숭실대학교 Soongsil University

시스템프로그래밍 2021 보고서

보고서 제출서약서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
   1.1. 나는 동료의 보고서를 베끼지 않았습니다.
   1.2. 나는 비공식적으로 얻은 해답/해설을 기초로 보고서를 작성하지 않았습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다. (나는 특히 인터넷에서 다운로드한 내용을 보고서에 거의 그대로 복사하여 사용하지 않았습니다.)
3. 나는 보고서를 제출하기 전에 동료에게 보여주지 않았습니다.
4. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

과목	시스템프로그래밍 2021
과제명	SIC/XE 어셈블러 구현 (Project #2)
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20163340 강원경 (출석번호 107번)
제출일	2021년 06월 03일

차 례

1장 프로젝트 동기/목적 ---------- 3p

2장 설계/구현 아이디어 ---------- 3p

3장 수행결과(구현 화면 포함) ---------- 5p

4장 결론 및 보충할 점 ---------- 6p

5장 디버깅 ---------- 7p

6장 소스코드(+주석) ---------- 8p

1장. 프로젝트 동기/목적

이번 프로젝트는 우리가 2021년 1학기 '시스템 프로그래밍' 시간에 배운 내용을 토대로 만들어진 Object Program을 해석하여 실행시키는 시뮬레이터를 만드는 것이다. 만들어진 Object Program을 실행시키기 위해서는 로더와 링커의 개념이 필요했고, 또 현재 실행중인 단계의 정보를 보여주기 위해 이를 시각화하여 보여주기 위한 모듈도 필요했다.

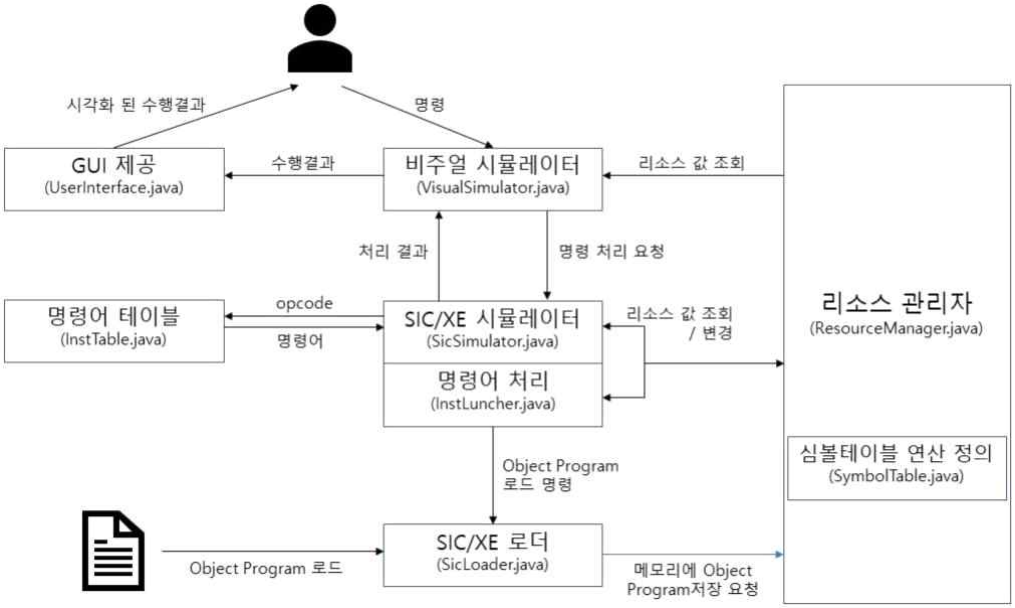
기존에 진행하였던 프로젝트들과 이번 프로젝트에서 가장 다른 점이 이 부분이라고 생각한다. 이전의 프로젝트들은 명령어에 연산을 수행하고 이를 취합하여 하나의 결과 파일로 출력하는 것이 전부였는데, 이번 프로젝트의 경우에는 시각화를 통해서도 중간 단계를 보여줘야 하고, 그 연산들이 모두 수행됨으로써 하나의 최종 결과를 보여줘야 하는데, 이 부분에서 따라오는 추가적인 작업들, 예를 들면 버튼을 눌렀을 때의 이벤트를 처리하고 각 버튼에 따라 어떤 연산이 처리되는지에 대한 부분들과 그 이벤트를 어떤 모듈이 어떻게 처리해야 하는지에 대한 설계가 필요해 보였다.

이번 프로젝트를 통해 사용자를 위한 인터페이스를 설계하고 이와 관련된 모듈들을 이용함으로써 사용자와 프로그램 간에 상호작용을 극대화하는 방법에 대해 이해하고 또한 로더와 링커의 개념을 이용해 Control Section이 나누어져있는 프로그램을 Linking하여 자연스러운 흐름으로 프로그램이 실행되어질 수 있도록 설계함으로써 이번 과목을 통해 배운 내용을 자세히 이해하는 것이 이번 프로젝트의 목표이다.

이번 프로젝트는 64비트의 Windows 10 Pro(20H2) 운영체제에서 jdk 1.8, Eclipse 버전은 2021.3, 문자 인코딩은 UTF-8을 이용하여 진행하였다.

2장. 설계/구현 아이디어

이번 프로젝트 결과물의 전체적인 흐름도는 다음과 같다.



프로젝트의 기본 명세를 기반으로 작성하되, 비주얼 시뮬레이터에서 GUI를 제공하는 모듈을 분리하여 UserInterface라는 모듈에서 구현하였고, opcode에서 명령어 이름을 효율적으로 추출하기 위해 기존 프로젝트에서 구현하였던 명령어 테이블의 모듈을 약간의 customizing을 통해 사용하였다.

각각의 모듈에 대한 대략적인 설명은 다음과 같다.

- VisualSimulator : 이 프로그램의 메인 루틴이 담겨있는 함수로, 사용자가 GUI의 버튼을 통해 연산의 수행을 명령하면 해당 버튼에 맞는 명령을 수행할 수 있도록 각 모듈에 이벤트를 전달한다. 각 실행 과정에서의 연산 중간 과정을 GUI에 표시할 수 있도록 ResourceManager에 있는 리소스 값을 참조하고 SicSimualtor에서 전달받은 값을 종합하여 시각화할 수 있도록 GUI를 담당하는 모듈에 전달한다.

- UserInterface : 사용자에게 제공될 GUI를 담당하는 모듈이다. 화면에 어떤 모습으로 보일지, 어떤 요소가 창에서의 어느 좌표에 표시되도록 할 것인지에 대한 내용을 구현하고 있으며, 사용자가 명령은 비주얼 시뮬레이터에게 하지만, 그 명령을 입력받는 매개는 UserInterface이다. 즉, VisualSimulator와 사용자 간에 중간다리 역할을 수행한다.

- SicSimulator : VisualSimulator에서 받은 사용자의 명령을 실질적으로 수행하는 모듈이다. 각 명령어에 따른 연산을 수행하는 InstLuncher를 가지고 있고, ResourceManager에 직접 접근하여 리소스의 값을 가져오거나 새로운 값을 할당/변경하는 등의 역할을 수행한다. 수행한 결과는 VisualSimulator와 ResourceManager에 전달한다.

