; //주소 3바이트의 맨 앞자리가 1. 즉 음수라면.

if(sign == 0x800) // 3바이트 앞의 모든 바이트를 1로 맵핑시켜서 음수로 만든다

{

targetAddr = targetAddr | 0xFFFFF000;

}

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if(rMgr.getRegister(9) == 1) // 같다면 다음 PC값에 targetAddr을 할당해 줌. = Jump의 기능

rMgr.setRegister(8, targetAddr);

rMgr.instField.add("JEQ"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x38 : //JLT

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

sign = targetAddr & 0x800; //주소 3바이트의 맨 앞자리가 1. 즉 음수라면.

if(sign == 0x800) // 3바이트 앞의 모든 바이트를 1로 맵핑시켜서 음수로 만든다

{

targetAddr = targetAddr | 0xFFFFF000;

}

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if(rMgr.getRegister(9) == -1) // 작다면 다음 PC값에 targetAddr을 할당해 줌. = Jump의 기능

rMgr.setRegister(8, targetAddr);

rMgr.instField.add("JLT"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x3C : //J

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

sign = targetAddr & 0x800; //주소 3바이트의 맨 앞자리가 1. 즉 음수라면.

if(sign == 0x800) // 3바이트 앞의 모든 바이트를 1로 맵핑시켜서 음수로 만든다

{

targetAddr = targetAddr | 0xFFFFF000;

}

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if (ni\_bit == 3) { // PC값에 targetAddr을 할당해 줌. = Jump의 기능

rMgr.setRegister(8, targetAddr);

} else if (ni\_bit == 2) {

rMgr.setRegister(8, Integer.parseInt(rMgr.memory.substring(targetAddr, targetAddr + 6), 16) \* 2);

} else if (ni\_bit == 1) {

rMgr.setRegister(8, targetAddr);

}

if(rMgr.getRegister(8) == rMgr.startAddr[0])//J 혹은 RSUB로 메인 프로그램이 끝나면 맨 처음 시작주소로 PC값이 할당된다. 이를 이용해 프로그램의 끝을 알린다.

rMgr.end = true;

rMgr.instField.add("J"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x48 : //JSUB

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

rMgr.setRegister(2, rMgr.getRegister(8)); //JSUB의 명세대로 현재 PC값을 L 레지스터에 저장 후, target주소를 PC값에 할당

rMgr.setRegister(8, targetAddr);

for(int x= 0; x<3;x++){ //JSUB를 수행하면 controlSection이 바뀌므로 GUI상에 나타나는 프로그램 이름, 시작주소 등의 정보가 바뀌어야한다.

if(targetAddr == rMgr.startAddr[x]) //그러므로 index값을 설정해줘서 바뀌도록 프로그래밍 하였다.

{

rMgr.beforeIndex = rMgr.sectionIndex;

rMgr.sectionIndex = x;

}

}

rMgr.instField.add("JSUB"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x4C : //RSUB

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

rMgr.setRegister(8, rMgr.getRegister(2)); //pc에 (L)값을 넣어줌

if(rMgr.getRegister(8) == rMgr.startAddr[0]) //J 혹은 RSUB로 메인 프로그램이 끝나면 맨 처음 시작주소로 PC값이 할당된다. 이를 이용해 프로그램의 끝을 알린다.

rMgr.end = true;

rMgr.currentDevice = "";

rMgr.sectionIndex = rMgr.beforeIndex; //JSUB로 서브루틴 부르기 전의 index로 돌아간다.

rMgr.instField.add("RSUB"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x50 : //LDCH

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if (ni\_bit == 3) {

rMgr.setRegister(0, Integer.parseInt(rMgr.memory.substring(targetAddr, targetAddr + 2), 16));

} else if (ni\_bit == 2) {

;

} else if (ni\_bit == 1) { // immediate로 바로 A레지스터에 저장

rMgr.setRegister(0, targetAddr);

}

rMgr.instField.add("LDCH"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x54 : //STCH

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if (ni\_bit == 3) {

rMgr.memory.replace(targetAddr, targetAddr+2, String.format("%02X",rMgr.getRegister(0)));//메모리의 형태가 STring으로 구현했으므로 메모리에 String.format을 이용해 스트링 형태로 저장한다.

