|  |
| --- |
|  |

|  |
| --- |
| **시스템 프로그래밍 Project #2** |
| **SIC/XE Simulator 구현** |
|  |
|  |
|  |
| **교수 : 최재영 교수님** |
| **가반 114번**  **컴퓨터학부 20132361 박진수** |
|  |

**목 차**

1. **동기/목적**
2. **설계/구현 아이디어**
3. **수행결과**
4. **결론 및 보충할 점**
5. **소스코드**
6. **동기/목적**

**프로젝트 내용**

**Control Section 방식으로 생성된 Object Program(프로젝트1,1B의 결과물)을 실행하고 시뮬**

**레이션 할 수 있는 시뮬레이터를 Visual하게 보여주는 JAVA GUI 프로그램.**

**목적**

**Control Section방식의 어셈블러를 C언어로 구현한 프로젝트 1과 JAVA로 구현한 프로젝트1B**

**를 통해서 어셈블러의 방식을 이해하고 어셈블러를 통한 결과물인 Object Program을 이용하**

**여 가상으로 메모리에 로더하여 실행하고 시뮬레이션을 할 수 있는 시뮬레이터를 만들어**

**실제 코드가 동작하는 방식을 이해하고 자바와 GUI에 대한 이해를 높이고 실력을 향상시킨다.**

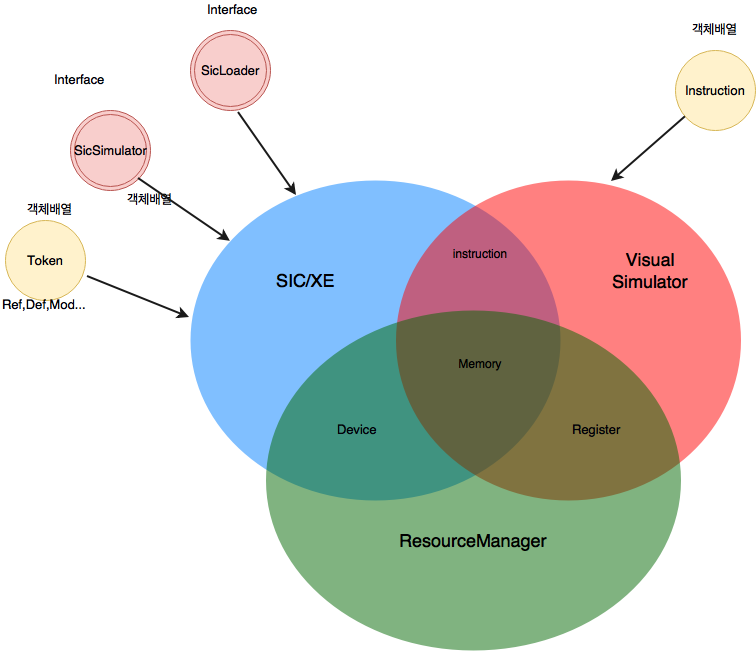
**동기**

**어셈블리한 프로그램을 Linking, Relocation, Loading의 기능이 있는 Linking Loader를 가상을**

**메모리를 통해 자바의 인터페이스인 GUI를 통해 가상 로더를 구현해본다.**

**.**

1. **설계/구현 아이디어**

 클래스 구조

**VisualSimulator(Main Frame)**

**시뮬레이터의 동작방식을 GUI방식으로 구현하였다. JTextField, JList, JButton 등을 통하여 시뮬**

**레이터의 GUI환경을 만들고 버튼을 누를 때 마다 그에 따라 textfield나 list의 값 등이 갱신을**

**하게 하였다. 실제 로더기능은 SICXE클래스를 동작시켜 데이터 값을 리소스 매니저의 값을 수**

**정하여 데이터를 받아와 갱신하였다. 메인함수에는 프레임을 동작시키고 명령어 테이블을 만들**

**었다.**

**Resource Manager**

**각 레지스터 넘버에 따른 값과 메모리, 디바이스가 저장되어있는 클래스이다. SICXE 클래스에서**

**Object Program을 통하여 메모리와 디바이스, 레지스터의 값의 변경이나 대입을 요청하여 값을**

**그에 따라 변경해준다. 메모리나 디바이스는 byte배열로 로더할 때 String 문자열을 2자로 쪼개어**

**1바이트롤 Parsing하여 메모리나 디바이스에 저장하였다. 디바이스의 입출력은 제대로된 값을**

**읽어오거나 저장하는 것을 구현하지 못하였다. 레지스터 같은 경우에는 각 명령어의 개수에 맞는**

**배열에 저장하여 명령어를 실행할 때 마다 값을 갱신해주었다.**

**SIC/XE(implements SicSimulator, SicLoader)**

**실질적으로 모든 기능을 수행해주는 클래스로 Object Code를 리소스 매니저에 로드시키고 로드**

**된 코드를 실행하는 클래스이다. Object Code를 레코드에 따라 나누어 읽어 들여 명령어, 주소,**

**정의된 값, 참조하는 값, 수정된 값과 주소, 등을 Arraylist나 token(임의로 만든 객체)객체배열**

**에 저장을 하거나 읽어들인 값을 절대주소를 통하여 리소스매니저의 메모리에 적재하고 레지스터**

**에 값을 넣는다. 형식이나 명령어에 따라 레지스터값의 저장이나 메모리의 적재를 나눠서 하였다.**

**-명령어 추출**

**생각보다 잘 안되었던 명령어는 OP코드와 ni값을 합산한 값이 BYTE이거나 WORD등의 지시어**

**의 아스키코드나 16진수가 일치하는 경우가 있어서 다음 값의 비교나 xbpe값의 유무에 따라**

**명령어를 추출하였다.**

**-레지스터**

**레지스터 같은 경우에는 레지스터 넘버에 따라, 저장/적재 명령어에 따라, 특별한 명령어에 따라**

**나누어 리소스매니저의 배열에 그에 맞는 값을 저장하였다.**

**-메모리(PARSING포함)**

**메모리의 경우 op코드에 의해 불러온 명령어 형식이나 xbpe값 등을 비교하여 기준에 따라 나누어**

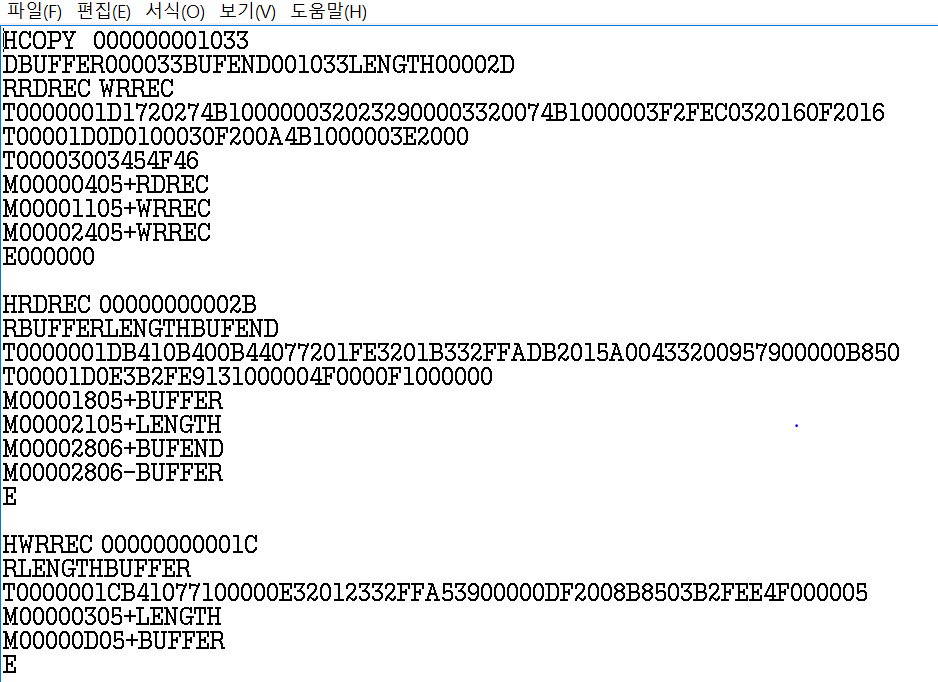
**리소스 매니저의 메모리 바이트배열에 저장하였다. String이나 int를 byte로 파싱하여 저장해야**

**하므로 따로 static메소드를 만들어 스트링의 길이에 따라 2개씩 분할하여 int형으로 다시 바꾼후**

**바이트 배열에 저장하였다.**

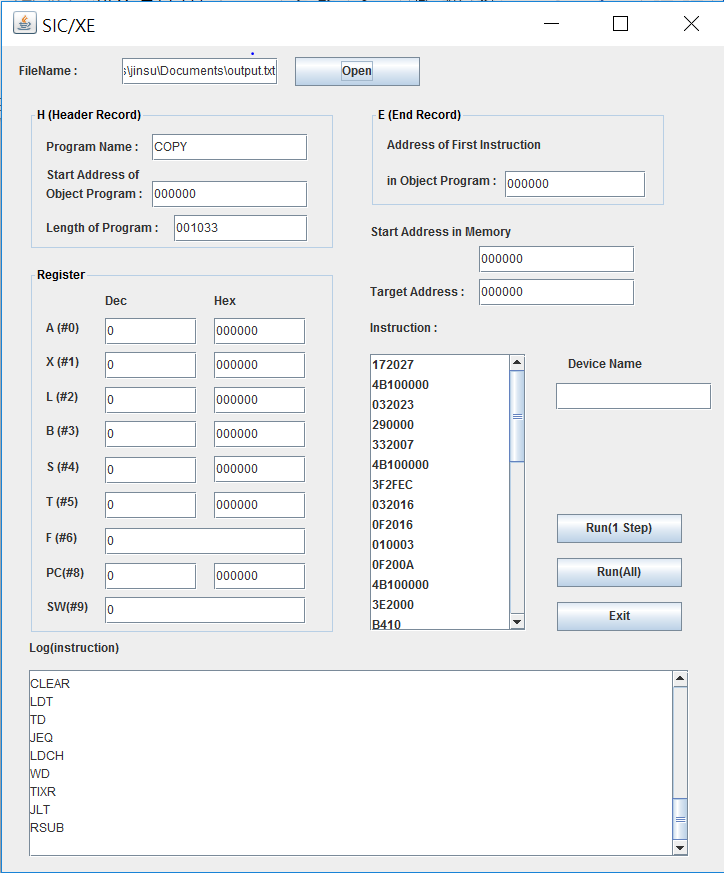
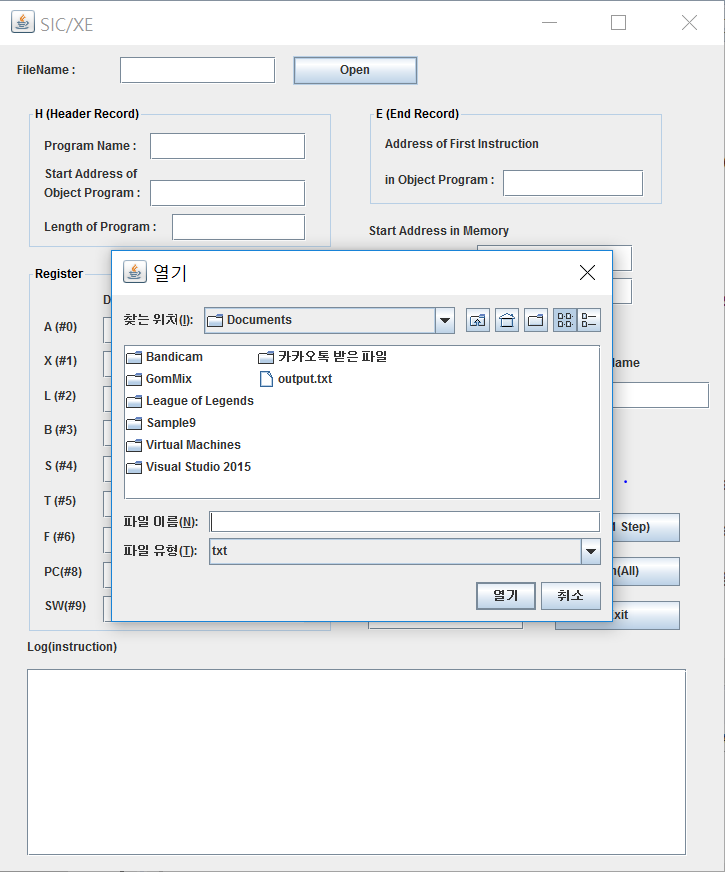
**3. 수행결과**