- InstTable : SIC/XE 머신의 명령어 정보가 모두 저장된 테이블이다. 명령어의 opcode를 입력하면 그 코드에 해당하는 명령어 이름 또는 명령어의 형식을 반환해주는 역할을 수행한다. 명령어 코드에 따라 SicSimulator에서 이름과 형식을 직접 지정하는 방법보다 새로운 모듈 내에 함수들을 정의하여 구현하는것이 더 효율적이고 직관적인 코드 작성이 가능할 것으로 판단해 구현한 모듈이다.

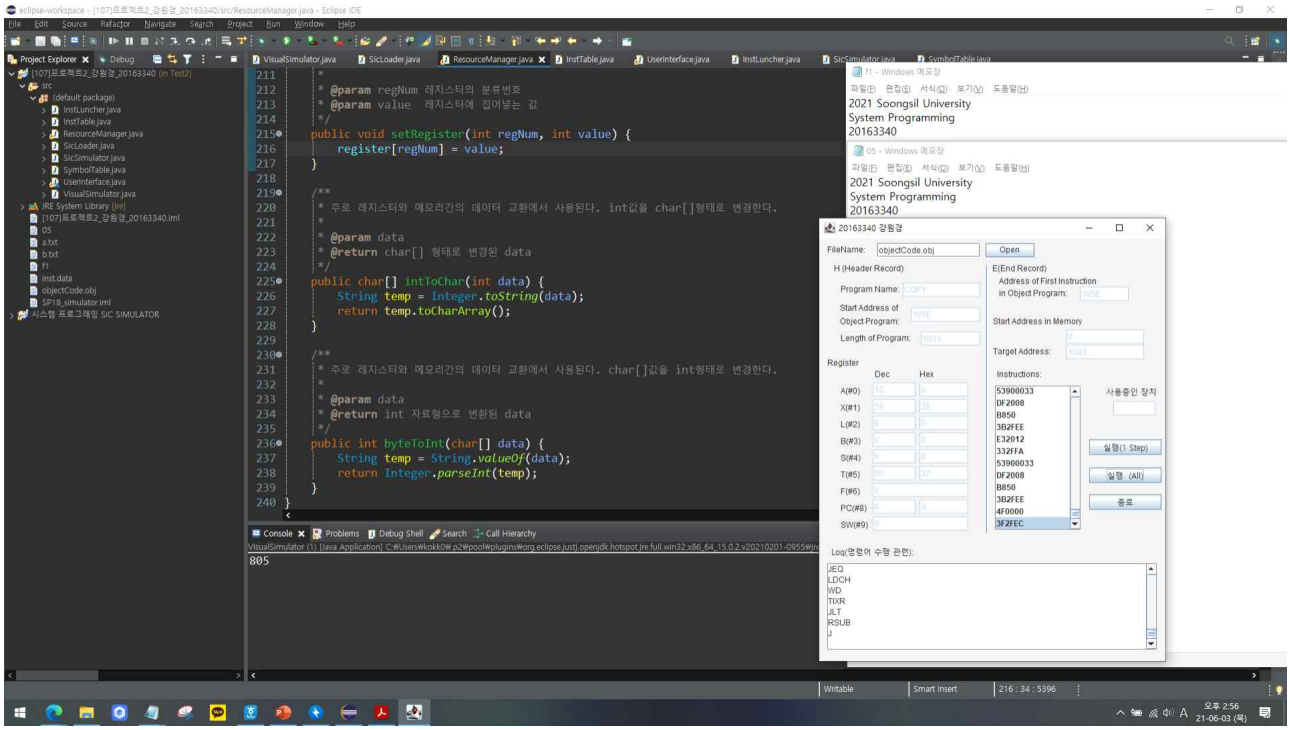
- InstLuncher : 각 명령어에 따른 실제 연산을 수행하는 모듈이다. 명령어에 따라 수행할 동작을 정의하고 있으며, SicSimulator로부터 연산에 필요한 리소스나 정보들을 전달받아 정의된 함수를 통해 연산을 수행하는 역할을 담당한다. 연산 수행 후의 결과값 (리소스 값 등)은 SicSimulator에게 전달하여 ResourceManager와 VisualSimulator에 반영될 수 있도록 한다.

- ResourceManager : 가상의 SIC/XE 머신으로, 메모리, 레지스터, 장치와 같은 리소스들을 관리하고 있는 모듈이다.

- SymbolTable : Symbol과 관련된 데이터 및 이와 관련된 함수를 정의하고 있는 모듈이다. 주로 D레코드와 관련된 심볼들을 다루는데 사용된다.

- SicLoader : SIC/XE 머신에서 로더와 같은 역할을 수행하는 모듈로, Object Program을 받아 메모리 (이 프로그램에서는 Resource Manager의 메모리 영역)에 올려주는 역할을 수행한다. 또, pass1과 pass2로 나누어 연산을 수행하는데, pass1에서는 프로그램의 이름과 길이, 그리고 각 프로그램에 정의된 EXTDEF(D레코드)의 심볼을 저장하고, pass2에서는 T 레코드인 경우 한 줄씩 프로그램 코드를 ResourceManager의 메모리 영역에 올리고, M 레코드인 경우 해당 Symbol의 주소값을 변경하는 연산을 수행한다.

3장. 수행 결과

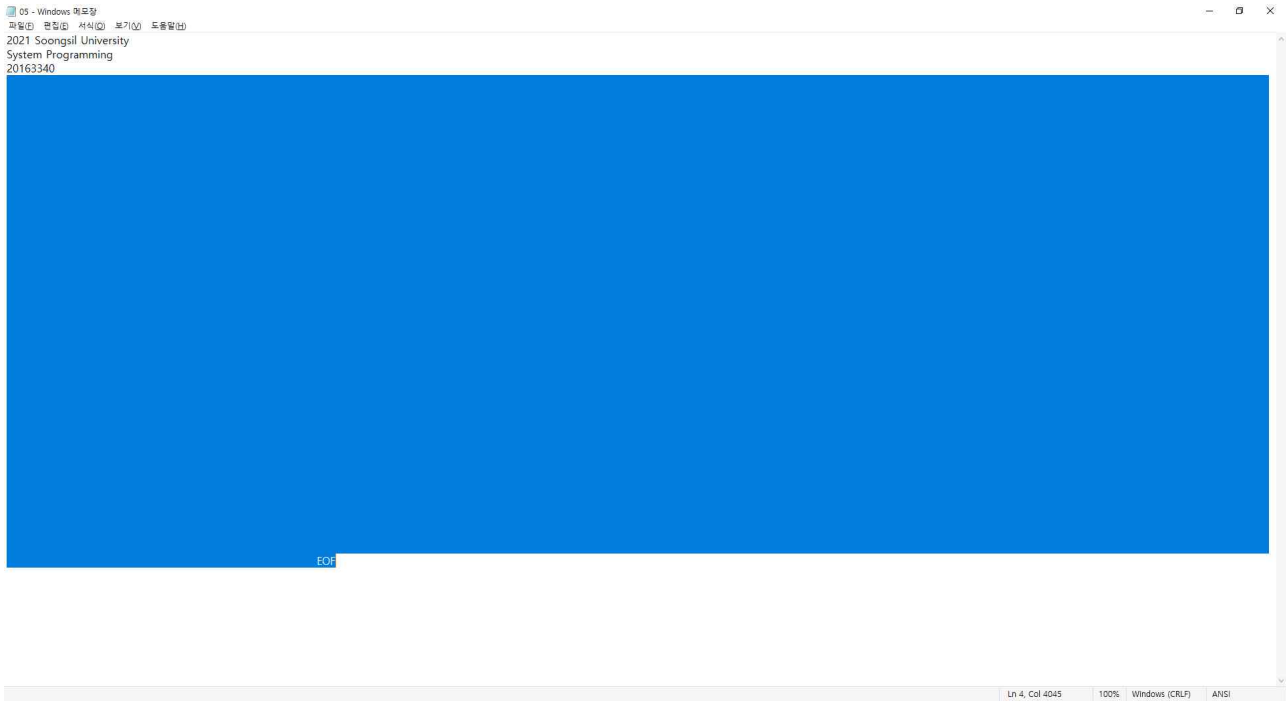


코드를 실행하면 위의 그림과 같이 f1에 저장된 파일의 내용이 05에 그대로 복사되어졌음을 알 수 있다. 또, 실행 (All) 버튼을 눌러 프로그램을 실행할 경우, 명령어가 몇 번 실행되어있는지 콘솔창에 출력하도록 설계하였다.