} else if (ni\_bit == 2) {

;

} else if (ni\_bit == 1) {

;

}

rMgr.instField.add("STCH"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0x74 : //LDT

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if (ni\_bit == 3) {

rMgr.setRegister(5, Integer.parseInt(rMgr.memory.substring(targetAddr, targetAddr + 6), 16) \* 2);

} else if (ni\_bit == 2) {

;

} else if (ni\_bit == 1) { // immediate로 바로 A레지스터에 저장

rMgr.setRegister(5, targetAddr);

}

rMgr.instField.add("LDT"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0xD8 : //RD

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

rMgr.setRegister(0, rMgr.readDevice("F1"));

rMgr.instField.add("RD"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0xDC : //WD

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

rMgr.writeDevice("05", rMgr.getRegister(0));

rMgr.instField.add("WD"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

case 0xE0 : //TD

if (e\_flag == 0) // 3형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 6), 16) \* 2;

else //4형식

targetAddr = Integer.parseInt(rMgr.memory.substring(currentAddr + 3, currentAddr + 8), 16) \* 2;

targetAddr = targetAddr + temp\_sum;

rMgr.setRegister(7, targetAddr);

if (Integer.parseInt(rMgr.memory.substring(targetAddr, targetAddr + 2), 16) == 0xF1) {//입출력 지정 장치인 F1 혹은 05 라면 flag값 -1로 설정

rMgr.setRegister(9, -1);

rMgr.currentDevice = "F1"; //TD명령어 부터 CurrentDevice 를 GUI에 뿌려줌

}

if (Integer.parseInt(rMgr.memory.substring(targetAddr, targetAddr + 2), 16) == 0x05) {

rMgr.setRegister(9, -1);

rMgr.currentDevice = "05";

}

rMgr.instField.add("TD"); //LOG GUI 부분에 나타날 명령어 이름 저장

break;

}

}

}

}

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

\* 설명 : 현재 PC 값부터 프로그램이 끝날 때까지 명령어를 수행한다.

\* 매개 : -

\* 반환 : -

\* ----------------------------------------------------------------------------------

\*/

@Override

public void allStep() {

while(true){ //다시 PC값이 시작주소로 할당되면 프로그램이 종료

if(rMgr.end){

break;

}

oneStep();

}

}

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

\* 설명 : 머신을 위한 기계 코드목록 파일을 읽어 기계어 목록 테이블(inst\_table)을

\* 생성하는 함수이다.

\* 매개 : 기계어 목록 파일

\* 반환 : 정상종료 = 0 , 에러 < 0

\* ----------------------------------------------------------------------------------

\*/

int init\_inst\_file(String inst\_file)

{

int errno = 0;

int i = 0;

try

{

Scanner scanner = new Scanner(new File(inst\_file));

while(scanner.hasNext()){

StringTokenizer st = new StringTokenizer(scanner.nextLine(),"\t\n"); //스캐너 클래스를 이용해 한줄씩 받아옴 + 탭,개행으로 토크나이즈

inst\_table[i] = new inst\_unit();

inst\_table[i].operator = st.nextToken();

inst\_table[i].type = st.nextToken();

inst\_table[i].opcode = st.nextToken();

inst\_table[i].operandAmount = Integer.parseInt(st.nextToken());

inst\_index = i; //0부터 갯수 시작

i++;

}

scanner.close();

}

catch (FileNotFoundException e) { //파일이 없으면 try catch 로 예외처리

System.out.println("The inst\_file doesn't exist");

e.printStackTrace();

return errno = -1;

}

return errno;

}

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

\* 설명 : 입력 문자열이 기계어 코드인지를 검사하는 함수이다.

\* 매개 : 토큰 단위로 구분된 문자열

\* 반환 : 정상종료 = 기계어 테이블 인덱스, 에러 < 0

\* ----------------------------------------------------------------------------------

\*/

int search\_opcode(String str) //str을 비교값으로 해서 inst\_table을 모두 뒤져서 index를 찾음. 없으면 -1 리턴

{

for (int i = 0; i <= inst\_index; i++)

{

if (str.equals(inst\_table[i].opcode))

return i;

}

return -1;

}

}--------------------------------------VisualSimulaotr\_R.java---------------------------------

import java.awt.Dimension;

import java.awt.FileDialog;

import java.awt.Font;

import java.awt.Point;

import java.awt.Rectangle;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.util.ArrayList;

import java.util.Vector;

import javax.swing.JButton;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JList;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTextField;

public class VisualSimulator\_R extends JFrame implements VisualSimulator{

public JPanel jPanel = null;

public JLabel [] label = new JLabel [30];

public JTextField filename = null;

public JButton fileopen = null;

public JTextField programName = null;

public JTextField startAddress = null;

public JTextField programLength = null;

public JTextField firstInstruction = null;

public JTextField startAddress2 = null;

public JTextField targetAddress = null;

public JTextField currentDevice = null;

public JTextField ARegDec = null;

public JTextField XRegDec = null;

public JTextField LRegDec = null;

public JTextField PCRegDec = null;

public JTextField ARegHex = null;

public JTextField XRegHex = null;

public JTextField LRegHex = null;

public JTextField PCRegHex = null;

public JTextField SWReg = null;

public JTextField BRegDec = null;

public JTextField SRegDec = null;

public JTextField TRegDec = null;

public JTextField BRegHex = null;

public JTextField SRegHex = null;

public JTextField TRegHex = null;

public JTextField FReg = null;

public JScrollPane instScroll = null;

public JScrollPane logScroll = null;

public JList jList = null;

public JButton stepOneBtn = null;

public JButton stepAllBtn = null;

public JButton Exit = null;

public JList logList = null;