**input**

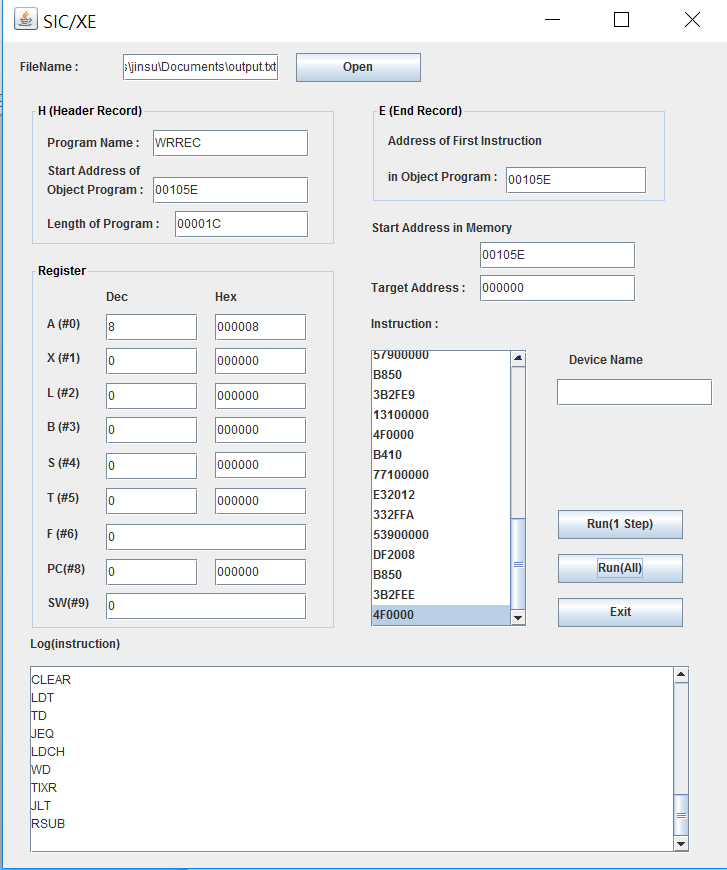
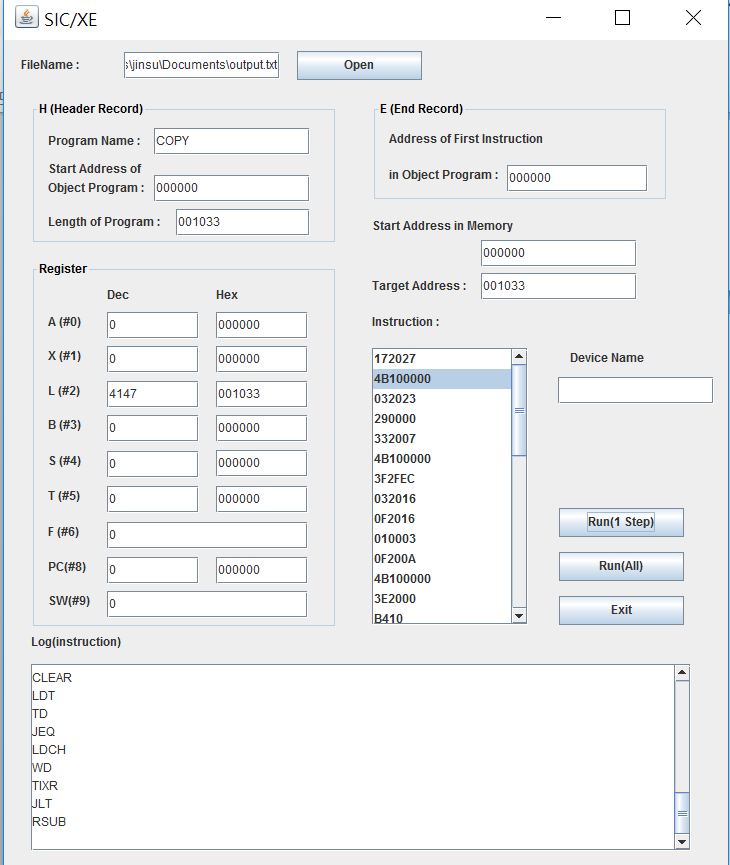


**프로젝트1과 1b의 결과인 control section방식의 object program을 input으로 넣었다.**

**시뮬레이터 결과(GUI)**



**파일 오픈(open)버튼 누를 시 첫 화면**

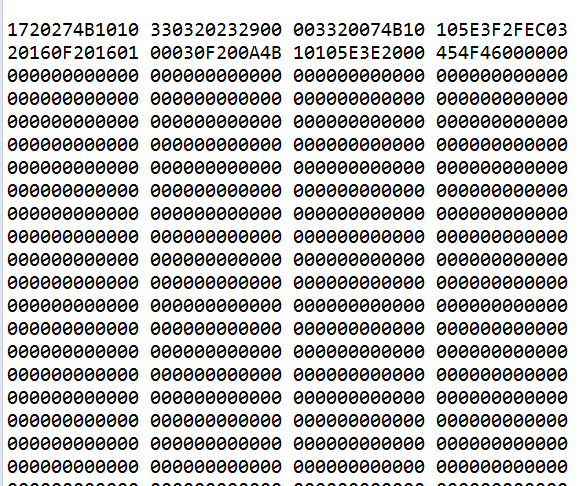
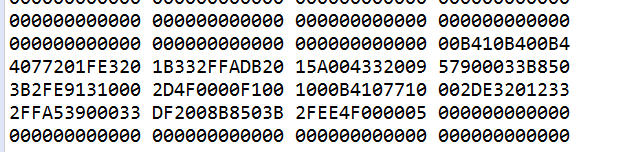


**Run(1 step)을 눌렀을 시 Run(All)을 눌렀을 시**

**수행결과 명령어를 읽어 들이고 그에 맞는 레지스터, 메모리의 주소, 타겟주소,**

**H레코드 E레코드 등의 값을 textfield에 값을 갱신하였음.**

**Memory 결과**

** **

**Object Program에서 Modification을 수행한 메모리를 출력하였음**

**4. 결론 및 보충할 점**

**Control Section방식의 어셈블러인 프로젝트의 1과 1B의 결과물인 Object Program을 로더하여 실행**

**하는 시뮬레이터를 자바의 GUI환경을 통해서 Visual하게 구현하였다. 가상 메모리에 modification을**

**통하여 주소값을 수정하고 Control Section으로 나누어진 프로그램을 링킹하여 메모리에 적재하고,**

**각 명령어에 맞는 레지스터 값, 메모리주소 값 등을 시뮬레이터를 통해 보여주었다. 디바이스 같은**

**경우에는 명령어에 맞는 디바이스를 GUI에 표현을 해주었지만 실제로 파일의 입출력은 제대로 실행**

**하는 것을 구현하지 못하였다. 단지 가상의 디바이스에 저장하고 로더할 수 있는 기능만 구현시켜**

**놓았다.**

**아직 익숙하지 못한 자바의 GUI환경과 가상의 메모리를 통하여 수업에서 배운 로더 및 링커의 기능을**

**직접적을 구현시켜보아 로더의 기능을 좀 더 이해하고 로딩, 링킹, 재배치 등의 기능에 대해 좀 더 세**

**부적이고 필요한 기법과 기능들을 알게 되었고, 저번 프로젝트1b와 함께 자바에 대해 익숙해지고 실력**

**을 한층 더 향상하게 되었다. 그리고 각 명령어의 Object Code를 바이트 단위로 분할하여 사용하고**

**그에 따른 레지스터의 사용을 더 구조적으로 이해하게 되었다.**

**완벽하게 구현을 하지는 못했지만 수업시간에 글로만 배운 것과 달리 직접 로더를 구현해봄으로써**

**로더의 기능인 링킹,로딩,재배치 등이 과정과 구조에 대해 훨씬 이해를 높이게 되었고 자바의 인터**

**페이스, 객체, ArrayList 등과 자바의 byte자료형 등 이전에 사용하지 않은 메소드나 키워드를 사용**

**함에 있어 자바의 활용도를 더 알게 되고 익숙함과 실력을 향상시키는 계기가 되었다.**

**5. 소스코드**

**VisualSimulator.class**

**package** project;

**import** java.awt.BorderLayout;

**import** java.awt.EventQueue;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.util.Vector;

**import** javax.swing.JFrame;

**import** javax.swing.JPanel;

**import** javax.swing.JScrollPane;

**import** javax.swing.border.EmptyBorder;

**import** javax.swing.JLabel;

**import** javax.swing.JList;

**import** javax.swing.JTextField;

**import** javax.swing.ListSelectionModel;

**import** javax.swing.SingleSelectionModel;

**import** javax.swing.JButton;

**import** javax.swing.JFileChooser;

**import** javax.swing.border.TitledBorder;

**import** javax.swing.filechooser.FileNameExtensionFilter;

**import** java.awt.Color;

**import** javax.swing.JTextArea;

//시뮬레이터를 동작시키기 위한 세팅을 수행한다.

//SICXE를 통해 로더를 수행시키고, 로드된 값들을 읽어 보여준다

**public** **class** VisualSimulator **extends** JFrame **implements** ActionListener{

**static** instruction[] *inst\_table* = **new** instruction[256];

**static** **int** *sub\_cnt*;

**public** JPanel contentPane;

**public** JTextField text\_file; //file 텍스트필드

**public** JTextField pro\_name; //프로그램 이름 텍스트필드

**public** JTextField fir\_inst\_addr; //E레코드의 시작주소

**public** JTextField start\_addr\_obj; //프로그램의 시작주소

**public** JTextField pro\_len; //프로그램의 길이

**public** JTextField a\_dec; //A레지스터의 1

**public** JTextField a\_hex; //

**public** JTextField x\_dec; //

**public** JTextField x\_hex; //

**public** JTextField l\_dec; //

**public** JTextField l\_hex; //

**public** JTextField b\_dec; // 각 레지스터들의 텍스트 필드

**public** JTextField b\_hex; //

**public** JTextField start\_addr\_m; // 메모리의 시작주소

**public** JTextField target;

**public** JTextField s\_dec; //

**public** JTextField s\_hex; //

**public** JTextField t\_dec; // 각 레지스터들의 텍스트 필드

**public** JTextField t\_hex; //

**public** JTextField f\_dec; //

**public** JTextField pc\_dec; //

**public** JTextField pc\_hex; //

**public** JTextField sw\_dec; //

**public** JLabel label\_2;

**public** JLabel label\_3;

**public** JLabel label\_4;

**public** JLabel lblinst;

**public** JScrollPane scroll;

**public** JTextField text\_ing; //사용중인 장치

**public** JTextArea log; //log

**public** JButton btn\_open; //열기 버튼

**public** JButton btn\_step; //실행(step)버튼

**private** JButton run\_btn; //실행(all)버튼

**public** JFileChooser jfc = **new** JFileChooser();

**public** **static** **int** *step*=0;

**private** JList list; //명령어 리스트

**private** Vector listData; //명령어 리스트에 넣을 vector

**private** JButton exit\_btn; //종료 버튼

**static** BufferedReader *bw*=**null**;

**static** String *filename* = **null**;

**static** **int** *inst\_num*=0;

**static** ResourceManager *rmgr* = **new** ResourceManager();

**static** SICXE *sic*=**new** SICXE();

/\*\*

\* Launch the application.