4장. 결론 및 보충할 점.

이번 프로젝트 진행을 통해 여러 가지 객체들을 유기적으로 결합시키는 프로그래밍을 하는 것은 물론, Swing과 awt를 이용한 GUI와 이벤트 처리를 통해 하나의 시각화된 프로그램을 만들어내는 목표를 달성하였다. 또, 프로그램을 설계하면서 SIC/XE 머신의 링커와 로더의 개념을 활용하여 수업시간에 배웠던 Control Section이 나눠진 오브젝트 프로그램을 실행시키는 시뮬레이터를 설계하는데 적용해보고 이를 이해하는 이론적인 목표도 달성하였다.

하지만, 이번 프로젝트에서 하나의 시뮬레이터를 구현하는데에 있어 설계하는 부분에서 크게 막힌 부분이 하나 있었고, 이 때문에 프로젝트의 결과물이 본인이 생각했던거와는 다소 다르게 완성이 된 부분이 있어 아쉬움이 남는다.



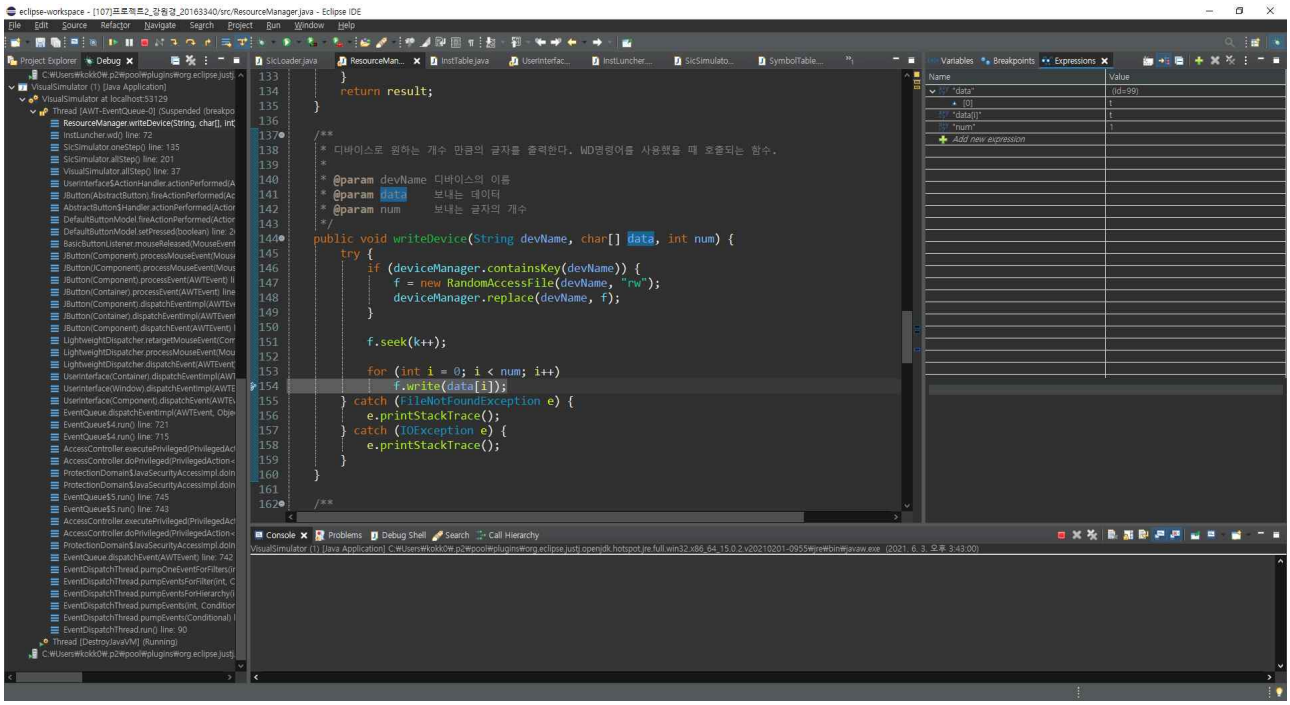
프로젝트에서 완성된 파일을 살펴보면 중간 부분이 텅 비어있는 상태로 한참 뒤에 EOF가 기록되는 것을 볼 수 있다.

이 부분은 본인이 디버깅 모드를 통해 살펴보고자 하였으나, 어떻게 고쳐야 하는지에 대한 아이디어가 없어 결국 고치지 못하고 제출하였다.

5장. 디버깅

이번 프로젝트는 Eclipse IDE에서 제공하는 디버거를 이용하기도 했으나, 거의 처음 다뤄보다시피 하는 Swing 에서의 디버깅은 어떻게 해야할지 잘 모르겠어서 콘솔창을 이용해 디버깅을 진행하기도 하였다.

콘솔창을 이용한 디버깅은 생략하고, 이 보고서에서는 Eclipse IDE에서 제공하는 디버거를 이용한 사례 중 위의 개선 사항에서 언급했었던 파일 쓰기의 부분에서 디버거를 사용했던 부분을 다뤄보고자 한다.



위 사진은 파일의 쓰기가 실질적으로 수행되는 ResourceManager 클래스의 writeDevice 함수에서의 디버거 사용 과정이다.

오른쪽에서 Expressions를 통해 본인이 탐색하려는 변수에 대해 지정하고, 그 내용을 살펴봄으로써 현재 수행단계에서 어떤 변수에 어떤 값이 기록되어있는지 살펴볼 수 있는데,

이 과정에서도 본인이 탐색하려고 했던 내용, 즉 어떤 시점에서 파일에 EOF가 기록되는지, 또 파일에 왜 빈 공간이 생기게 되는지에 대해 탐색해보고 싶었으나, 디버거를 통한 변수 탐색만으로는 이 과정을 찾아낼 수 없었고, 결국 수정하지 못한 채 프로젝트를 마무리하였다.