ResourceManager\_R rMgr = null;

SicSimulator\_R simulator=null;

public File objectFile = null;

public VisualSimulator\_R() {

initjPanel();

}

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

\* 설명 : 패널을 구성하는 각 컴포넌트들을 만들고 이를 패널에 붙여주는 메소드

\* 매개 : -

\* 반환 : -

\* ----------------------------------------------------------------------------------

\*/

public void initjPanel() {

label[0] = new JLabel();

label[0].setBounds(new Rectangle(11, 15, 65, 25));

label[0].setText("File Name");

label[1] = new JLabel();

label[1].setBounds(new Rectangle(11, 55, 135, 25));

label[1].setFont(new Font("Dialog", Font.BOLD, 14));

label[1].setText("H (Header Record)");

label[2] = new JLabel();

label[2].setBounds(new Rectangle(11, 86, 92, 25));

label[2].setText("Program Name");

label[3] = new JLabel();

label[3].setBounds(new Rectangle(11, 115, 200, 33));

label[3].setText("<HTML>Start Address of<br> Object Program</HTML>");

label[4] = new JLabel();

label[4].setText("Length of Program");

label[4].setLocation(new Point(11, 150));

label[4].setSize(new Dimension(109, 25));

label[5] = new JLabel();

label[5].setBounds(new Rectangle(237, 55, 109, 25));

label[5].setFont(new Font("Dialog", Font.BOLD, 14));

label[5].setText("E (End Record)");

label[6] = new JLabel();

label[6].setBounds(new Rectangle(237, 86, 159, 33));

label[6].setText("<HTML>Address of First Instruction<br>in Object Program</HTML>");

label[7] = new JLabel();

label[7].setBounds(new Rectangle(237, 130, 160, 25));

label[7].setText("Start Address in Memory");

label[8] = new JLabel();

label[8].setBounds(new Rectangle(237, 160, 93, 25));

label[8].setText("Target Address");

label[9] = new JLabel();

label[9].setBounds(new Rectangle(11, 180, 105, 25));

label[9].setFont(new Font("Dialog", Font.BOLD, 14));

label[9].setText("Register");

label[10] = new JLabel();

label[10].setBounds(new Rectangle(70, 205, 38, 25));

label[10].setText("Dec");

label[11] = new JLabel();

label[11].setBounds(new Rectangle(140, 205, 38, 25));

label[11].setText("Hex");

label[12] = new JLabel();

label[12].setBounds(new Rectangle(11, 235, 38, 25));

label[12].setText("A(#0)");

label[13] = new JLabel();

label[13].setBounds(new Rectangle(11, 265, 38, 25));

label[13].setText("X(#1)");

label[14] = new JLabel();

label[14].setBounds(new Rectangle(11, 295, 38, 25));

label[14].setText("L(#2)");

label[15] = new JLabel();

label[15].setBounds(new Rectangle(11, 325, 38, 25));

label[15].setText("PC(#8)");

label[16] = new JLabel();

label[16].setBounds(new Rectangle(11, 355, 46, 25));

label[16].setText("SW(#9)");

label[17] = new JLabel();

label[17].setBounds(new Rectangle(11, 388, 155, 25));

label[17].setFont(new Font("Dialog", Font.BOLD, 14));

label[17].setText("Register(for XE)");

label[18] = new JLabel();

label[18].setBounds(new Rectangle(70, 413, 38, 25));

label[18].setText("Dec");

label[19] = new JLabel();

label[19].setBounds(new Rectangle(140, 413, 38, 25));

label[19].setText("Hex");

label[20] = new JLabel();

label[20].setBounds(new Rectangle(11, 443, 38, 25));

label[20].setText("B(#3)");

label[21] = new JLabel();

label[21].setBounds(new Rectangle(11, 473, 38, 25));

label[21].setText("S(#4)");

label[22] = new JLabel();

label[22].setBounds(new Rectangle(11, 503, 38, 25));

label[22].setText("T(#5)");

label[23] = new JLabel();

label[23].setBounds(new Rectangle(11, 533, 38, 25));

label[23].setText("F(#6)");

label[24] = new JLabel();

label[24].setBounds(new Rectangle(237, 250, 121, 25));

label[24].setFont(new Font("Dialog", Font.BOLD, 14));

label[24].setText("Instructions");

label[25] = new JLabel();

label[25].setBounds(new Rectangle(11, 563, 46, 25));

label[25].setFont(new Font("Dialog", Font.BOLD, 14));

label[25].setText("LOG");

label[26] = new JLabel();

label[26].setBounds(new Rectangle(380, 230, 150, 25));

label[26].setText("Current Using Device");

filename = new JTextField();

filename.setBounds(new Rectangle(77, 12, 180, 22));

fileopen = new JButton();

fileopen.setBounds(new Rectangle(259, 15, 79, 18));

fileopen.setText("Open");

fileopen.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

final FileDialog fd = new FileDialog((JFrame)getParent(), "File open!", FileDialog.LOAD);

fd.setVisible(true);

if(fd.getDirectory()!= null)