\*/

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

VisualSimulator frame = **new** VisualSimulator();

frame.setTitle("SIC/XE");

frame.setVisible(**true**);

**for**(**int** i=0; i<256; i++)

{

*inst\_table*[i]=**new** instruction(**null**,0,0);

}

//

//명령어를 inst.data에서 불러와 inst테이블에 저장

//

**try** {

*bw* = **new** BufferedReader(**new** FileReader("inst.data"));

**for**(**int** i=0; i<256; i++)

{

String li;

li=*bw*.readLine();

**if**(li==**null**)

{

*inst\_num*=i;

**break**;

}

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

*inst\_table*[i].setName(line[0]);

*inst\_table*[i].opcode=Integer.*parseInt*(line[1],16);

*inst\_table*[i].format=Integer.*parseInt*(line[2]);

}

*bw*.close();

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

/\* EventQueue.invokeLater(new Runnable() {

public void run() {

try {

VisualSimulator frame = new VisualSimulator();

frame.setTitle("SIC/XE");

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});\*/

}

/\*\*

\* Create the frame.

\*/

**public** VisualSimulator() {

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

setBounds(100, 100, 746, 882);

contentPane = **new** JPanel();

contentPane.setForeground(Color.***PINK***);

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

setContentPane(contentPane);

contentPane.setLayout(**null**);

JLabel label\_file = **new** JLabel("FileName :");

label\_file.setBounds(17, 15, 93, 21);

contentPane.add(label\_file);

text\_file = **new** JTextField();

text\_file.setBounds(120, 12, 156, 27);

contentPane.add(text\_file);

text\_file.setColumns(10);

btn\_open = **new** JButton("Open");

btn\_open.setBounds(293, 11, 125, 29);

btn\_open.addActionListener(**this**);

contentPane.add(btn\_open);

JPanel panel\_h = **new** JPanel();

panel\_h.setForeground(Color.***PINK***);

panel\_h.setBorder(**new** TitledBorder(**null**, "H (Header Record)", TitledBorder.***LEADING***, TitledBorder.***TOP***, **null**, Color.***BLACK***));

panel\_h.setBounds(27, 60, 306, 144);

contentPane.add(panel\_h);

panel\_h.setLayout(**null**);

JLabel label\_name = **new** JLabel("Program Name :");

label\_name.setBounds(17, 31, 134, 21);

panel\_h.add(label\_name);

pro\_name = **new** JTextField();

pro\_name.setBounds(123, 28, 156, 27);

panel\_h.add(pro\_name);

pro\_name.setColumns(10);

JLabel label\_start1 = **new** JLabel("Start Address of");

label\_start1.setBounds(17, 59, 144, 21);

panel\_h.add(label\_start1);

JLabel label\_start2 = **new** JLabel("Object Program :");

label\_start2.setBounds(17, 78, 144, 21);

panel\_h.add(label\_start2);

start\_addr\_obj = **new** JTextField();

start\_addr\_obj.setBounds(123, 75, 156, 27);

panel\_h.add(start\_addr\_obj);

start\_addr\_obj.setColumns(10);

JLabel label\_length = **new** JLabel("Length of Program :");

label\_length.setBounds(17, 112, 170, 21);

panel\_h.add(label\_length);

pro\_len = **new** JTextField();

pro\_len.setBounds(145, 109, 134, 27);

panel\_h.add(pro\_len);

pro\_len.setColumns(10);

JPanel panel\_e = **new** JPanel();

panel\_e.setBorder(**new** TitledBorder(**null**, "E (End Record)", TitledBorder.***LEADING***, TitledBorder.***TOP***, **null**, Color.***BLACK***));

panel\_e.setBounds(368, 60, 296, 101);

contentPane.add(panel\_e);

panel\_e.setLayout(**null**);

JLabel label\_end1 = **new** JLabel("Address of First Instruction");

label\_end1.setBounds(17, 29, 231, 21);

panel\_e.add(label\_end1);

JLabel label\_end2 = **new** JLabel("in Object Program :");

label\_end2.setBounds(17, 65, 169, 21);

panel\_e.add(label\_end2);

fir\_inst\_addr = **new** JTextField();

fir\_inst\_addr.setBounds(135, 65, 141, 27);

panel\_e.add(fir\_inst\_addr);

fir\_inst\_addr.setColumns(10);

JPanel panel\_r = **new** JPanel();

panel\_r.setBorder(**new** TitledBorder(**null**, "Register", TitledBorder.***LEADING***, TitledBorder.***TOP***, **null**, Color.***BLACK***));

panel\_r.setBounds(27, 220, 306, 368);

contentPane.add(panel\_r);

panel\_r.setLayout(**null**);

JLabel lblA = **new** JLabel("A (#0)");

lblA.setBounds(17, 52, 57, 21);

panel\_r.add(lblA);

JLabel lblDec = **new** JLabel("Dec");

lblDec.setBounds(76, 25, 78, 21);

panel\_r.add(lblDec);

a\_dec = **new** JTextField();

a\_dec.setBounds(76, 52, 92, 27);

panel\_r.add(a\_dec);

a\_dec.setColumns(10);

a\_hex = **new** JTextField();

a\_hex.setColumns(10);

a\_hex.setBounds(185, 52, 92, 27);

panel\_r.add(a\_hex);

x\_dec = **new** JTextField();

x\_dec.setColumns(10);

x\_dec.setBounds(76, 86, 92, 27);

panel\_r.add(x\_dec);

x\_hex = **new** JTextField();

x\_hex.setColumns(10);

x\_hex.setBounds(185, 86, 92, 27);

panel\_r.add(x\_hex);

l\_dec = **new** JTextField();

l\_dec.setColumns(10);

l\_dec.setBounds(76, 121, 92, 27);

panel\_r.add(l\_dec);

l\_hex = **new** JTextField();

l\_hex.setColumns(10);

l\_hex.setBounds(185, 121, 92, 27);

panel\_r.add(l\_hex);

b\_dec = **new** JTextField();

b\_dec.setColumns(10);

b\_dec.setBounds(76, 155, 92, 27);

panel\_r.add(b\_dec);

b\_hex = **new** JTextField();

b\_hex.setColumns(10);

b\_hex.setBounds(185, 155, 92, 27);

panel\_r.add(b\_hex);

JLabel lblHex = **new** JLabel("Hex");

lblHex.setBounds(185, 25, 78, 21);

panel\_r.add(lblHex);

JLabel lblX = **new** JLabel("X (#1)");

lblX.setBounds(17, 86, 57, 21);

panel\_r.add(lblX);

JLabel lblL = **new** JLabel("L (#2)");

lblL.setBounds(17, 121, 57, 21);

panel\_r.add(lblL);

JLabel lblSw = **new** JLabel("SW(#9)");

lblSw.setBounds(17, 331, 57, 21);

panel\_r.add(lblSw);

JLabel label = **new** JLabel("B (#3)");

label.setBounds(17, 155, 57, 21);

panel\_r.add(label);

JLabel label\_1 = **new** JLabel("S (#4)");

label\_1.setBounds(17, 191, 57, 21);

panel\_r.add(label\_1);

s\_dec = **new** JTextField();

s\_dec.setColumns(10);

s\_dec.setBounds(76, 191, 92, 27);

panel\_r.add(s\_dec);

s\_hex = **new** JTextField();

s\_hex.setColumns(10);

s\_hex.setBounds(185, 190, 92, 27);

panel\_r.add(s\_hex);

t\_dec = **new** JTextField();

t\_dec.setColumns(10);

t\_dec.setBounds(76, 226, 92, 27);

panel\_r.add(t\_dec);

t\_hex = **new** JTextField();

t\_hex.setColumns(10);

t\_hex.setBounds(185, 226, 92, 27);

panel\_r.add(t\_hex);

f\_dec = **new** JTextField();

f\_dec.setColumns(10);

f\_dec.setBounds(76, 262, 201, 27);

panel\_r.add(f\_dec);

pc\_dec = **new** JTextField();

pc\_dec.setColumns(10);

pc\_dec.setBounds(76, 297, 92, 27);

panel\_r.add(pc\_dec);

pc\_hex = **new** JTextField();

pc\_hex.setColumns(10);

pc\_hex.setBounds(185, 297, 92, 27);

panel\_r.add(pc\_hex);

sw\_dec = **new** JTextField();

sw\_dec.setColumns(10);

sw\_dec.setBounds(76, 331, 201, 27);

panel\_r.add(sw\_dec);

label\_2 = **new** JLabel("T (#5)");

label\_2.setBounds(17, 226, 57, 21);

panel\_r.add(label\_2);

label\_3 = **new** JLabel("F (#6)");

label\_3.setBounds(17, 262, 57, 21);

panel\_r.add(label\_3);

label\_4 = **new** JLabel("PC(#8)");

label\_4.setBounds(17, 297, 57, 21);

panel\_r.add(label\_4);

JLabel lbl\_start\_addr = **new** JLabel("Start Address in Memory");

lbl\_start\_addr.setBounds(368, 176, 213, 21);

contentPane.add(lbl\_start\_addr);

start\_addr\_m = **new** JTextField();

start\_addr\_m.setBounds(477, 200, 156, 27);

contentPane.add(start\_addr\_m);

start\_addr\_m.setColumns(10);

JLabel lblNewLabel\_1 = **new** JLabel("Target Address :");

lblNewLabel\_1.setBounds(368, 236, 137, 21);

contentPane.add(lblNewLabel\_1);

target = **new** JTextField();

target.setBounds(477, 233, 156, 27);

contentPane.add(target);

target.setColumns(10);

lblinst = **new** JLabel("Instruction :");

lblinst.setBounds(368, 272, 102, 21);

contentPane.add(lblinst);

listData = **new** Vector();

list = **new** JList(listData);

list.setSelectionMode(ListSelectionModel.***SINGLE\_SELECTION***);

//inst\_txt.setBounds(368, 308, 156, 277);

//contentPane.add(inst\_txt);

scroll = **new** JScrollPane(list);

scroll.setBounds(368, 308, 156, 277);

contentPane.add(scroll);

JLabel lblNewLabel = **new** JLabel("Device Name");

lblNewLabel.setBounds(566, 308, 114, 21);

contentPane.add(lblNewLabel);

text\_ing = **new** JTextField();

text\_ing.setBounds(554, 337, 156, 27);

contentPane.add(text\_ing);

text\_ing.setColumns(10);

exit\_btn = **new** JButton("Exit");

exit\_btn.setBounds(555, 556, 125, 29);

contentPane.add(exit\_btn);

exit\_btn.addActionListener(**this**);

run\_btn = **new** JButton("Run(All)");

run\_btn.setBounds(555, 512, 125, 29);

contentPane.add(run\_btn);

run\_btn.addActionListener(**this**);

btn\_step = **new** JButton("Run(1 Step)");

btn\_step.setBounds(555, 468, 125, 29);

btn\_step.addActionListener(**this**);

contentPane.add(btn\_step);

JLabel lblLog = **new** JLabel("Log(instruction)");

lblLog.setBounds(27, 592, 200, 21);

contentPane.add(lblLog);

log = **new** JTextArea();

JScrollPane log\_scroll = **new** JScrollPane(log);

log\_scroll.setBounds(27, 624, 660, 187);

contentPane.add(log\_scroll);

jfc.setFileFilter(**new** FileNameExtensionFilter("txt", "txt"));

// 파일 필터

jfc.setMultiSelectionEnabled(**false**);// 다중 선택 불가

}

// 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 rMgr)

{

*sub\_cnt*++;

*sic*.initialize(objFile, rMgr); //로더를 위해 rmgr의 메모리와 레지스털를 초기화

*sic*.load(objFile, rMgr); //로드 시작

//

//리스트데이타에 명령어를 추가

**for**(**int** i=0; i<*sic*.inst.size();i++)

{

listData.addElement(*sic*.inst.get(i));

//inst\_txt.append(sic.inst.get(i)+"\n");

}

//명령어를 추가한 리스트데이타를 명령어리스트에 추가

list.setListData(listData);

//

//모든 텍스트필드를 초기화한다.