6장. 소스코드(+주석)

VisualSimulator.java

```
import javax.swing.*;
import java.io.File;

/**
 * VisualSimulator는 사용자와의 상호작용을 담당한다. 즉, 버튼 클릭등의 이벤트를 전달하고
그에 따른 결과값을 화면에 업데이트
 * 하는 역할을 수행한다.
 * 실제적인 작업은 SicSimulator에서 수행하도록 구현한다.
 */
public class VisualSimulator {
    ResourceManager resourceManager = new ResourceManager();
    SicLoader sicLoader = new SicLoader(resourceManager);
    SicSimulator sicSimulator = new SicSimulator(resourceManager);
    UserInterface frame = new UserInterface(this);

    /**
     * 프로그램 로드 명령을 전달한다.
     */
    public void load(File program) {
        sicLoader.load(program);
        sicSimulator.load(program, sicLoader, this);
        update();
    };

    /**
     * 하나의 명령어만 수행할 것을 SicSimulator에 요청한다.
     */
    public void oneStep() {
        sicSimulator.oneStep();
        update();
    };

    /**
     * 남아있는 모든 명령어를 수행할 것을 SicSimulator에 요청한다.
     */
    public void allStep() {
        sicSimulator.allStep();
        update();
    };

    /**
     * 화면을 최신값으로 갱신하는 역할을 수행한다.
     */
    public void update() {
        frame.deviceTF.setText(resourceManager.curDevice); // 사용중인 장치
        frame.programNameTF.setText(resourceManager.programName); // 프로그램 이름

frame.lengthTF.setText(Integer.toHexString(resourceManager.programLength).toUpperCase()); //
프로그램 길이
        frame.firstInstTF.setText(Integer.toHexString(resourceManager.firstInst).toUpperCase()); //
시작주소

frame.opStartAddrTF.setText(Integer.toHexString(resourceManager.opStartAddr).toUpperCase());
        frame.taTF.setText(Integer.toHexString(resourceManager.targetAddr)); // TargetAddress
        frame.instList.setListData(resourceManager.instList); // Instructions
        frame.LogTA.setText(resourceManager.logText); // 프로그램 하단 로그

frame.memStartAddrTF.setText(Integer.toHexString(resourceManager.memStartAddr).toUpperCas
e()); // 시작 주소

        //화면에 보이는 레지스터 값 갱신
        for (int i = 0; i < resourceManager.REGISTER_NUMBER; i++) {
            if (i == 0) {
                frame.decATF.setText(Integer.toString(resourceManager.getRegister(i)));
                frame.hexATF.setText(Integer.toHexString(resourceManager.getRegister(i)));
            } else if (i == 1) {
                frame.decXTF.setText(Integer.toString(resourceManager.getRegister(i)));
                frame.hexXTF.setText(Integer.toHexString(resourceManager.getRegister(i)));
            } else if (i == 2) {
                frame.decLTF.setText(Integer.toString(resourceManager.getRegister(i)));
                frame.hexLTF.setText(Integer.toHexString(resourceManager.getRegister(i)));
            } else if (i == 3) {
                frame.decBTF.setText(Integer.toString(resourceManager.getRegister(i)));
                frame.hexBTF.setText(Integer.toHexString(resourceManager.getRegister(i)));
            } else if (i == 4) {
                frame.decSTF.setText(Integer.toString(resourceManager.getRegister(i)));
                frame.hexSTF.setText(Integer.toHexString(resourceManager.getRegister(i)));
            } else if (i == 5) {
                frame.decTTF.setText(Integer.toString(resourceManager.getRegister(i)));
                frame.hexTTF.setText(Integer.toHexString(resourceManager.getRegister(i)));
            } else if (i == 6) {
                frame.hexFTF.setText(Integer.toHexString(resourceManager.getRegister(i)));
            } else if (i == 8) {
                frame.decPCTF.setText(Integer.toString(resourceManager.getRegister(i)));
                frame.hexPCTF.setText(Integer.toHexString(resourceManager.getRegister(i)));
            } else if (i == 9) {
                frame.hexSWTF.setText(Integer.toHexString(resourceManager.getRegister(i)));
            }
        }
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                VisualSimulator visualSimulator = new VisualSimulator();
                visualSimulator.frame.setVisible(true);
            }
        });
    }
`
```