{

rMgr = new ResourceManager\_R();

objectFile = new File(fd.getDirectory()+fd.getFile());

stepOneBtn.setEnabled(true);

stepAllBtn.setEnabled(true);

initialize(objectFile, rMgr);

}

}

});

programName = new JTextField();

programName.setBounds(new Rectangle(128, 86, 81, 22));

startAddress = new JTextField();

startAddress.setBounds(new Rectangle(128, 120, 81, 22));

programLength = new JTextField();

programLength.setBounds(new Rectangle(128, 150, 81, 22));

firstInstruction = new JTextField();

firstInstruction.setBounds(new Rectangle(401, 95, 85, 22));

startAddress2 = new JTextField();

startAddress2.setBounds(new Rectangle(401, 130, 85, 22));

targetAddress = new JTextField();

targetAddress.setBounds(new Rectangle(401, 160, 85, 22));

currentDevice = new JTextField();

currentDevice.setBounds(new Rectangle(401, 258, 85, 22));

ARegDec = new JTextField();

ARegDec.setBounds(new Rectangle(70, 235, 60, 22));

XRegDec = new JTextField();

XRegDec.setBounds(new Rectangle(70, 265, 60, 22));

LRegDec = new JTextField();

LRegDec.setBounds(new Rectangle(70, 295, 60, 22));

PCRegDec = new JTextField();

PCRegDec.setBounds(new Rectangle(70, 325, 60, 22));

ARegHex = new JTextField();

ARegHex.setBounds(new Rectangle(140, 235, 60, 22));

XRegHex = new JTextField();

XRegHex.setBounds(new Rectangle(140, 265, 60, 22));

LRegHex = new JTextField();

LRegHex.setBounds(new Rectangle(140, 295, 60, 22));

PCRegHex = new JTextField();

PCRegHex.setBounds(new Rectangle(140, 325, 60, 22));

SWReg = new JTextField();

SWReg.setBounds(new Rectangle(70, 355, 130, 22));

BRegDec = new JTextField();

BRegDec.setBounds(new Rectangle(70, 443, 60, 22));

SRegDec = new JTextField();

SRegDec.setBounds(new Rectangle(70, 473, 60, 22));

TRegDec = new JTextField();

TRegDec.setBounds(new Rectangle(70, 503, 60, 22));

BRegHex = new JTextField();

BRegHex.setBounds(new Rectangle(140, 443, 60, 22));

SRegHex = new JTextField();

SRegHex.setBounds(new Rectangle(140, 473, 60, 22));

TRegHex = new JTextField();

TRegHex.setBounds(new Rectangle(140, 503, 60, 22));

FReg = new JTextField();

FReg.setBounds(new Rectangle(70, 533, 130, 22));

jList = new JList();

logList = new JList();

instScroll = new JScrollPane();

instScroll.setBounds(new Rectangle(237, 288, 124, 260));

instScroll.setViewportView(jList);

stepOneBtn = new JButton();

stepOneBtn.setBounds(new Rectangle(375, 370, 131, 23));

stepOneBtn.setEnabled(false);

stepOneBtn.setText("execute(1 step)");

stepOneBtn.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

oneStep();

}

});

stepAllBtn = new JButton();

stepAllBtn.setBounds(new Rectangle(375, 410, 131, 23));

stepAllBtn.setEnabled(false);

stepAllBtn.setText("execute(All)");

stepAllBtn.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

allStep();

}

});

Exit = new JButton();

Exit.setBounds(new Rectangle(375, 462, 131, 23));

Exit.setText("Exit");

Exit.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

// TODO Auto-generated method stub

System.exit(0);

}

});

logScroll = new JScrollPane();

logScroll.setBounds(new Rectangle(11, 590, 501, 170));

logScroll.setViewportView(logList);

//////////////////////////////////////////////////////// 패널에 각 컴포넌트들을 붙이는 부분

jPanel = new JPanel();

jPanel.setLayout(null);

for(int i =0;i<27;i++){

jPanel.add(label[i], null);