//

pro\_name.setText(*sic*.name.get(0));

start\_addr\_obj.setText(*sic*.start.get(0));

pro\_len.setText(*sic*.length.get(0));

fir\_inst\_addr.setText(*sic*.start.get(0));

start\_addr\_m.setText(*sic*.start.get(0));

a\_dec.setText("0");

b\_dec.setText("0");

s\_dec.setText("0");

t\_dec.setText("0");

l\_dec.setText("0");

pc\_dec.setText("0");

sw\_dec.setText("0");

f\_dec.setText("0");

x\_dec.setText("0");

target.setText("000000");

a\_hex.setText("000000");

b\_hex.setText("000000");

s\_hex.setText("000000");

t\_hex.setText("000000");

l\_hex.setText("000000");

pc\_hex.setText("000000");

x\_hex.setText("000000");

//log에 명령어 추가

**for**(**int** i=0; i<*sic*.inst.size();i++)

{

log.append(*sic*.inst\_name.get(i) + "\n");

}

}

// 하나의 명령어만 수행하는 메소드로써 sic 시뮬레이터에게 작업을 전달한다.

**public** **void** oneStep() {

*sic*.oneStep(*sub\_cnt*,*step*);

//디바이스 이름이 없을 경우에는 빈칸을 넣음

**if**(*rmgr*.device\_name==**null**)

{

text\_ing.setText("");

}

//디바이스 이름이 있을 경우에는 이름 추가

**else**

{

text\_ing.setText(*rmgr*.device\_name);

}

//서브루틴이 시작 되면 h레코드와 e레코드 등을 모두 수정

**if**(*step*+1==*rmgr*.r\_cnt[*sub\_cnt*-1] && *sub\_cnt*!=3)

{

*sub\_cnt*++;

pro\_name.setText(*sic*.name.get(*sub\_cnt*-1));

start\_addr\_obj.setText(*sic*.r\_start.get(*sub\_cnt*-2));

pro\_len.setText(*sic*.length.get(*sub\_cnt*-1));

fir\_inst\_addr.setText(*sic*.r\_start.get(*sub\_cnt*-2));

start\_addr\_m.setText(*sic*.r\_start.get(*sub\_cnt*-2));

}

//타겟 주소 수정

target.setText(String.*format*("%06X",*rmgr*.target[*step*]));

update(); //scroll 업데이트

//

// 레지스터값 수정

//

**if** (*rmgr*.reg\_a[*step*] != -1)

{

a\_dec.setText(Integer.*toString*(*rmgr*.reg\_a[*step*]));

a\_hex.setText(String.*format*("%06X%n", *rmgr*.reg\_a[*step*]));

}

**if** (*rmgr*.reg\_b[*step*] != -1)

{

b\_dec.setText(Integer.*toString*(*rmgr*.reg\_b[*step*]));

b\_hex.setText(String.*format*("%06X%n", *rmgr*.reg\_b[*step*]));

}

**if** (*rmgr*.reg\_x[*step*] != -1)

{

x\_dec.setText(Integer.*toString*(*rmgr*.reg\_x[*step*]));

x\_hex.setText(String.*format*("%06X%n", *rmgr*.reg\_x[*step*]));

}

**if** (*rmgr*.reg\_l[*step*] != -1)

{

l\_dec.setText(Integer.*toString*(*rmgr*.reg\_l[*step*]));

l\_hex.setText(String.*format*("%06X%n", *rmgr*.reg\_l[*step*]));

}

**if** (*rmgr*.reg\_s[*step*] != -1)

{

s\_dec.setText(Integer.*toString*(*rmgr*.reg\_s[*step*]));

s\_hex.setText(String.*format*("%06X%n", *rmgr*.reg\_s[*step*]));

}

**if** (*rmgr*.reg\_t[*step*] != -1)

{

t\_dec.setText(Integer.*toString*(*rmgr*.reg\_t[*step*]));

t\_hex.setText(String.*format*("%06X%n", *rmgr*.reg\_t[*step*]));

}

**if** (*rmgr*.reg\_f[*step*] != -1)

{

f\_dec.setText(Integer.*toString*(*rmgr*.reg\_b[*step*]));

}

**if** (*rmgr*.reg\_pc[*step*] != -1)

{

pc\_dec.setText(Integer.*toString*(*rmgr*.reg\_pc[*step*]));

pc\_hex.setText(String.*format*("%06X%n", *rmgr*.reg\_pc[*step*]));

}

**if** (*rmgr*.reg\_sw[*step*] != -1)

{

sw\_dec.setText(Integer.*toString*(*rmgr*.reg\_sw[*step*]));

}

*step*++;

}

// 남은 명령어를 모두 수행하는 메소드로써 sic 시뮬레이터에 작업을 전달

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

{

*sic*.allStep();

**for**(**int** i=0; i<*sic*.inst.size();i++)

{

oneStep();

}

System.***out***.println(*rmgr*.memory.length);

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

{

**if**(i%6==0)

{

System.***out***.print(" ");

}

**if**(i%24==0)

{

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

}

**if**((**int**)*rmgr*.memory[i]<0)

{

System.***out***.printf("%02X",(**int**)*rmgr*.memory[i]& 0XFF);

}

**else**

{

System.***out***.printf("%02X",(**int**)*rmgr*.memory[i]);

}

}

}

// 작업이 완료되었을 때 변화된 결과를 화면에 업데이트

**public** **void** update() {

list.setSelectedIndex(*step*);

**if**(*step*==12)

{

scroll.getVerticalScrollBar().setValue(scroll.getVerticalScrollBar().getMaximum());

}

}

//버튼을 수행하는 메소드

@Override

**public** **void** actionPerformed(ActionEvent e)

{

//파일 열기 버튼

**if** (e.getSource() == btn\_open)

{

**if** (jfc.showOpenDialog(**this**) == JFileChooser.***APPROVE\_OPTION***)

{

*filename* = jfc.getSelectedFile().toString();

File file = **new** File(*filename*);

text\_file.setText(*filename*);

initialize(file,*rmgr*);

}

}

//진행(step) 버튼

**else** **if**(e.getSource()==btn\_step)

{

oneStep();

}

//실행(ALL) 버튼

**else** **if**(e.getSource()==run\_btn)

{

allStep();

}

//종료 버튼

**else** **if**(e.getSource()==exit\_btn)

{

System.*exit*(0);

}

}

}

**ResourceManager.class**

**package** project;

**import** java.util.ArrayList;

**public** **class** ResourceManager **extends** VisualSimulator {

**static** String *pro\_name*; //프로그램의 이름을 저장(서브루틴 포함)

**static** **int** *start\_addr*; //프로그램의 시작주소

**static** **int** *length*; //프로그램의 길이

**static** **int** *first*; //시작주소

**static** **int**[] *r\_cnt* = **new** **int**[10]; //절대주소를 위해 길이를 저장

**static** **int** *reg\_cnt* = 0; //레지스터의 인덱스

**static** String *device\_name*=**null**; //디바이스 이름

**byte**[] device = **new** **byte**[100]; //디바이스의 값을 저장

**byte**[] memory = **new** **byte**[10000]; //가상 메모리

**static** **int**[] *target* = **new** **int**[300]; //

**static** **int**[] *reg\_a* = **new** **int**[300]; //

**static** **int**[] *reg\_x* = **new** **int**[300]; // 각 레지스터의 배열

**static** **int**[] *reg\_l* = **new** **int**[300]; // (명령어 마다)

**static** **int**[] *reg\_b* = **new** **int**[300]; //

**static** **int**[] *reg\_s* = **new** **int**[300]; //

**static** **int**[] *reg\_t* = **new** **int**[300]; //

**static** **int**[] *reg\_f* = **new** **int**[300]; //

**static** **int**[] *reg\_pc* = **new** **int**[300]; //

**static** **int**[] *reg\_sw* = **new** **int**[300]; //

// 메모리 영역을 초기화 하는 메소드

**public** **void** initializeMemory() {

**for**(**int** i=0; i<10000; i++)

{

memory[i]=0;

}

}

// 각 레지스터 값을 초기화 하는 메소드

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

{

**for** (**int** i = 0; i < 300; i++)

{

*reg\_a*[i] = -1;

*reg\_b*[i] = -1;

*reg\_x*[i] = -1;

*reg\_l*[i] = -1;

*reg\_s*[i] = -1;

*reg\_t*[i] = -1;

*reg\_f*[i] = -1;

*reg\_pc*[i] = -1;

*reg\_sw*[i] = -1;

}

}

// 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(String devName) {

*device\_name*=devName;

}

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

**public** **void** writeDevice(String devName, **byte**[] data, **int** size) {

**if**(*device\_name*==devName)

{

**for**(**int** i=0; i<size; i++)

{

device[i]=data[i];

}

}

}

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

**public** **byte**[] readDevice(String devName, **int** size)

{

**byte** [] by = **new** **byte**[100];

**if**(*device\_name*==devName)

{

**for**(**int** i=0; i<size; i++)

{

by[i]=device[i];

}

}

**return** by;

}

// 메모리 영역에 값을 쓰는 메소드

**public** **void** setMemory(**int** locate, **byte**[] data, **int** size)

{

**for**(**int** i=0; i<size; i++)

{

memory[locate]=data[i];

locate++;

}

}

// 프로그램의 이름를 지정

**public** **void** setName(String str) {

*pro\_name*=str;

}

//레지스터에 값을 세팅하는 메소드 str은 명령어/value는 값/cnt는 레지스터 순서

**public** **void** setRe(String str,**int** value,**int** cnt)

{

**if**(str.equals("JSUB"))

{

*reg\_l*[cnt]=value;

*target*[cnt]=value;

}

**else** **if**(str.equals("STCH") || str.equals("STX"))

{

*target*[cnt]=value;

}

**else** **if**(str.equals("LDT"))

{

*reg\_t*[cnt]=value;

*target*[cnt]=value;

}

**else** **if**(str.equals("LDCH"))

{

*reg\_a*[cnt]=value;

*target*[cnt]=value;

}

}

// 레지스터에 값을 세팅하는 메소드. regNum1,2은 레지스터 종류를 나타낸다.

//reg\_r은 주소방식 value는 값 \loc는 LOCCTR \flag 는 저장,로드 명령어 \format은 명령어의 형식 \ inst는 명령어 이름

//flag 0은 적재/1은 저장/-1은 다른 명령어 reg\_r은 상대주소지정방식에 따라 3은 베이스, 8은 pc, -1은 절대주소지정방식 또는 2형식이다.