UserInterface.java

```
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.filechooser.FileNameExtensionFilter;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.ArrayList;

class UserInterface extends JFrame {
    VisualSimulator vs;
    JTextArea LogTA;
    JPanel contentPane;
    JTextField fileTF;
    JTextField programNameTF;
    JTextField opStartAddrTF;
    JTextField lengthTF;
    JTextField decATF;
    JTextField hexATF;
    JTextField decXTF;
    JTextField hexXTF;
    JTextField decLTF;
    JTextField hexLTF;
    JTextField decPCTF;
    JTextField hexPCTF;
    JTextField hexSWTF;
    JTextField decBTF;
    JTextField hexBTF;
    JTextField decSTF;
    JTextField hexSTF;
    JTextField decTTF;
    JTextField hexTTF;
    JTextField hexFTF;
    JTextField firstInstTF;
    JTextField memStartAddrTF;
    JTextField taTF;
    JTextField deviceTF;

    JButton openBtn;
    JButton oneStepBtn;
    JButton allStepBtn;
    JButton exitBtn;

    JList<String> instList;

    /**
     * Create the frame.
     */
    public UserInterface(VisualSimulator _vs) {
        vs = _vs;
        setTitle("20163340 강원경");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 534, 670);

        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JSeparator separator = new JSeparator();
        separator.setOrientation(SwingConstants.VERTICAL);
        separator.setBounds(250, 40, 11, 400);
        contentPane.add(separator);

        JLabel lblNewLabel = new JLabel("FileName:");
        lblNewLabel.setFont(new Font("Arial", Font.PLAIN, 12));
        lblNewLabel.setBounds(12, 10, 63, 15);
        contentPane.add(lblNewLabel);

        fileTF = new JTextField();
        fileTF.setBounds(87, 7, 154, 21);
        contentPane.add(fileTF);
        fileTF.setColumns(10);

        openBtn = new JButton("Open");
        openBtn.addActionListener(new ActionHandler());
        openBtn.setFont(new Font("Arial", Font.PLAIN, 12));
        openBtn.setBounds(249, 7, 73, 21);
        contentPane.add(openBtn);

        JLabel lblH = new JLabel("H (Header Record)");
        lblH.setFont(new Font("Arial", Font.PLAIN, 12));
        lblH.setBounds(22, 38, 118, 15);
        contentPane.add(lblH);

        JLabel lblProgramName = new JLabel("Program Name:");
        lblProgramName.setFont(new Font("Arial", Font.PLAIN, 12));
        lblProgramName.setBounds(32, 69, 97, 15);
        contentPane.add(lblProgramName);

        JLabel lblStartAddressOf = new JLabel("Start Address of");
        lblStartAddressOf.setFont(new Font("Arial", Font.PLAIN, 12));
        lblStartAddressOf.setBounds(31, 97, 98, 15);
        contentPane.add(lblStartAddressOf);

        JLabel lblOb = new JLabel("Object Program:");
        lblOb.setFont(new Font("Arial", Font.PLAIN, 12));
        lblOb.setBounds(31, 117, 98, 15);
        contentPane.add(lblOb);

        JLabel lblLeng = new JLabel("Length of Program:");
        lblLeng.setFont(new Font("Arial", Font.PLAIN, 12));
        lblLeng.setBounds(32, 142, 108, 15);
        contentPane.add(lblLeng);
```

```
programNameTF = new JTextField();
programNameTF.setBounds(125, 66, 116, 21);
programNameTF.setEnabled(false);
contentPane.add(programNameTF);
programNameTF.setColumns(10);

opStartAddrTF = new JTextField();
opStartAddrTF.setBounds(137, 103, 104, 21);
opStartAddrTF.setEnabled(false);
contentPane.add(opStartAddrTF);
opStartAddrTF.setColumns(10);

lengthTF = new JTextField();
lengthTF.setBounds(152, 139, 89, 21);
lengthTF.setEnabled(false);
contentPane.add(lengthTF);
lengthTF.setColumns(10);

JLabel label = new JLabel("Register");
label.setFont(new Font("Arial", Font.PLAIN, 12));
label.setBounds(12, 179, 57, 15);
contentPane.add(label);

JLabel lblDec = new JLabel("Dec");
lblDec.setFont(new Font("Arial", Font.PLAIN, 12));
lblDec.setBounds(83, 196, 57, 15);
contentPane.add(lblDec);

JLabel lblHex = new JLabel("Hex");
lblHex.setFont(new Font("Arial", Font.PLAIN, 12));
lblHex.setBounds(150, 196, 57, 15);
contentPane.add(lblHex);

JLabel lblA = new JLabel("A(#0)");
lblA.setFont(new Font("Arial", Font.PLAIN, 12));
lblA.setBounds(32, 221, 57, 15);
contentPane.add(lblA);

decATF = new JTextField();
decATF.setBounds(80, 215, 64, 21);
decATF.setEnabled(false);
contentPane.add(decATF);
decATF.setColumns(10);

hexATF = new JTextField();
hexATF.setBounds(150, 215, 73, 21);
hexATF.setEnabled(false);
contentPane.add(hexATF);
hexATF.setColumns(10);

JLabel lblX = new JLabel("X(#1)");
lblX.setFont(new Font("Arial", Font.PLAIN, 12));
lblX.setBounds(32, 246, 57, 15);
contentPane.add(lblX);

decXTF = new JTextField();
decXTF.setColumns(10);
decXTF.setEnabled(false);
decXTF.setBounds(80, 240, 64, 21);
contentPane.add(decXTF);

hexXTF = new JTextField();
hexXTF.setColumns(10);
hexXTF.setEnabled(false);
hexXTF.setBounds(150, 240, 73, 21);
contentPane.add(hexXTF);

JLabel lblL = new JLabel("L(#2)");
lblL.setFont(new Font("Arial", Font.PLAIN, 12));
lblL.setBounds(32, 271, 57, 15);
contentPane.add(lblL);

decLTF = new JTextField();
decLTF.setColumns(10);
decLTF.setEnabled(false);
decLTF.setBounds(80, 265, 64, 21);
contentPane.add(decLTF);

hexLTF = new JTextField();
hexLTF.setColumns(10);
hexLTF.setEnabled(false);
hexLTF.setBounds(150, 265, 73, 21);
contentPane.add(hexLTF);

JLabel lblB = new JLabel("B(#3)");
lblB.setFont(new Font("Arial", Font.PLAIN, 12));
lblB.setBounds(32, 296, 57, 15);
contentPane.add(lblB);

decBTF = new JTextField();
decBTF.setColumns(10);
decBTF.setEnabled(false);
decBTF.setBounds(80, 290, 64, 21);
contentPane.add(decBTF);

hexBTF = new JTextField();
hexBTF.setColumns(10);
hexBTF.setEnabled(false);
hexBTF.setBounds(150, 290, 73, 21);
contentPane.add(hexBTF);

JLabel lblS = new JLabel("S(#4)");
lblS.setFont(new Font("Arial", Font.PLAIN, 12));
lblS.setBounds(32, 321, 57, 15);
contentPane.add(lblS);

decSTF = new JTextField();
decSTF.setColumns(10);
decSTF.setEnabled(false);
decSTF.setBounds(80, 315, 64, 21);
contentPane.add(decSTF);
```

```
hexSTF = new JTextField();
hexSTF.setColumns(10);
hexSTF.setEnabled(false);
hexSTF.setBounds(150, 315, 73, 21);
contentPane.add(hexSTF);

JLabel lblT = new JLabel("T(#5)");
lblT.setFont(new Font("Arial", Font.PLAIN, 12));
lblT.setBounds(32, 346, 57, 15);
contentPane.add(lblT);

decTTF = new JTextField();
decTTF.setColumns(10);
decTTF.setEnabled(false);
decTTF.setBounds(80, 340, 64, 21);
contentPane.add(decTTF);

hexTTF = new JTextField();
hexTTF.setColumns(10);
hexTTF.setEnabled(false);
hexTTF.setBounds(150, 340, 73, 21);
contentPane.add(hexTTF);

JLabel lblF = new JLabel("F(#6)");
lblF.setFont(new Font("Arial", Font.PLAIN, 12));
lblF.setBounds(32, 371, 57, 15);
contentPane.add(lblF);

hexFTF = new JTextField();
hexFTF.setColumns(10);
hexFTF.setEnabled(false);
hexFTF.setBounds(80, 365, 143, 21);
contentPane.add(hexFTF);

JLabel lblPC = new JLabel("PC(#8)");
lblPC.setFont(new Font("Arial", Font.PLAIN, 12));
lblPC.setBounds(32, 396, 57, 15);
contentPane.add(lblPC);

decPCTF = new JTextField();
decPCTF.setColumns(10);
decPCTF.setEnabled(false);
decPCTF.setBounds(80, 390, 64, 21);
contentPane.add(decPCTF);

hexPCTF = new JTextField();
hexPCTF.setColumns(10);
hexPCTF.setEnabled(false);
hexPCTF.setBounds(150, 390, 73, 21);
contentPane.add(hexPCTF);

JLabel lblSW = new JLabel("SW(#9)");
lblSW.setFont(new Font("Arial", Font.PLAIN, 12));
lblSW.setBounds(32, 421, 57, 15);
contentPane.add(lblSW);
```

```
hexSWTF = new JTextField();
hexSWTF.setColumns(10);
hexSWTF.setEnabled(false);
hexSWTF.setBounds(80, 415, 143, 21);
contentPane.add(hexSWTF);

JLabel lblEndRecord = new JLabel("E(End Record)");
lblEndRecord.setFont(new Font("Arial", Font.PLAIN, 12));
lblEndRecord.setBounds(259, 38, 98, 15);
contentPane.add(lblEndRecord);

JLabel lblFA = new JLabel("Address of First Instruction");
lblFA.setFont(new Font("Arial", Font.PLAIN, 12));
lblFA.setBounds(269, 57, 159, 15);
contentPane.add(lblFA);

JLabel lblInObjectProgram = new JLabel("in Object Program:");
lblInObjectProgram.setFont(new Font("Arial", Font.PLAIN, 12));
lblInObjectProgram.setBounds(269, 75, 138, 15);
contentPane.add(lblInObjectProgram);

firstInstTF = new JTextField();
firstInstTF.setBounds(390, 72, 73, 21);
firstInstTF.setEnabled(false);
contentPane.add(firstInstTF);
firstInstTF.setColumns(10);

JLabel lblStartAddressIn = new JLabel("Start Address in Memory");
lblStartAddressIn.setFont(new Font("Arial", Font.PLAIN, 12));
lblStartAddressIn.setBounds(260, 117, 184, 15);
contentPane.add(lblStartAddressIn);

memStartAddrTF = new JTextField();
memStartAddrTF.setBounds(370, 136, 116, 21);
memStartAddrTF.setEnabled(false);
contentPane.add(memStartAddrTF);
memStartAddrTF.setColumns(10);

JLabel lblTargetAddress = new JLabel("Target Address:");
lblTargetAddress.setFont(new Font("Arial", Font.PLAIN, 12));
lblTargetAddress.setBounds(260, 162, 118, 15);
contentPane.add(lblTargetAddress);

taTF = new JTextField();
taTF.setBounds(370, 159, 116, 21);
taTF.setEnabled(false);
contentPane.add(taTF);
taTF.setColumns(10);

JLabel lblInstructions = new JLabel("Instructions:");
lblInstructions.setFont(new Font("Arial", Font.PLAIN, 12));
lblInstructions.setBounds(266, 196, 73, 15);
contentPane.add(lblInstructions);

instList = new JList<String>();
JScrollPane scrollPane2 = new JScrollPane();
```

```
    instList.setBounds(263, 220, 129, 216);
    scrollPane2.add(instList);
    scrollPane2.setBounds(263, 220, 129, 216);
    scrollPane2.setViewportView(instList);
    scrollPane2.setVerticalScrollBarPolicy(scrollPane2.VERTICAL_SCROLLBAR_ALWAYS);
    contentPane.add(scrollPane2);

    deviceTF = new JTextField();
    deviceTF.setBounds(440, 243, 63, 21);
    deviceTF.setEnabled(false);
    contentPane.add(deviceTF);
    deviceTF.setColumns(10);

    JLabel label_1 = new JLabel("사용중인 장치");
    label_1.setFont(new Font("맑은 고딕", Font.PLAIN, 12));
    label_1.setBounds(429, 221, 89, 15);
    contentPane.add(label_1);

    oneStepBtn = new JButton("실행(1 Step)");
    oneStepBtn.setFont(new Font("맑은 고딕", Font.PLAIN, 12));
    oneStepBtn.addActionListener(new ActionHandler());
    oneStepBtn.setBounds(404, 300, 108, 23);
    contentPane.add(oneStepBtn);

    allStepBtn = new JButton("실행 (All)");
    allStepBtn.setFont(new Font("맑은 고딕", Font.PLAIN, 12));
    allStepBtn.setBounds(404, 342, 108, 23);
    allStepBtn.addActionListener(new ActionHandler());
    contentPane.add(allStepBtn);

    exitBtn = new JButton("종료");
    exitBtn.setFont(new Font("맑은 고딕", Font.PLAIN, 12));
    exitBtn.setBounds(404, 382, 108, 23);
    exitBtn.addActionListener(new ActionHandler());
    contentPane.add(exitBtn);

    JLabel lblLog = new JLabel("Log(명령어 수행 관련):");
    lblLog.setFont(new Font("맑은 고딕", Font.PLAIN, 12));
    lblLog.setBounds(18, 460, 143, 15);
    contentPane.add(lblLog);

    JScrollPane scrollPane = new JScrollPane();
    LogTA = new JTextArea();
    LogTA.setBounds(22, 485, 484, 130);
    scrollPane.add(LogTA);
    scrollPane.setBounds(12, 485, 494, 130);
    scrollPane.setViewportView(LogTA);
    scrollPane.setVerticalScrollBarPolicy(scrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    contentPane.add(scrollPane);
}

private class ActionHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        JButton b = (JButton) e.getSource();
        if (b == openBtn) {
            JFileChooser fc = new JFileChooser(".");
            FileNameExtensionFilter filter = new FileNameExtensionFilter("Object File (*.obj)",
"obj");
            fc.setFileFilter(filter);
            int choice = fc.showOpenDialog(null);
            if (choice == JFileChooser.APPROVE_OPTION) {
                File objFile = fc.getSelectedFile();
                fileTF.setText(objFile.getName());
                vs.load(objFile);
            }
        } else if (b == oneStepBtn) {
            vs.oneStep();
            instList.setSelectedIndex(instList.getModel().getSize() - 1);
            instList.ensureIndexIsVisible(instList.getSelectedIndex());
        } else if (b == allStepBtn) {
            vs.allStep();
            instList.setSelectedIndex(instList.getModel().getSize() - 1);
            instList.ensureIndexIsVisible(instList.getSelectedIndex());
            System.out.println(instList.getModel().getSize());
        } else if (b == exitBtn) {
            System.exit(0);
        }
    }
}
```