}

jPanel.add(filename,null);

jPanel.add(fileopen,null);

jPanel.add(programName,null);

jPanel.add(startAddress,null);

jPanel.add(programLength,null);

jPanel.add(firstInstruction,null);

jPanel.add(startAddress2,null);

jPanel.add(targetAddress,null);

jPanel.add(currentDevice,null);

jPanel.add(ARegDec,null);

jPanel.add(XRegDec,null);

jPanel.add(LRegDec,null);

jPanel.add(PCRegDec,null);

jPanel.add(ARegHex,null);

jPanel.add(XRegHex,null);

jPanel.add(LRegHex,null);

jPanel.add(PCRegHex,null);

jPanel.add(SWReg,null);

jPanel.add(BRegDec,null);

jPanel.add(SRegDec,null);

jPanel.add(TRegDec,null);

jPanel.add(BRegHex,null);

jPanel.add(SRegHex,null);

jPanel.add(TRegHex,null);

jPanel.add(FReg,null);

jPanel.add(instScroll, null);

jPanel.add(logScroll, null);

jPanel.add(stepOneBtn, null);

jPanel.add(stepAllBtn, null);

jPanel.add(Exit, null);

this.setSize(530, 800);

this.setContentPane(jPanel);

this.setTitle("SIC/XE Simulator");

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

}

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

\* 설명 : fileDialog를 통해 파일이 오픈되면 호출되며, SicSimulator의 인스턴스를 생성하고, update()함수를 통해서 GUI에 초기 정보를 업데이트한다.

\* 매개 : -

\* 반환 : -

\* ----------------------------------------------------------------------------------

\*/

@Override

public void initialize(File objFile, ResourceManager\_R rMgr) {

simulator = new SicSimulator\_R();

simulator.initialize(objFile, rMgr);

filename.setText(objFile.getName());

firstInstruction.setText(rMgr.firstInst);

startAddress2.setText(rMgr.startAddr\_string[0]);

update();

}

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

\* 설명 : oneStep 버튼에 매핑시켜놓은 메소드. SicSimulator의 oneStep() 메소드를 호출하며 호출 후 업데이트 된 정보를

\* GUI에 나타내준다

\* 매개 : -

\* 반환 : -

\* ----------------------------------------------------------------------------------

\*/

@Override

public void oneStep() {

simulator.oneStep();

update();

if(rMgr.end)

new JOptionPane().showMessageDialog(null, "Object code running is finished");

}

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

\* 설명 : allStep 버튼에 매핑시켜놓은 메소드. SicSimulator의 allStep() 메소드를 호출하며 호출 후 업데이트 된 정보를

\* GUI에 나타내준다

\* 매개 : -

\* 반환 : -

\* ----------------------------------------------------------------------------------

\*/

@Override

public void allStep() {

simulator.allStep();

update();

if(rMgr.end)

new JOptionPane().showMessageDialog(null, "Object code running is finished");

}

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

\* 설명 : SicSimulator에서 명령어를 수행하고 난 뒤, GUI에 수정된 정보를 뿌려주는 메소드.

\* 매개 : -

\* 반환 : -

\* ----------------------------------------------------------------------------------

\*/

@Override

public void update() {

// TODO Auto-generated method stub

programName.setText(rMgr.progName[rMgr.sectionIndex]); //섹션 변화에 따른 프로그램 이름 업데이트

startAddress.setText(rMgr.startAddr\_string[rMgr.sectionIndex]); //섹션 변화에 따른 프로그램 주소

// programLength.setText(String.format("%06X", (rMgr.progLength[0]+rMgr.progLength[1]+rMgr.progLength[2]) / 2));

programLength.setText(String.format("%06X", rMgr.progLength[rMgr.sectionIndex]/ 2)); //섹션 변화에 따른 프로그램 길이

// 명령어 수행에 따른 각 레지스터 값 변화

ARegDec.setText(String.format("%d", rMgr.A));

XRegDec.setText(String.format("%d", rMgr.X/2));

LRegDec.setText(String.format("%d", rMgr.L/2));

PCRegDec.setText(String.format("%d", rMgr.PC/2));

ARegHex.setText(String.format("%06X", rMgr.A));

XRegHex.setText(String.format("%06X", rMgr.X/2));

LRegHex.setText(String.format("%06X", rMgr.L/2));

PCRegHex.setText(String.format("%06X", rMgr.PC/2));

SWReg.setText(String.format("%06d", rMgr.SW));

BRegDec.setText(String.format("%d", rMgr.B/2));

SRegDec.setText(String.format("%d", rMgr.S));

TRegDec.setText(String.format("%d", rMgr.T/2));

BRegHex.setText(String.format("%06X", rMgr.B/2));

SRegHex.setText(String.format("%06X", rMgr.S));

TRegHex.setText(String.format("%06X", rMgr.T/2));