**public** **void** setRegister(**int** regNum1, **int** regNum2, **int** reg\_r, **int** value, **int** loc, **int** flag,**int** format, String inst)

{

*target*[*reg\_cnt*]=value; //타겟주소 저장

**if** (reg\_r == 3) //베이스 레지스터를 사용하는 경우

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_b*[k] >= 0)

{

*reg\_b*[*reg\_cnt*] = *reg\_b*[k];

**break**;

}

}

//레지스터 넘버에 따라 나눔

**switch** (regNum1) {

**case** 0:

{

**if** (flag == 0) //A레지스터, flag는 저장/적재 명령어를 나눈 기준 0은 적재

{

*reg\_a*[*reg\_cnt*] = value;

}

**else**

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_a*[k] >= 0)

{

*reg\_a*[*reg\_cnt*] = *reg\_a*[k];

**break**;

}

}

}

**break**;

}

**case** 1: { //X레지스터인 경우

**if** (flag == 0)

{

*reg\_x*[*reg\_cnt*] = value;

}

**else**

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

*reg\_x*[*reg\_cnt*] = *reg\_x*[k];

**break**;

}

}

}

**break**;

}

**case** 2:

{

**if** (flag == 0)

{

*reg\_l*[*reg\_cnt*]=value;

}

**else**

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_l*[k] >= 0)

{

*reg\_l*[*reg\_cnt*] = *reg\_l*[k];

**break**;

}

}

}

**break**;

}

**case** 3: //base레지스터인 경우

{

**if** (flag == 0)

{

*reg\_b*[*reg\_cnt*] = value;

}

**else**

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_b*[k] >= 0)

{

*reg\_b*[*reg\_cnt*] = *reg\_b*[k];

**break**;

}

}

}

**break**;

}

**case** 4: //S레지스터인 경우

{

**if** (flag == 0)

{

*reg\_s*[*reg\_cnt*] = value;

} **else** {

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_s*[k] >= 0)

{

*reg\_s*[*reg\_cnt*] = *reg\_s*[k];

**break**;

}

}

}

**break**;

}

**case** 5: //t레지스터인 경우

{

**if** (flag == 0)

{

*reg\_t*[*reg\_cnt*] = value;

}

**else**

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_t*[k] >= 0)

{

*reg\_t*[*reg\_cnt*] = *reg\_t*[k];

**break**;

}

}

}

**break**;

}

//분기 명령어일 경우

**case** 8:

{

*reg\_pc*[*reg\_cnt*]=value;

}

//TIX일 경우

**case** 11:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

*reg\_x*[*reg\_cnt*] = *reg\_x*[k]+1;

**break**;

}

}

}

**default**:

**break**;

}

**if** (regNum2 == 1) //레지스터가 2개인 경우

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

*reg\_x*[*reg\_cnt*] = *reg\_x*[k];

**break**;

}

}

}

*reg\_cnt*++;

}

**else** **if** (reg\_r == 8) //pc상대주소지정방식인 경우

{

*reg\_pc*[*reg\_cnt*] = loc;

**switch** (regNum1) { //레지스터 넘버에 따라 나눔 베이스 상대주소지정방식과 동일한 방식

**case** 0:

{

**if** (flag == 0)

{

*reg\_a*[*reg\_cnt*] = value;

}

**else**

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_a*[k] >= 0)

{

*reg\_a*[*reg\_cnt*] = *reg\_a*[k];

**break**;

}

}

}

**break**;

}

**case** 1: {

**if** (flag == 0)

{

*reg\_x*[*reg\_cnt*] = value;

}

**else**

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

*reg\_x*[*reg\_cnt*] = *reg\_x*[k];

**break**;

}

}

}

**break**;

}

**case** 2:

{

**if** (flag == 0)

{

*reg\_l*[*reg\_cnt*] = value;

}

**else**

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_l*[k] >= 0)

{

*reg\_l*[*reg\_cnt*] = *reg\_l*[k];

**break**;

}

}

}

**break**;

}

**case** 3:

{

**if** (flag == 0)

{

*reg\_b*[*reg\_cnt*] = value;

}

**else**

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_b*[k] >= 0)

{

*reg\_b*[*reg\_cnt*] = *reg\_b*[k];

**break**;

}

}

}

**break**;

}

**case** 4:

{

**if** (flag == 0)

{

*reg\_s*[*reg\_cnt*] = value;

} **else** {

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_s*[k] >= 0)

{

*reg\_s*[*reg\_cnt*] = *reg\_s*[k];

**break**;

}

}

}

**break**;

}

**case** 5:

{

**if** (flag == 0)

{

*reg\_t*[*reg\_cnt*] = value;

}

**else**

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_t*[k] >= 0)

{

*reg\_t*[*reg\_cnt*] = *reg\_t*[k];

**break**;

}

}

}

**break**;

}

//분기 명령어일 경우

**case** 8:

{

*reg\_pc*[*reg\_cnt*]=value;

}

**case** 11:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

*reg\_x*[*reg\_cnt*] = *reg\_x*[k]+1;

**break**;

}

}

}

**default**:

**break**;

}

**if** (regNum2 == 1)

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

*reg\_x*[*reg\_cnt*] = *reg\_x*[k];

**break**;

}

}

}

*reg\_cnt*++;

}

**else** **if** (reg\_r == -1) //절대주소지정방식이거나 2형식일 경우

{

//2형식일 경우

**if**(format==2)

{

**if**(inst.equals("ADDR")) //addr인 경우

{

**int** rg1 = 0;

**switch**(regNum1)

{

**case** 0:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_a*[k] >= 0)

{

rg1 = *reg\_a*[k];

*reg\_a*[*reg\_cnt*] = *reg\_a*[k];

**break**;

}

}

}

**case** 1:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

rg1 = *reg\_x*[k];

*reg\_x*[*reg\_cnt*] = *reg\_x*[k];

**break**;

}

}

}

**case** 2:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_l*[k] >= 0)

{

rg1 = *reg\_l*[k];

*reg\_l*[*reg\_cnt*] = *reg\_l*[k];

**break**;

}

}

}

**case** 3:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_b*[k] >= 0)

{

rg1 = *reg\_b*[k];

*reg\_b*[*reg\_cnt*] = *reg\_b*[k];

**break**;

}

}

}

**case** 4:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_s*[k] >= 0)

{

rg1 = *reg\_s*[k];

*reg\_s*[*reg\_cnt*] = *reg\_s*[k];

**break**;

}

}

}

**case** 5:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_t*[k] >= 0)

{

rg1 = *reg\_t*[k];

*reg\_t*[*reg\_cnt*] = *reg\_t*[k];

**break**;

}

}

}

**case** 6:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_f*[k] >= 0)

{

rg1 = *reg\_f*[k];

*reg\_f*[*reg\_cnt*] = *reg\_f*[k];

**break**;

}

}

}

**case** 8:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_pc*[k] >= 0)

{

rg1 = *reg\_pc*[k];

*reg\_pc*[*reg\_cnt*] = *reg\_pc*[k];

**break**;

}

}

}

**case** 9:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_sw*[k] >= 0)

{

rg1 = *reg\_sw*[k];

*reg\_sw*[*reg\_cnt*] = *reg\_sw*[k];

**break**;

}

}

}

}

**switch**(regNum2)

{

**case** 0:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_a*[k] >= 0)

{

*reg\_a*[*reg\_cnt*] = *reg\_a*[k]+rg1;

**break**;

}

}

}

**case** 1:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

*reg\_x*[*reg\_cnt*] = *reg\_x*[k]+rg1;

**break**;

}

}

}

**case** 2:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_l*[k] >= 0)

{

*reg\_l*[*reg\_cnt*] = *reg\_l*[k]+rg1;

**break**;

}

}

}

**case** 3:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_b*[k] >= 0)

{

*reg\_b*[*reg\_cnt*] = *reg\_b*[k]+rg1;

**break**;

}

}

}

**case** 4:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_s*[k] >= 0)

{

*reg\_s*[*reg\_cnt*] = *reg\_s*[k]+rg1;

**break**;

}

}

}

**case** 5:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_t*[k] >= 0)

{

*reg\_t*[*reg\_cnt*] = *reg\_t*[k]+rg1;

**break**;

}

}

}

**case** 6:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_f*[k] >= 0)

{

*reg\_f*[*reg\_cnt*] = *reg\_f*[k]+rg1;

**break**;

}

}

}

**case** 8:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_pc*[k] >= 0)

{

*reg\_pc*[*reg\_cnt*] = *reg\_pc*[k]+rg1;

**break**;

}

}

}

**case** 9:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_sw*[k] >= 0)

{

*reg\_sw*[*reg\_cnt*] = *reg\_sw*[k]+rg1;

**break**;

}

}

}

}

}

**else** **if**(inst.equals("CLEAR")) //clear인 경우

{

**switch**(regNum1)

{

**case** 0:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_a*[k] >= 0)

{

*reg\_a*[*reg\_cnt*] = 0;

**break**;

}

}

}

**case** 1:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

*reg\_x*[*reg\_cnt*] = 0;

**break**;

}

}

}

**case** 2:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_l*[k] >= 0)

{

*reg\_l*[*reg\_cnt*] = 0;

**break**;

}

}

}

**case** 3:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_b*[k] >= 0)

{

*reg\_b*[*reg\_cnt*] = 0;

**break**;

}

}

}

**case** 4:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_s*[k] >= 0)

{

*reg\_s*[*reg\_cnt*] = 0;

**break**;

}

}

}

**case** 5:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_t*[k] >= 0)

{

*reg\_t*[*reg\_cnt*] = 0;

**break**;

}

}

}

**case** 6:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_f*[k] >= 0)

{

*reg\_f*[*reg\_cnt*] = 0;

**break**;

}

}

}

**case** 8:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_pc*[k] >= 0)

{

*reg\_pc*[*reg\_cnt*] = 0;

**break**;

}

}

}

**case** 9:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_sw*[k] >= 0)

{

*reg\_sw*[*reg\_cnt*] = 0;

**break**;

}

}

}

}

}

**else** **if**(inst.equals("COMPR")) //compr인 경우

{

**switch**(regNum1)

{

**case** 0:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_a*[k] >= 0)

{

*reg\_a*[*reg\_cnt*] = *reg\_a*[k];

**break**;

}

}

}

**case** 1:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

*reg\_x*[*reg\_cnt*] = *reg\_x*[k];

**break**;

}

}

}

**case** 2:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_l*[k] >= 0)

{

*reg\_l*[*reg\_cnt*] = *reg\_l*[k];

**break**;

}

}

}

**case** 3:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_b*[k] >= 0)

{

*reg\_b*[*reg\_cnt*] = *reg\_b*[k];

**break**;

}

}

}

**case** 4:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_s*[k] >= 0)

{

*reg\_s*[*reg\_cnt*] = *reg\_s*[k];

**break**;

}

}

}

**case** 5:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_t*[k] >= 0)

{

*reg\_t*[*reg\_cnt*] = *reg\_t*[k];

**break**;

}

}

}

**case** 6:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_f*[k] >= 0)

{

*reg\_f*[*reg\_cnt*] = *reg\_f*[k];

**break**;

}

}

}

**case** 8:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_pc*[k] >= 0)

{

*reg\_pc*[*reg\_cnt*] = *reg\_pc*[k];

**break**;

}

}

}

**case** 9:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_sw*[k] >= 0)

{

*reg\_sw*[*reg\_cnt*] = *reg\_sw*[k];

**break**;

}

}

}

}

}

**else** //나머지 명령어인 경우

{

**switch**(regNum1)

{

**case** 0:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_a*[k] >= 0)

{

*reg\_a*[*reg\_cnt*] = *reg\_a*[k];