SICSimulator.java

```java
import java.io.File;

/**
 * 시뮬레이터로서의 작업을 담당한다. VisualSimulator에서 사용자의 요청을 받으면 이에 따라 ResourceManager에 접근하여
 * 작업을 수행한다.
 *
 * 작성중의 유의사항 : 1) 새로운 클래스, 새로운 변수, 새로운 함수 선언은 얼마든지 허용됨. 단, 기존의 변수와 함수들을 삭제하거나
 * 완전히 대체하는 것은 지양할 것. 2) 필요에 따라 예외처리, 인터페이스 또는 상속 사용 또한 허용됨. 3) 모든 void 타입의 리턴값은
 * 유저의 필요에 따라 다른 리턴 타입으로 변경 가능. 4) 파일, 또는 콘솔창에 한글을 출력시키지 말 것. (채점상의 이유. 주석에 포함된
 * 한글은 상관 없음)
 *
 * + 제공하는 프로그램 구조의 개선방법을 제안하고 싶은 분들은 보고서의 결론 뒷부분에 첨부 바랍니다. 내용에 따라 가산점이 있을 수
 * 있습니다.
 */
public class SicSimulator {
    ResourceManager rMgr;
    private SicLoader loader;
    InstTable instTab;
    InstLuncher instLauncher;
    private int currentAddr;
    int returnAddr;
    boolean isStart;
    boolean hasEnd;
    VisualSimulator vs;

    private final int END_ADDR = 0x1003;

    public SicSimulator(ResourceManager resourceManager) {
        // 필요하다면 초기화 과정 추가
        this.rMgr = resourceManager;
        isStart = true;
        hasEnd = false;
        loader = new SicLoader(this.rMgr);
        instTab = new InstTable("inst.data");
        instLauncher = new InstLuncher(resourceManager);
    }

    /**
     * 레지스터, 메모리 초기화 등 프로그램 load와 관련된 작업 수행. 단, object code의 메모리 적재 및 해석은
     * SicLoader에서 수행하도록 한다.
     */
    public void load(File program) {
        /* 메모리 초기화, 레지스터 초기화 등 */
        rMgr.initializeResource();

    }

    public void load(File program, SicLoader sicloader, VisualSimulator _vs) {
        loader = sicloader;
        vs = _vs;
        load(program);
    }

    /**
     * 1개의 instruction이 수행된 모습을 보인다.
     */
    public void oneStep() {
        vs.update();
        char[] temp;
        loader.setSection(currentAddr);
        if (isStart) {
            rMgr.firstInst = currentAddr;
            rMgr.opStartAddr = currentAddr;
            isStart = false;
        }
        temp = rMgr.getMemory(currentAddr, 2);
        rMgr.targetAddr = 0;

        int op = (temp[0]) & 0xFC;
        int pc = 0;
        Instruction inst = instTab.getName(op);
        if (inst != null) {
            addLog(inst.instruction);
        }
        int format;
        if (inst.format == 2) {
            format = 2;
        } else {
            if ((temp[1] & 0x10) == 0)
                format = 3;
            else
                format = 4;
        }

        // 명령어가 2형식일 때
        if (format == 2) {
            pc = currentAddr + 2;
            temp = rMgr.getMemory(currentAddr, 2);
            rMgr.instList.add(beforeLogged(temp));

            int reg = temp[1];
            int reg1 = reg >> 4;
            int reg2 = reg & 0x0f;

            switch (inst.instruction) {
            case "CLEAR":
                instLauncher.clear(reg1);
                break;
            case "COMPR":
                instLauncher.compr(reg1, reg2);
```

```
            break;
        case "TIXR":
            instLauncher.tixr(reg1);
            break;
        }

        currentAddr = pc;
    } else if (format == 3) { //명령어가 3형식일 때
        pc = currentAddr + 3;
        temp = rMgr.getMemory(currentAddr, 3);
        rMgr.instList.add(beforeLogged(temp));

        int disp = ((((int) temp[1]) & 0x0f) << 8) | (int) temp[2];
        int ta = pc + disp;
        if (ta > 4218) {
            ta -= 4096;
        }
        rMgr.targetAddr = ta;
        currentAddr += 3;

        switch (inst.instruction) {
        case "LDT":
            instLauncher.ldt(disp, pc, 3);
            break;
        case "TD":
            instLauncher.td(disp, pc);
            break;
        case "RD":
            instLauncher.rd();
            break;
        case "STA":
            instLauncher.sta(disp, pc);
            break;
        case "WD":
            instLauncher.wd();
            break;
        case "JEQ":
            currentAddr = instLauncher.jeq(disp, pc, currentAddr);
            break;
        case "COMP":
            instLauncher.comp(disp);
            break;
        case "JLT":
            currentAddr = instLauncher.jlt(disp, pc, currentAddr);
            break;
        case "RSUB":
            currentAddr = instLauncher.rsub(returnAddr);
            break;
        case "LDA":
            instLauncher.lda(temp, disp, pc, 3);
            break;
        case "J":
            currentAddr = instLauncher.j(disp, pc);
            hasEnd = (currentAddr == END_ADDR);
            break;
        }
    } else { //명령어가 4형식일 때
        pc = currentAddr + 4;
        temp = rMgr.getMemory(currentAddr, 4);
        int addr = (temp[1] & 0x0f) << 16;
        addr |= temp[2] << 8;
        addr |= temp[3];
        rMgr.targetAddr = addr;
        rMgr.instList.add(beforeLogged(temp));
        switch (inst.instruction) {
        case "JSUB":
            // 이 함수는 Simulator 내에서 처리하는게 훨씬 효율적이라 이렇게 처리함.
            isStart = true;
            returnAddr = pc;
            currentAddr = addr;
            break;
        case "STCH":
            instLauncher.stch(addr);
            currentAddr = pc;
            break;
        case "STX":
            instLauncher.stx(addr);
            currentAddr = pc;
            break;
        case "LDA":
            instLauncher.lda(temp, addr, pc, 4);
            break;
        case "LDT":
            instLauncher.ldt(addr, 0, 4);
            currentAddr = pc;
            break;
        case "LDCH":
            instLauncher.ldch(addr);
            currentAddr = pc;
            break;
        }
    }

}

/**
 * 남은 모든 instruction이 수행된 모습을 보인다.
 */
public void allStep() {
    while (!hasEnd) {
        oneStep();
    }
    rMgr.closeDevice();
}

/**
 * 각 단계를 수행할 때 마다 관련된 기록을 남기도록 한다.
 */
public void addLog(String log) {
    rMgr.logText += log + "₩n";
}
```

```
    private String beforeLogged(char[] temp) {
        int tempaddr = 0;
        for (char c : temp) {
            tempaddr <<= 8;
            tempaddr |= c;
        }
        return Integer.toHexString(tempaddr).toUpperCase();
    }
```