FReg.setText(String.format("%06d", rMgr.F));

targetAddress.setText(String.format("%06X", rMgr.TA/2));

currentDevice.setText(rMgr.currentDevice); //Device에 접근할 때 변하는 CurrentUsingDevice를 업데이트해줌

jList.setListData(rMgr.memoryField); //명령어 수행에 따른 objectcode를 업데이트

logList.setListData(rMgr.instField); //명령어 수행에 따른 명령어 이름을 업데이트

}

}

주요 소스코드 부분 설명

**else** **if** (inst\_table[index].type.equals("3/4"))

{

**int** targetAddr=0;

**int** x\_flag=0,b\_flag=0,p\_flag=0,e\_flag=0;

**int** temp\_sum = 0;

**int** sign=0;

// target 주소를 구하기 위해 x, b, p, e flag를 설정.

x\_flag = (xbpe\_bit & 8) >> 3;

b\_flag = (xbpe\_bit & 4) >> 2;

p\_flag = (xbpe\_bit & 2) >> 1;

e\_flag = xbpe\_bit & 1;

**if**( e\_flag == 0){

rMgr.setRegister(8, rMgr.getRegister(8)+ 3 \* 2); //다음 PC값 증가 모든 PC값들도 \*2를 해서 증가시켜줌

rMgr.memoryField.add(rMgr.memory.substring(currentAddr, currentAddr+6)); //insturction GUI 부분에 나타낼 정보 저장

}

**else** **if**(e\_flag == 1){

rMgr.setRegister(8, rMgr.getRegister(8)+ 4 \* 2); //다음 PC값 증가

rMgr.memoryField.add(rMgr.memory.substring(currentAddr, currentAddr+8)); //insturction GUI 부분에 나타낼 정보 저장

}

//Target address 설정

**if** (x\_flag == 1)

temp\_sum += rMgr.getRegister(1); //X를 증가시킬 때, 2를 곱해서 증가시켜놨음에 유의

**if** (e\_flag == 1) {

;

} **else** {

**if** (b\_flag == 1)

temp\_sum += rMgr.getRegister(3);

**if** (p\_flag == 1)

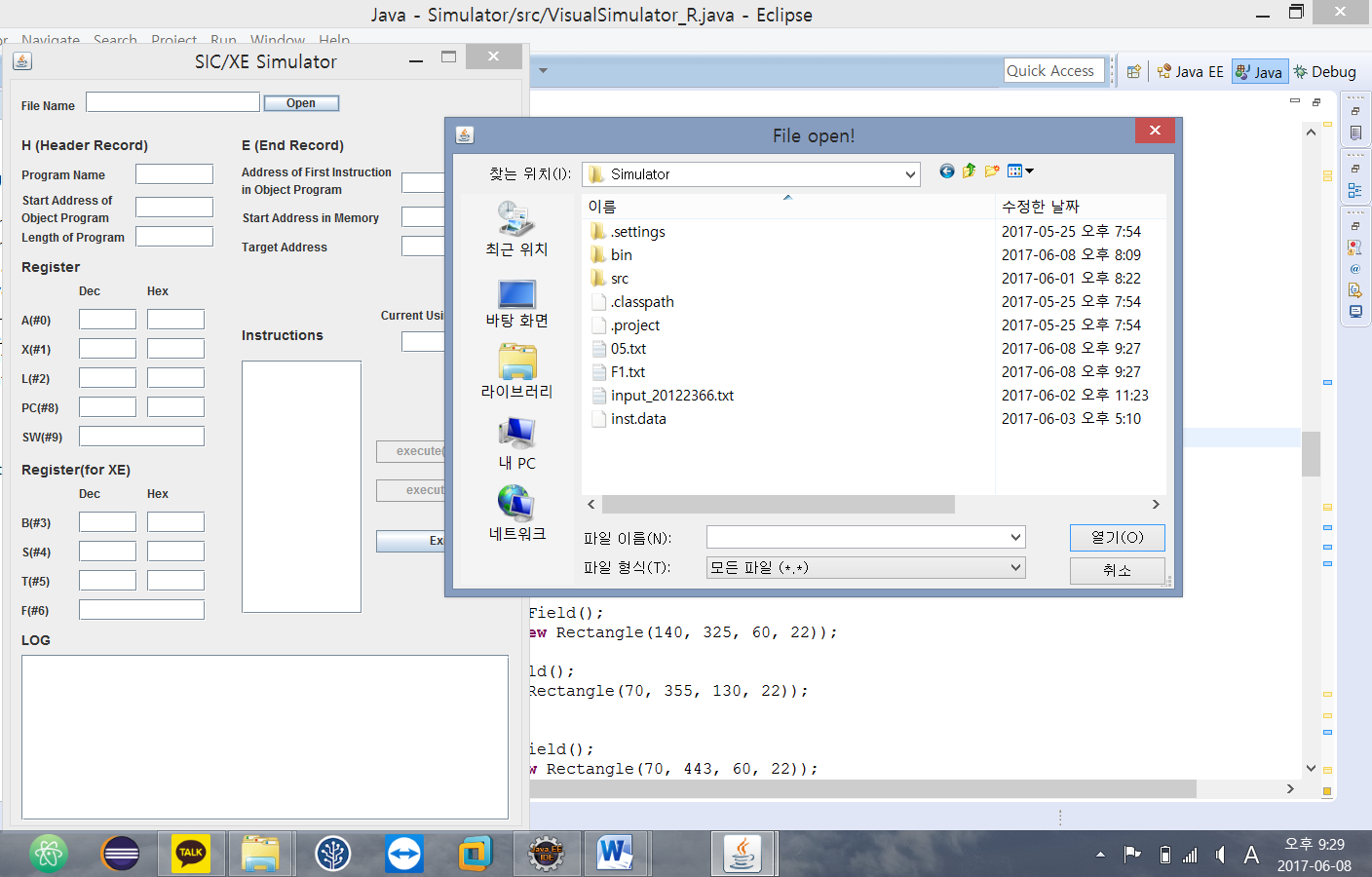
temp\_sum += rMgr.getRegister(8);