**break**;

}

}

}

**case** 1:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

*reg\_x*[*reg\_cnt*] = *reg\_x*[k];

**break**;

}

}

}

**case** 2:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_l*[k] >= 0)

{

*reg\_l*[*reg\_cnt*] = *reg\_l*[k];

**break**;

}

}

}

**case** 3:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_b*[k] >= 0)

{

*reg\_b*[*reg\_cnt*] = *reg\_b*[k];

**break**;

}

}

}

**case** 4:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_s*[k] >= 0)

{

*reg\_s*[*reg\_cnt*] = *reg\_s*[k];

**break**;

}

}

}

**case** 5:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_t*[k] >= 0)

{

*reg\_t*[*reg\_cnt*] = *reg\_t*[k];

**break**;

}

}

}

**case** 6:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_f*[k] >= 0)

{

*reg\_f*[*reg\_cnt*] = *reg\_f*[k];

**break**;

}

}

}

**case** 8:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_pc*[k] >= 0)

{

*reg\_pc*[*reg\_cnt*] = *reg\_pc*[k];

**break**;

}

}

}

**case** 9:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_sw*[k] >= 0)

{

*reg\_sw*[*reg\_cnt*] = *reg\_sw*[k];

**break**;

}

}

}

}

}

}

**else** //4형식 등 절대주소지정 방식일 경우

{

**if**(inst.contains("LD")) //적재방식 명령어일 경우

{

**switch** (regNum1) {

**case** 0:

{

*reg\_a*[*reg\_cnt*] = value;

**break**;

}

**case** 1: {

*reg\_x*[*reg\_cnt*] = value;

**break**;

}

**case** 2:

{

*reg\_l*[*reg\_cnt*]=value;

**break**;

}

**case** 3:

{

*reg\_b*[*reg\_cnt*] = value;

**break**;

}

**case** 4:

{

*reg\_s*[*reg\_cnt*] = value;

**break**;

}

**case** 5:

{

*reg\_t*[*reg\_cnt*] = value;

**break**;

}

//분기 명령어일 경우

**case** 8:

{

*reg\_pc*[*reg\_cnt*]=value;

}

**default**:

**break**;

}

}

**else** **if**(inst.contains("ST")) //저장명령어일 경우

{

**switch** (regNum1) {

**case** 0:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_a*[k] >= 0)

{

*reg\_a*[*reg\_cnt*] = *reg\_a*[k];

**break**;

}

}

**break**;

}

**case** 1: {

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_x*[k] >= 0)

{

*reg\_x*[*reg\_cnt*] = *reg\_x*[k];

**break**;

}

}

**break**;

}

**case** 2:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_l*[k] >= 0)

{

*reg\_l*[*reg\_cnt*] = *reg\_l*[k];

**break**;

}

}

**break**;

}

**case** 3:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_b*[k] >= 0)

{

*reg\_b*[*reg\_cnt*] = *reg\_b*[k];

**break**;

}

}

**break**;

}

**case** 4:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_s*[k] >= 0)

{

*reg\_s*[*reg\_cnt*] = *reg\_s*[k];

**break**;

}

}

**break**;

}

**case** 5:

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_t*[k] >= 0)

{

*reg\_t*[*reg\_cnt*] = *reg\_t*[k];

**break**;

}

}

**break**;

}

//분기 명령어일 경우

**case** 8:

{

*reg\_pc*[*reg\_cnt*]=value;

}

**default**:

**break**;

}

}

**else** **if**(inst.equals("JSUB")) //JSUB명령어일 경우

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_pc*[k] >= 0)

{

*reg\_l*[*reg\_cnt*] = *reg\_pc*[k];

**break**;

}

}

*reg\_pc*[*reg\_cnt*]=value;

}

**else** **if**(inst.equals("RSUB")) //RSUB명령어일 경우

{

**for** (**int** k = *reg\_cnt* - 1; k >= 0; k--)

{

**if** (*reg\_l*[k] >= 0)

{

*reg\_pc*[*reg\_cnt*] = *reg\_l*[k];

**break**;

}

}

}

}

*reg\_cnt*++;

}

}

// 메모리 영역에서 값을 읽어오는 메소드

**public** **byte**[] getMemory(**int** locate, **int** size) {

**byte**[] by = **null**;

**int** k=0;

**for**(**int** i=locate; i<locate+size; i++)

{

by[k]=memory[i];

k++;

}

**return** by;

}

// 레지스터에서 값을 가져오는 메소드

**public** **int** getRegister(**int** regNum,**int** step) {

**switch**(regNum)

{

**case** 0:

{

**return** *reg\_a*[step];

}

**case** 1:

{

**return** *reg\_x*[step];

}

**case** 2:

{

**return** *reg\_l*[step];

}

**case** 3:

{

**return** *reg\_b*[step];

}

**case** 4:

{

**return** *reg\_s*[step];

}

**case** 5:

{

**return** *reg\_t*[step];

}

**case** 6:

{

**return** *reg\_f*[step];

}

**case** 8:

{

**return** *reg\_pc*[step];

}

**case** 9:

{

**return** *reg\_sw*[step];

}

**default**:

**break**;

}

**return** step;

}

}

**SICXE.class**

**package** project;

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.util.ArrayList;

//시뮬레이터를 동작시키기 위한 세팅을 수행한다.

//메모리 작업 등 실질적인 초기화 작업을 수행한다.

**public** **class** SICXE **extends** VisualSimulator **implements** SicSimulator, SicLoader {

BufferedReader br = **null**;

**static** Token\_unit[] *ref* = **new** Token\_unit[50]; //참조값을 저장하는 객체배열

**static** Token\_unit[] *def* = **new** Token\_unit[50]; //정의값을 저장한는 객체배열

**static** Token\_unit[] *mod* = **new** Token\_unit[50]; //수정값을 저장하는 객체배열

**static** Token\_unit[] *dev* = **new** Token\_unit[50]; //디바이스의 이름과 주소를 저장하는 객체배열

**static** **int** *ref\_cnt*=0; //참조값 개수

**static** **int** *def\_cnt*=0; //정의값 개수

**static** **int** *sub\_cnt*=0; //루틴의 개수

**static** **int** *mod\_cnt*=0; //수정값 개수

**static** **int** *dev\_cnt*=0; //디바이스 개수

**static** **byte**[] *r\_byte* = **new** **byte**[100]; //디바이스에서 읽어올 바이트배열

**static** ArrayList<Integer> *f\_mod* = **new** ArrayList<Integer>(); //mod레코드에서 상대주소를 저장하는 리스트

**static** ArrayList<String> *inst* = **new** ArrayList<String>(); //명령어의 주소를 저장하는 리스트

**static** ArrayList<String> *inst\_name* = **new** ArrayList<String>(); //명령어의 이름을 저장하는 리스트

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

**static** ArrayList<String> *r\_start* = **new** ArrayList<String>(); //루틴의 실제시작주소

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

**static** ArrayList<String> *t\_start* = **new** ArrayList<String>(); //T레코드의 시작주소

**static** ArrayList<String> *t\_length* = **new** ArrayList<String>(); //T레코드의 길이

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

**static** ArrayList<String> *target* = **new** ArrayList<String>(); //target 리스트

**static** **int** *loc*=0; //loc값

//목적코드를 읽어 메모리에 로드한다.

//목적코드의 각 헤더(H, T, M 등)를 읽어 동작을 수행한다.

@Override

**public** **void** load(File objFile, ResourceManager rMgr)

{

String line;

**int** b\_cnt=0;

**byte**[] by = **new** **byte**[3]; //메모리에 저장할떄 바이트로 변환(packing)할 바이트배열

//각 token을 초기화

**for**(**int** i=0; i<50; i++)

{

*ref*[i]=**new** Token\_unit(**null**, 0);

*def*[i]=**new** Token\_unit(**null**, 0);

*mod*[i]=**new** Token\_unit(**null**, 0);

*dev*[i]=**new** Token\_unit(**null**, 0);

}

**try** {

br = **new** BufferedReader(**new** FileReader(objFile));

//object program을 한 줄씩 읽어들임

**while**((line=br.readLine())!=**null**)

{

**if**(line.charAt(0)=='H') //H레코드일 경우

{

*sub\_cnt*++;

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

//메인 프로그램일 경우

**if**(*sub\_cnt*==1)

{

rMgr.setName(line.substring(1,7));

}

//서브루틴인 경우에는 상대주소가 아닌 절대주소로 계산

**if**(*sub\_cnt*>=2)

{

**int** s1 = 0;

**int** s2 = 0;

**for**(**int** i=*start*.size()-1; i>=0; i--)

{

s1+=Integer.*parseInt*(*length*.get(i),16);

}

s2=Integer.*parseInt*(line.substring(7,13),16);

*loc*=s1+s2;

String s=String.*format*("%06X", s1+s2);

*r\_start*.add(s);

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

b\_cnt=0;

b\_cnt+=Integer.*parseInt*(s,16);

}

*start*.add(line.substring(7,13));

*length*.add(line.substring(13,19));

}

**else** **if**(line.charAt(0)=='D') //D레코드일 경우 def 토큰에 주소,이름,무슨루틴인지를 각각 저장

{

*def*[*def\_cnt*].sub\_cnt=*sub\_cnt*;

*def*[*def\_cnt*].name=line.substring(1,7);

**if**(*sub\_cnt*>=2)

{

*def*[*def\_cnt*++].addr=Integer.*parseInt*(line.substring(7,13),16)

+Integer.*parseInt*(*r\_start*.get(*sub\_cnt*-2),16);

}

**else**

{

*def*[*def\_cnt*++].addr=Integer.*parseInt*(line.substring(7,13),16);

}

**if**(line.length()>14)

{

*def*[*def\_cnt*].sub\_cnt=*sub\_cnt*;

*def*[*def\_cnt*].name=line.substring(13,19);

**if**(*sub\_cnt*>=2)

{

*def*[*def\_cnt*++].addr=Integer.*parseInt*(line.substring(19,25),16)

+Integer.*parseInt*(*r\_start*.get(*sub\_cnt*-2),16);

}

**else**

{

*def*[*def\_cnt*++].addr=Integer.*parseInt*(line.substring(19,25),16);

}

}

**if**(line.length()>26)

{

*def*[*def\_cnt*].sub\_cnt=*sub\_cnt*;

*def*[*def\_cnt*].name=line.substring(25,31);

**if**(*sub\_cnt*>=2)

{

*def*[*def\_cnt*++].addr=Integer.*parseInt*(line.substring(31,line.length()),16)

+Integer.*parseInt*(*r\_start*.get(*sub\_cnt*-2),16);

}

**else**

{

*def*[*def\_cnt*++].addr=Integer.*parseInt*(line.substring(31,line.length()),16);

}

}

}

**else** **if**(line.charAt(0)=='R') //R레코드일 경우도 D레코드와 동일하게 토큰에 저장

{

*ref*[*ref\_cnt*].sub\_cnt=*sub\_cnt*;

*ref*[*ref\_cnt*++].name=line.substring(1,7);

**if**(line.length()>8)

{

*ref*[*ref\_cnt*].sub\_cnt=*sub\_cnt*;

*ref*[*ref\_cnt*++].name=line.substring(7,13);

}

**if**(line.length()>14)

{

*ref*[*ref\_cnt*].sub\_cnt=*sub\_cnt*;

*ref*[*ref\_cnt*++].name=line.substring(13,line.length());