InstLuncher.java

```java
// instruction에 따라 동작을 수행하는 메소드를 정의하는 클래스

public class InstLuncher {
  ResourceManager rMgr;
  char[] Data;

  public InstLuncher(ResourceManager resourceManager) {
    this.rMgr = resourceManager;
  }

  // instruction 별로 동작을 수행하는 메소드를 정의
  public void clear(int reg) {
    rMgr.setRegister(reg, 0);
  }

  public void compr(int reg1, int reg2) {
    if (rMgr.getRegister(reg1) - rMgr.MAX_MEMORY_SIZE + 1 == rMgr.getRegister(reg2))
      rMgr.setRegister(9, 1);
  }

  public void tixr(int reg) {
    int x = rMgr.getRegister(1);
    rMgr.setRegister(1, rMgr.getRegister(1) + 1);
    if (x < rMgr.getRegister(reg))
      rMgr.setRegister(9, 2);
    else
      rMgr.setRegister(9, 0);
  }

  public void ldt(int disp, int pc, int format) {
    if (format == 3) {
      char[] targetAddress = rMgr.getMemory(disp + pc, 3);
      int value = (targetAddress[0] << 16);
      value |= targetAddress[1] << 8;
      value |= targetAddress[2];
      rMgr.setRegister(5, value);
    } else {
      char[] targetAddress = rMgr.getMemory(disp, 3);
      int value = (targetAddress[0] << 16);
      value |= (targetAddress[1] << 8);
      value |= (targetAddress[2]);

      rMgr.setRegister(5, value - 1);
    }
  }

  public void td(int disp, int pc) {
    char[] dev = rMgr.getMemory(pc + disp, 2);
    rMgr.testDevice(String.format("%02X", (int) dev[0]));
    rMgr.setRegister(9, 0);
  }

  public void rd() {
    Data = rMgr.readDevice(rMgr.curDevice, 1);
    rMgr.setRegister(0, Data[0]);
  }

  public void sta(int disp, int pc) {
    int x = rMgr.getRegister(0);
    char[] regA = new char[3];
    regA[0] = (char) (x >> 16);
    regA[1] = (char) ((x >> 8) & 0xff);
    regA[2] = (char) x;

    rMgr.setMemory(disp + pc, regA, 3);
  }

  public void wd() {
    int a = rMgr.getRegister(0);
    char[] va = new char[1];
    va[0] = (char) a;
    rMgr.writeDevice(rMgr.curDevice, va, 1);
  }

  public int jeq(int disp, int pc, int currentAddr) {
    int sw = rMgr.getRegister(9);
    if (sw == 1) {
      currentAddr = pc + disp;
    }
    return currentAddr;
  }

  public void comp(int disp) {
    int a = rMgr.getRegister(0);
    if (a == disp)
      rMgr.setRegister(9, 1);
    else if (a < disp)
      rMgr.setRegister(9, 2);
    else
      rMgr.setRegister(9, 3);
  }

  public int jlt(int disp, int pc, int currentAddr) {
    int sw = rMgr.getRegister(9);
    if (sw == 2) {
      currentAddr = disp + pc;
      if (currentAddr > 4218) {
        currentAddr -= 4096;
      }
    }
    return currentAddr;
  }

  public int rsub(int retAddr) {
    rMgr.curDevice = "";
    return retAddr;
```

```
    }

    public void lda(char[] regVal, int disp, int pc, int format) {
        if (regVal[0] == 0x01)
            rMgr.setRegister(0, disp);
        else {
            char[] taValue = new char[3];
            if (format == 3)
                taValue = rMgr.getMemory(disp + pc, 3);
            else
                taValue = rMgr.getMemory(disp, 3);

            int value = taValue[0] << 16;
            value |= taValue[1] << 8;
            value |= taValue[2];
            rMgr.setRegister(0, value);
        }
    }

    public int j(int disp, int pc) {
        return disp + pc;
    }

    public void stch(int addr) {
        int a = rMgr.getRegister(0);
        char[] cA = new char[1];
        cA[0] = (char) a;
        rMgr.setMemory(addr + rMgr.getRegister(1), cA, 1);
    }

    public void stx(int addr) {
        int x = rMgr.getRegister(1);
        char[] cX = new char[3];
        cX[0] = (char) (x >> 16);
        cX[1] = (char) ((x >> 8) & 0xff);
        cX[2] = (char) x;
        rMgr.setMemory(addr, cX, 3);
    }

    public void ldch(int addr) {
        char[] value = rMgr.getMemory(addr + rMgr.getRegister(1), 1);
        rMgr.setRegister(0, value[0]);
    }
}
```