}

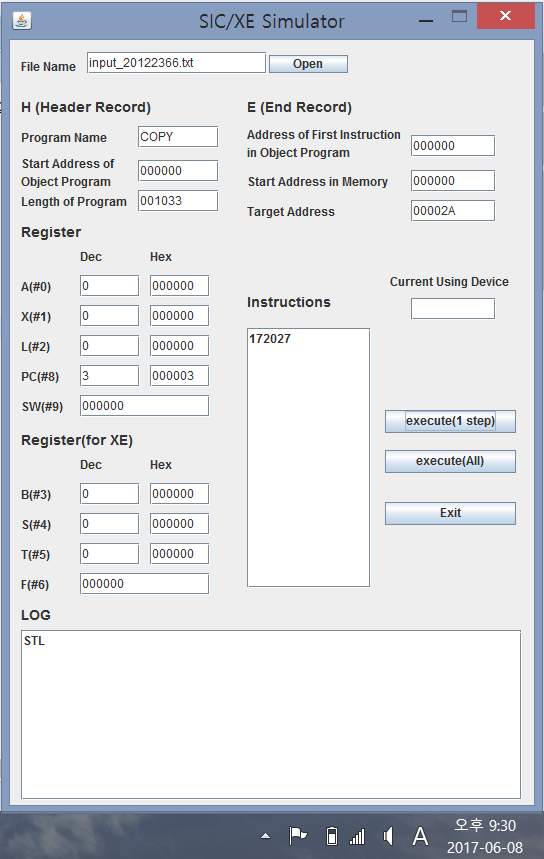
* SicSimulator\_R 클래스 내의 oneStep() 메소드

3/4 형식일 때, xbpe 비트를 각각 구한다. 그 후, 각 로직에 맞게 x , b , p 레지스터의 값의 합을 임시 변수인 temp\_sum에 저장한다. 이 temp\_sum은 3형식이라면 object code의 하위 3바이트와 더하여 TargetAddr에 저장하고, 4형식이라면 하위 5바이트 더하여 저장한다. 이렇게 구해진 TargetAddr을 이용해 명령어를 수행한다.