}

}

//T레코드일 경우 길이에 맞게 파싱하여 명령어를 구분하고 메모리에 로더하고 레지스터값 지정

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

{

*t\_start*.add(line.substring(1,7)); //T레코드의 시작주소 저장

*t\_length*.add(line.substring(7,9)); //T레코드의 길이 저장

*loc*+=Integer.*parseInt*(line.substring(1,7),16);

**int** len = Integer.*parseInt*(line.substring(7,9),16); //레코드의 길이

**int** cnt=9; //파싱하기 위한 count변수

**while**(**true**)

{

**if**((cnt/2)>len+3) //길이를 넘었을 경우 반복문 종료

{

**break**;

}

**if**(line.length()<cnt+3) //레코드 마지막에 바이트만 있을 경우 메모리에 로더하고 반복문 종료

{

**if**(Integer.*parseInt*(line.substring(cnt,cnt+2),16)==0)

{

**break**;

}

by=*byteorder*(line.substring(cnt,cnt+2), 2);

*dev*[*dev\_cnt*].sub\_cnt=*sub\_cnt*;

*dev*[*dev\_cnt*].addr=*loc*;

*dev*[*dev\_cnt*++].name=line.substring(cnt,cnt+2);

rMgr.setMemory(b\_cnt, by, 1);

b\_cnt+=1;

cnt+=2;

**break**;

}

**int** op=Integer.*parseInt*(line.substring(cnt,cnt+2),16); //op코드 저장(ni값도 포함)

**int** xbpe=Integer.*parseInt*(line.substring(cnt+2,cnt+3),16); //xbpe값 저장

**int** cal=Integer.*parseInt*(line.substring(cnt+2,cnt+4),16); //명령어 구분을 위한 값

**int** k = *search\_opcode*(op); //op코드에 따라 명령어인지 구분

**if**(inst\_table[k].format==2) //2형식일 경우 xbpe=0

{

xbpe=0;

}

**if**(k>-1) //명령어일 경우

{

**int** jp=0;

**int** jp2=0;

**int** jp3=0;

String iname = inst\_table[k].name; //명령어의 이름

//op코드가 같은 값일 경우를 대비한 if문

**if**(xbpe>15 || inst\_table[k].format!=2 && (cal>=65 && cal<=90))

{

jp=Integer.*parseInt*(line.substring(cnt,cnt+2),16);

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

{

jp2 = Integer.*parseInt*(line.substring(cnt + 2, cnt + 4), 16);

}

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

{

jp3 = Integer.*parseInt*(line.substring(cnt + 4, cnt + 6), 16);

}

**if**(jp3<=65 && jp2<=65)

{

by=*byteorder*(line.substring(cnt,cnt+2), 2);

rMgr.setMemory(b\_cnt, by, 1);

b\_cnt++;

cnt+=2;

}

**else**

{

by=*byteorder*(line.substring(cnt,cnt+6), 6);

rMgr.setMemory(b\_cnt, by, 3);

b\_cnt+=3;

cnt+=6;

}

}

//명령어인 경우

**else**

{

*inst\_name*.add(iname); //명령어 이름을 추가

//3형식일 경우 크기 6의 문자열을 3바이트로 파싱하여 메모리에 올림

**if** (xbpe % 2 == 0 && inst\_table[k].format==3)

{

*inst*.add(line.substring(cnt, cnt + 6));

by=*byteorder*(line.substring(cnt,cnt+6), 6);

rMgr.setMemory(b\_cnt, by, 3);

b\_cnt+=3;

cnt+=6;

*loc*+=3;

}

//2형식일 경우

**else** **if**(inst\_table[k].format==2)

{

*inst*.add(line.substring(cnt, cnt + 4));

by=*byteorder*(line.substring(cnt,cnt+4), 4);

rMgr.setMemory(b\_cnt, by, 2);

b\_cnt+=2;

cnt+=4;

*loc*+=2;

}

//4형식일 경우

**else** **if**(xbpe>=1)

{

*inst*.add(line.substring(cnt, cnt + 8));

by=*byteorder*(line.substring(cnt,cnt+8), 8);

rMgr.setMemory(b\_cnt, by, 4);

b\_cnt+=4;

cnt+=8;

*loc*+=4;

}

//명령어가 아닌 경우를 대비

**else** **if**(xbpe==0)

{

*inst\_name*.remove(*inst\_name*.size()-1);

by=*byteorder*(line.substring(cnt,cnt+2), 2);

*dev*[*dev\_cnt*].sub\_cnt=*sub\_cnt*;

*dev*[*dev\_cnt*].addr=*loc*;

*dev*[*dev\_cnt*++].name=line.substring(cnt,cnt+2);

rMgr.initialDevice(line.substring(cnt,cnt+2));

rMgr.setMemory(b\_cnt, by, 1);

b\_cnt+=1;

cnt+=6;

}

//레지스터 A를 쓰고 3/4형식일 경우 레지스터값 저장

**if** (iname.equals("COMP") || iname.equals("AND") || iname.equals("ADD") || iname.equals("LDA")

|| iname.equals("DIV") || iname.equals("LDCH") || iname.equals("MUL")

|| iname.equals("OR") || iname.equals("RD") || iname.equals("SUB")|| iname.equals("WD")

|| iname.equals("SSK") || iname.equals("STA") || iname.equals("STCH"))

{

**if**(xbpe>=8 && iname.equals("STCH")) //x레지스터를 사용하고 STCH명령어일 경우

{

**if**(xbpe==12) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(0, 1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==10) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(0,1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,1,3,

inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(0, 1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3,

inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(0, 1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 1, 4,

inst\_table[k].name);

}

}

}

**else** **if**(xbpe>=8 && iname.equals("LDCH")) //x레지스터를 사용하고 LDCH인 경우

{

**if**(xbpe==12) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(0, 1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==10) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(0,1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,0,3

,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(0, 1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(0, 1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 0, 4

,inst\_table[k].name);

}

}

}

**else** **if**(iname.contains("ST")) //store명령어일 경우

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(0, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(0,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,1,3

,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(0, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(0, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 1, 4

,inst\_table[k].name);

}

}

}

**else** //나머지 명령어일 경우

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(0, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(0,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,0,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(0, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(0, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 0, 4

,inst\_table[k].name);

}

}

}

}

//B레지스터에 로드하거나 메모리에 B레지스터를 저장한경우

**else** **if** (iname.equals("LDB") || iname.equals("STB"))

{

**if**(iname.equals("LDB"))

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(3, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(3,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,0,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(3, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(3, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 0, 4

,inst\_table[k].name);

}

}

}

**else**

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(3, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(3,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,1,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(3, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(3, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 1, 4

,inst\_table[k].name);

}

}

}

}

//T레지스터를 사용하는 경우

**else** **if** (iname.equals("LDT") || iname.equals("STT"))

{

**if**(iname.equals("LDT"))

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(5, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(5,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,0,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(5, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(5, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 0, 4

,inst\_table[k].name);

}

}

}

**else**

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(5, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(5,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,1,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(5, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(5, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 1, 4

,inst\_table[k].name);

}

}

}

}

//S레지스터를 사용하는 경우

**else** **if** (iname.equals("LDS") || iname.equals("STS"))

{

**if**(iname.equals("LDS"))

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(4, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(4,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,0,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(4, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(4, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 0, 4

,inst\_table[k].name);

}

}

}

**else**

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(4, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(4,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,1,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(4, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(4, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 1, 4

,inst\_table[k].name);

}

}

}

}

//L레지스터를 사용하는 경우

**else** **if** (iname.equals("LDL") || iname.equals("STL"))

{

**if**(iname.equals("LDL"))

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(2, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(2,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,0,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(2, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(2, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 0, 4

,inst\_table[k].name);

}

}

}

**else**

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(2, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(2,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,1,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(2, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(2, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 1, 4

,inst\_table[k].name);

}

}

}

}

//X레지스터를 사용하고 3/4형식일 경우

**else** **if** (iname.equals("TIX") || iname.equals("STX") || iname.equals("LDX"))

{

**if**(iname.equals("LDX"))

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(1, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(1,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,0,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(1, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(1, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 0, 4

,inst\_table[k].name);

}

}

}

**else** **if**(iname.equals("STX"))

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(1, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(1,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,1,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(1, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 1, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(1, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 1, 4

,inst\_table[k].name);

}

}

}

**else**

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(11, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(11,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,0,3,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(11, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(11, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 0, 4

,inst\_table[k].name);

}

}

}

}

//PC레지스터를 이용하여 분기하는 경우

**else** **if** (iname.equals("J") || iname.equals("JSUB") || iname.equals("RSUB") || iname.equals("JEQ")

|| iname.equals("JLT") || iname.equals("JGT"))

{

**if**(iname.equals("JSUB"))

{

**if**(xbpe%2==0) //3형식일 경우

{

rMgr.setRegister(82, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(82, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 0, 4

,inst\_table[k].name);

}

}

**else** **if**(iname.equals("RSUB")) //명령어가 RSUB인 경우

{

rMgr.setRegister(81, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else**

{

**if**(xbpe==4) //base 레지스터를 사용하는 경우

{

rMgr.setRegister(8, -1, 3,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** **if**(xbpe==2) //pc 레지스터를 사용하는 경우

{

rMgr.setRegister(8,-1,8,Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3,6),16),*loc*,0,3

,inst\_table[k].name);

}

**else** //절대 주소일 경우

{

**if**(xbpe%2==0) //직접주소지정 방식일 경우

{

rMgr.setRegister(8, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 6), 16), *loc*, 0, 3

,inst\_table[k].name);

}

**else** //4형식일 경우

{

rMgr.setRegister(8, -1, -1,

Integer.*parseInt*(*inst*.get(*inst*.size()-1).substring(3, 8), 16), *loc*, 0, 4

,inst\_table[k].name);

}

}

}

}

//2형식일 경우

//레지스터가 2개일 경우와 1개일 경우를 나눠서 레지스터값 저장

**else** **if**(inst\_table[k].format==2)

{

**if**(iname.equals("ADDR") || iname.equals("CLEAR") || iname.equals("DIVR") || iname.equals("COMPR")

|| iname.equals("MULR") || iname.equals("RMO") || iname.equals("SUBR") ||

iname.equals("SHIFTR") || iname.equals("SHIFTL"))

{

rMgr.setRegister(Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(2),16), Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(3),16), -1, 0, 0, 0, 2,inst\_table[k].name);

}

**else**

{

rMgr.setRegister(Integer.*parseInt*(*inst*.get(*inst*.size()-1)

.substring(2),16), -1, -1, 0, 0, 0, 2,inst\_table[k].name);

}

}

}

}

//만약을 대비한 명령어가 아닌 경우

**else**

{

**if**(xbpe>15 && inst\_table[k].format!=2 || cal>=65)

{

**int** jp=Integer.*parseInt*(line.substring(cnt,cnt+2),16);

**int** jp2=Integer.*parseInt*(line.substring(cnt+2,cnt+4),16);

**int** jp3=Integer.*parseInt*(line.substring(cnt+4,line.length()),16);

**if**(jp3<=65)

{

cnt+=2;

}

**else**

{

cnt+=6;

}

}

}

}

}

//수정 레코드

//mod 토큰에 이름과 주소를 저장 주소의 경우에는 절대주소로 고치기 위해 절대주소로 고쳐 저장

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

{

**if**(*sub\_cnt*>=2)

{

*mod*[*mod\_cnt*].addr=Integer.*parseInt*(line.substring(1,7),16)

+Integer.*parseInt*(*r\_start*.get(*sub\_cnt*-2),16);

}

**else**

{

*mod*[*mod\_cnt*].addr=Integer.*parseInt*(line.substring(1,7),16);

}

*f\_mod*.add(Integer.*parseInt*(line.substring(1,7),16));