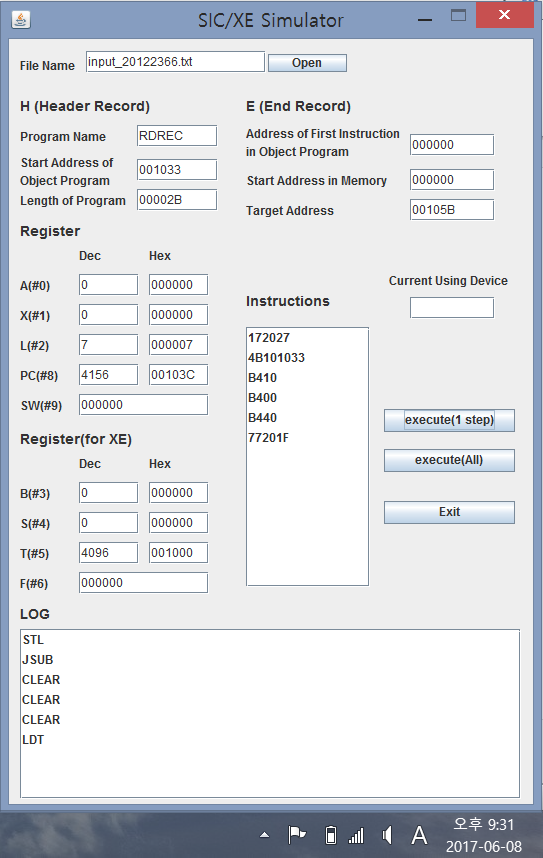
1. 수행 결과



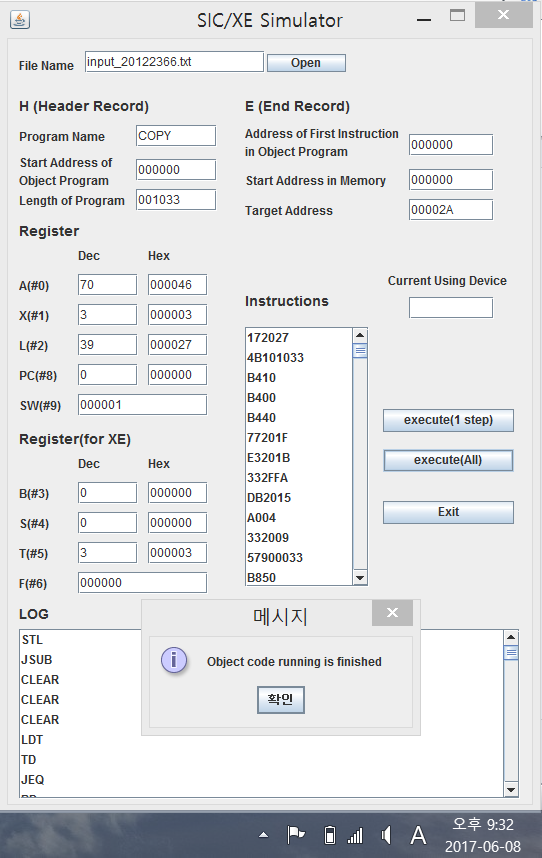
* 파일 다이얼로그를 통해 인풋파일을 받아오는 모습



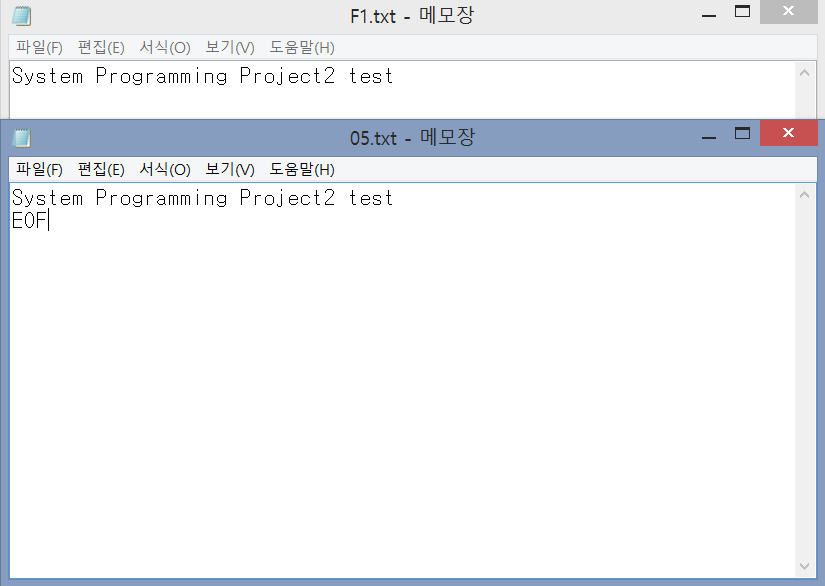
* 인풋파일 로딩 후, 1step 버튼을 한번 클릭함



* 1step 버튼을 여러 번 클릭함. 섹션이동으로 인해, Program Name이 RDREC로 바뀌며 각종 정보가 바뀜



* Allstep 버튼을 클릭한 후, 완료되었다는 메시지 창 나타남



* 잘 복사가 되었나 실제 파일을 보고 확인

1. 결론 및 보충할 점

프로젝트를 진행하면서 로더의 동작과정에 대해 심도있게 파악하게 되었다. 비록 직접 시스템 속에 실제로 돌아가는 로더를 짠 것은 아니지만 어떻게 메모리에 넣고 어떻게 레지스터를 활용하는 지에 대하여 알게 된 것은 큰 소득이라 할 수 있다. 이번 과제는 GUI로 시뮬레이터까지 만들어야 했다는 점에서 번거로운 작업이 많이 있었다. 하지만 이렇게라도 GUI를 접해보고, 유저들의 편의를 위해 가시적으로 프로그래밍 했다는 점에서는 배울 점이 있었다. 하지만 과제 제출을 위해 빠르게 구현하느라 UI/UX에 큰 비중을 두지 않았다는 점이 아쉽게 느껴진다. 그리고 로더에서도 주어진 인풋 파일로만 돌아가게 끔 만들었다는 것도 아쉬운 점이다. 이는 추후 기회가 닿는다면 다시 스스로 구현을 함으로써 프로그램의 퀄리티를 높여보고 싶다.