*mod*[*mod\_cnt*].mod\_n=Integer.*parseInt*(line.substring(7,9),16);

//연산이 있을 경우

**if** (*mod\_cnt* >= 1)

{

**if** ((*mod*[*mod\_cnt*].addr == *mod*[*mod\_cnt* - 1].addr) && line.substring(9, 10).equals("+"))

{

*mod*[*mod\_cnt*].is\_op = 1;

}

**if** ((*mod*[*mod\_cnt*].addr == *mod*[*mod\_cnt* - 1].addr) && line.substring(9, 10).equals("-"))

{

*mod*[*mod\_cnt*].is\_op = 2;

}

}

*mod*[*mod\_cnt*++].name=line.substring(10,line.length());

}

//E레코드일 경우

**else** **if**(line.charAt(0)=='E')

{

rMgr.r\_cnt[*sub\_cnt*-1]=*inst*.size()+1;

}

}

//서브루틴이나 메인프로그램이 끝난 경우 mod를 통해 modification 수행

**int** m\_cnt=0;

String m=**null**;

**for**(**int** i=0; i<*mod\_cnt*; i++)

{

**int** op1=0;

**int** op2=0;

**if**(*mod*[i].mod\_n==5) //5자리를 수정해야 할 경우

{

**for**(**int** j=0; j<*def\_cnt*;j++)

{

**if**(*def*[j].name.contains(*mod*[i].name))

{

//연산이 있을 경우 연산을 수행하여 저장

**if**(*mod*[i].is\_op==1) //+일 경우

{

**for**(**int** k=0; k<*def\_cnt*; k++)

{

**if**(*def*[k].name.contains(*mod*[i-1].name))

{

op1=*def*[k].addr;

}

}

op2=*def*[j].addr;

m=String.*format*("%04X",op1+op2);

}

**else** **if**(*mod*[i].is\_op==2) //-일 경우

{

**for**(**int** k=0; k<*def\_cnt*; k++)

{

**if**(*def*[k].name.contains(*mod*[i-1].name))

{

op1=*def*[k].addr;

}

}

op2=*def*[j].addr;

m=String.*format*("%04X",op1-op2); //수정할 주소에 4자리를 수정

}

**else** //연산이 아닐 경우 4자리를 수정

{

m=String.*format*("%04X",*def*[j].addr);

}

**break**;

}

}

**for**(**int** k=0; k<*name*.size();k++)

{

**if**(*name*.get(k).contains(*mod*[i].name))

{

m=*r\_start*.get(k-1).substring(2,*r\_start*.get(k-1).length());

//m=start.get(k).substring(2,start.get(k).length());

**break**;

}

}

//절대주소를 계산 하여 레지스터값을 변경

**int** mor=(**int**)((*f\_mod*.get(i)+2)/3);

**if**(*inst\_name*.get(mor-1).equals("JSUB"))

{

rmgr.setRe(*inst\_name*.get(mor-1), Integer.*parseInt*(m,16), mor-1);

}

**else** **if**(*inst\_name*.get(mor-1).equals("STCH"))

{

rmgr.setRe(*inst\_name*.get(mor-1), Integer.*parseInt*(m,16), mor-1);

}

**else** **if**(*inst\_name*.get(mor-1).equals("STX"))

{

rmgr.setRe(*inst\_name*.get(mor-1), Integer.*parseInt*(m,16), mor-1);

}

**else** **if**(*inst\_name*.get(mor-1).equals("LDT"))

{

rmgr.setRe(*inst\_name*.get(mor-1), Integer.*parseInt*(m,16), mor-1);

}

**else** **if**(*inst\_name*.get(mor-1).equals("LDCH"))

{

rmgr.setRe(*inst\_name*.get(mor-1), Integer.*parseInt*(m,16), mor-1);

}

by=*byteorder*(m, 4);

rMgr.setMemory(*mod*[i].addr+1, by, 2);

}

**else** //6자리를 수정해야 할 경우 위와 방식은 동일 6자리로 수정만 다름

{

**for**(**int** j=0; j<*def\_cnt*;j++)

{

**if**(*def*[j].name.contains(*mod*[i].name))

{

**if**(*mod*[i].is\_op==1)

{

**for**(**int** k=0; k<*def\_cnt*; k++)

{

**if**(*def*[k].name.contains(*mod*[i-1].name))

{

op1=*def*[k].addr;

}

}

op2=*def*[j].addr;

m=String.*format*("%0X",op1+op2);

}

**else** **if**(*mod*[i].is\_op==2)

{

**for**(**int** k=0; k<*def\_cnt*; k++)

{

**if**(*def*[k].name.contains(*mod*[i-1].name))

{

op1=*def*[k].addr;

}

}

op2=*def*[j].addr;

m=String.*format*("%06X",op1-op2);

}

**else**

{

m=String.*format*("%06X",*def*[j].addr);

}

**break**;

}

}

**for**(**int** k=0; k<*name*.size();k++)

{

**if**(*name*.get(k).contains(*mod*[i].name))

{

m=*r\_start*.get(k-1);

//m=start.get(k);

**break**;

}

}

by=*byteorder*(m, 6);

rMgr.setMemory(*mod*[i].addr, by, 3);

}

}

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

//메모리와 레지스터를 초기화 하는 함수

@Override

**public** **void** initialize(File objFile, ResourceManager rMgr)

{

rMgr.initializeMemory();

rMgr.initializeRegister();

}

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

//보여주고, 다음 명령어를 포인팅한다.

//실질적인 동작을 수행하는 메소드

@Override

**public** **void** oneStep(**int** cnt,**int** step) {

//디바이스를 불러오는 경우

**if**(*inst\_name*.size()>step)

{

**if** (*inst\_name*.get(step).equals("TD"))

{

**for** (**int** i = 0; i < *dev\_cnt*; i++)

{

**if** (*dev*[i].sub\_cnt == cnt)

{

rmgr.initialDevice(*dev*[i].name);

}

}

} **else** **if** (*inst\_name*.get(step).equals("RD"))

{

**for** (**int** i = 0; i < *dev\_cnt*; i++)

{

**if** (*dev*[i].sub\_cnt == cnt)

{

*r\_byte* = rmgr.readDevice(*dev*[i].name, rmgr.reg\_x[cnt]);

rmgr.initialDevice(*dev*[i].name);

}

}

}

**else** **if**(*inst\_name*.get(step).equals("WD"))

{

**byte** [] data = **new** **byte**[10];

**for**(**int** i=0; i<*dev\_cnt*; i++)

{

**if**(*dev*[i].sub\_cnt==cnt)

{

rmgr.initialDevice(*dev*[i].name);

rmgr.writeDevice(*dev*[i].name, data, rmgr.reg\_x[cnt]);

**break**;

}

}

}

//디바이스를 사용안하는 경우

**else**

{

rmgr.initialDevice(**null**);

}

}

rmgr.setName(*name*.get(cnt-1));

}

//남은 명령어를 모두 수행하는 메소드.

//목적 코드를 모두 수행하고 난 값의 변화를 보여준다.

@Override

**public** **void** allStep () {

rmgr.initialDevice(**null**);

}

//op코드를 받아 명령어를 찾아주는 함수(ni값도 고려하여)

//반환 값 : 명령어의 index

**public** **static** **int** search\_opcode(**int** op)

{

**int** k = -1;

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

{

**if** ((op - 3) == inst\_table[i].opcode)

{

**return** i;

} **else** **if** ((op - 2) == inst\_table[i].opcode)

{

**return** i;

} **else** **if** ((op - 1) == inst\_table[i].opcode)

{

**return** i;

} **else** **if** (op == inst\_table[i].opcode)

{

**return** i;

}

}

**return** k;

}

//문자열을 길이에 맞게 바이트로 parsing해주는 함수

//반환값 : parsing된 바이트 배열

//매개변수 : str=파싱할 문자열, len=파싱할 문자열의 길이

**public** **static** **byte**[] byteorder(String str,**int** len)

{

**int** s1;

**int** s2;

**int** s3;

**int** s4;

**if**(len==6) //일반 명령어일 경우

{

**byte**[] by = **new** **byte**[3];

s1=Integer.*parseInt*(str.substring(0,2),16);

s2=Integer.*parseInt*(str.substring(2,4),16);

s3=Integer.*parseInt*(str.substring(4,str.length()),16);

by[0]=(**byte**)s1;

by[1]=(**byte**)s2;

by[2]=(**byte**)s3;

**return** by;

}

**else** **if**(len==4) //2형식이거나 길이가 2일 경우

{

**byte**[] by3 = **new** **byte**[2];

s1=Integer.*parseInt*(str.substring(0,2),16);

s2=Integer.*parseInt*(str.substring(2,str.length()),16);

by3[0]=(**byte**)s1;

by3[1]=(**byte**)s2;

**return** by3;

}

//4형식일 경우

**else** **if**(len==8)

{

**byte**[] by4 = **new** **byte**[4];

s1=Integer.*parseInt*(str.substring(0,2),16);

s2=Integer.*parseInt*(str.substring(2,4),16);

s3=Integer.*parseInt*(str.substring(4,6),16);

s4=Integer.*parseInt*(str.substring(6,str.length()),16);

by4[0]=(**byte**)s1;

by4[1]=(**byte**)s2;

by4[2]=(**byte**)s3;

by4[3]=(**byte**)s4;

**return** by4;

}

//1바이트일 경우

**else**

{

**byte**[] by2 = **new** **byte**[1];

s1=Integer.*parseInt*(str.substring(0,str.length()),16);

by2[0]=(**byte**)s1;

**return** by2;

}

}

}

SicLoader.interface

**package** project;

**import** java.io.File;

**public** **interface** SicLoader **extends** SicSimulator {

//목적코드를 읽어 메모리에 로드한다.

//목적코드의 각 헤더(H, T, M 등)를 읽어 동작을 수행한다.

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

}

Token\_unit

**package** project;

//

//참조값, 정의값, 수정값 등의 이름,형식 등을 저장하는 테이블

//

**public** **class** Token\_unit {

**public** String name; //이름

**public** **int** addr; //주소

**public** **int** sub\_cnt; //루틴 index

**public** **int** mod\_n; //수정갯수

**public** **int** is\_op; //연산 /+ : 1 /- : 2 /없음 : 0

Token\_unit(String str,**int** f)

{

**this**.name=str;

**this**.addr=f;

**this**.sub\_cnt=f;

**this**.mod\_n=f;

**this**.is\_op=f;

}

}

SicSimulator.interface

**package** project;

**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 rMgr);

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

//보여주고, 다음 명령어를 포인팅한다.

//실질적인 동작을 수행하는 메소드

**public** **void** oneStep(**int** cnt,**int** step);

//남은 명령어를 모두 수행하는 메소드.

//목적 코드를 모두 수행하고 난 값의 변화를 보여준다.

//실질적인 동작을 수행하는 메소드

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

}

Instruction

**package** project;

//

//instruction 목록 파일로 부터 정보를 받아와서 생성하는 클래스

//

**public** **class** instruction {

**public** String name; //명령어의 이름

**public** **int** opcode; //명령어의 opcode

**public** **int** format; //명령어의 형식

instruction(String n, **int** op,**int** form)

{

**this**.name=n;

**this**.opcode=op;

**this**.format=form;

}

//명령어의 이름을 지정하는 함수

**public** **void** setName(String n)

{

**this**.name=n;

}

//명령어의 형식을 지정하는 함수

**public** **void** setFormat(**int** f)

{

**this**.format=f;

}

//명령어의 형식을 가져오는 함수

**public** **int** getFormat()

{

**return** format;

}

}