InstTable.java

```
import java.util.HashMap;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

/**
 * 모든 instruction의 정보를 관리하는 클래스. instruction data들을 저장한다 또한 instruction 관련 연산,
 * 예를 들면 목록을 구축하는 함수, 관련 정보를 제공하는 함수 등을 제공 한다.
 */
public class InstTable {
  /**
   * inst.data 파일을 불러와 저장하는 공간. 명령어의 이름을 집어넣으면 해당하는 Instruction의 정보들을 리턴할 수 있다.
   */
  HashMap<String, Instruction> instMap;

  public static final int MAX_INST = 256;

  /**
   * 클래스 초기화. 파싱을 동시에 처리한다.
   *
   * @param instFile : instruction에 대한 명세가 저장된 파일 이름
   */

  public InstTable(String instFile) {
    instMap = new HashMap<String, Instruction>();
    openFile(instFile);
  }

  /**
   * 입력받은 이름의 파일을 열고 해당 내용을 파싱하여 instMap에 저장한다.
   */
  public void openFile(String fileName) {
    try {
      File file = new File(fileName);
      FileReader filereader = new FileReader(file);
      BufferedReader bufReader = new BufferedReader(filereader);
      String line = "";
      while ((line = bufReader.readLine()) != null) {
        Instruction inst = new Instruction(line);
        instMap.put(inst.instruction, inst);
      }
      bufReader.close();
    } catch (FileNotFoundException e) {
      System.out.println(e);
    } catch (IOException e) {
      System.out.println(e);
    }
  }

  /**
   * 입력받은 이름의 명령어에 대한 정보를 반환
   *
   * @param name : 검색하려고 하는 명령어의 이름
   * @return : 명령어에 대한 정보가 담긴 Instruction
   */
  private Instruction search(String name) {
    return instMap.get(nameToKey(name));
  }

  /**
   * 입력받은 이름의 명령어가 instTable에 있는지 판단해주는 함수.
   *
   * @param name : instTable에 있는지 판단하려는 명령어
   * @return : 명령어 테이블에 있는경우 true, 없는경우 false.
   */
  public boolean exists(String name) {
    return instMap.containsKey(nameToKey(name));
  }

  /**
   * 입력받은 operator가 어떤 형태이던지 간에 instTable의 검색에 사용 가능한 형태로 바꿔주는 함수.
   *
   * @param name : 변환되기 전 String 형태의 operator
   * @return : instTable에서 검색 가능한 형태로 변환된 String 형태의 operator.
   */
  private String nameToKey(String name) {
    return name.replace("+", "");
  }

  /**
   * 입력받은 이름을 가진 명령어가 몇 형식인지 찾아주는 함수.
   *
   * @param name : 찾으려는 명령어의 이름
   * @return : 명령어의 format
   * @exception : 입력받은 이름을 가진 명령어가 없을 경우 Exception throw.
   */
  public int getFormat(String name) throws Exception {
    if (name.charAt(0) == '+')
      return 4;
    if (!exists(name))
      throw new Exception("없는 명령어입니다.");
    return search(name).format;
  }

  /**
   * 입력받은 이름을 가진 명령어가 있다면 그 명령어의 opcode를, 아닐 경우 Exception을 throw한다.
   *
   * @param name : 찾으려는 명령어의 이름
   * @return : 명령어의 opcode
   * @exception : 입력받은 이름을 가진 명령어가 없다면 오류문과 함께 Exception이
```

```
Throw됨.
     */
    public int getOpCode(String name) throws Exception {
        if (!exists(name))
            throw new Exception("없는 명령어입니다.");
        return search(name).opcode;
    }

    public Instruction getName(int opcode) {
        for (String o : instMap.keySet()) {
            if (instMap.get(o).opcode == opcode) {
                return instMap.get(o);
            }
        }
        return null;
    }
}

/**
 * 명령어 하나하나의 구체적인 정보는 Instruction클래스에 담긴다. instruction과 관련된
정보를 저장하고 기초적인 연산을
 * 수행한다.
 */
class Instruction {

    String instruction;
    int opcode;
    int numberOfOperand;

    /** instruction이 몇 바이트 명령어인지 저장. 이후 편의성을 위함 */
    int format;

    /**
     * 클래스를 선언하면서 일반문자열을 즉시 구조에 맞게 파싱한다.
     *
     * @param line : instruction 명세파일로부터 한줄씩 가져온 문자열
     */
    public Instruction(String line) {
        parsing(line);
    }

    /**
     * 일반 문자열을 파싱하여 instruction 정보를 파악하고 저장한다.
     *
     * @param line : instruction 명세파일로부터 한줄씩 가져온 문자열
     */
    public void parsing(String line) {
        String[] line_seperate = line.split("₩t");
        instruction = line_seperate[0];
        format = Integer.parseInt(line_seperate[1]);
        opcode = Integer.parseInt(line_seperate[2], 16);
        numberOfOperand = Integer.parseInt(line_seperate[3]);
    }

}
```

ResourceManager.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Vector;

/**
 * ResourceManager는 컴퓨터의 가상 리소스들을 선언하고 관리하는 클래스이다. 크게 네가지의 가상 자원 공간을 선언하고, 이를
 * 관리할 수 있는 함수들을 제공한다.
 *
 *
 * 1) 입출력을 위한 외부 장치 또는 device 2) 프로그램 로드 및 실행을 위한 메모리 공간. 여기서는 64KB를 최대값으로 잡는다.
 * 3) 연산을 수행하는데 사용하는 레지스터 공간. 4) SYMTAB 등 simulator의 실행 과정에서 사용되는 데이터들을 위한 변수들.
 *
 * 2번은 simulator위에서 실행되는 프로그램을 위한 메모리공간인 반면, 4번은 simulator의 실행을 위한 메모리 공간이라는 점에서
 * 차이가 있다.
 */
public class ResourceManager {
    /**
     * 디바이스는 원래 입출력 장치들을 의미 하지만 여기서는 파일로 디바이스를 대체한다. 즉, 'F1'이라는 디바이스는 'F1'이라는 이름의
     * 파일을 의미한다. deviceManager는 디바이스의 이름을 입력받았을 때 해당 이름의 파일 입출력 관리 클래스를 리턴하는 역할을 한다.
     * 예를 들어, 'A1'이라는 디바이스에서 파일을 read모드로 열었을 경우, hashMap에 <"A1", scanner(A1)> 등을
     * 넣음으로서 이를 관리할 수 있다.
     *
     * 변형된 형태로 사용하는 것 역시 허용한다. 예를 들면 key값으로 String대신 Integer를 사용할 수 있다. 파일 입출력을 위해
     * 사용하는 stream 역시 자유로이 선택, 구현한다.
     *
     * 이것도 복잡하면 알아서 구현해서 사용해도 괜찮습니다.
     */
    public final int REGISTER_NUMBER = 10;
    public final int MAX_MEMORY_SIZE = 65536; // =64KB

    // <String, Obj> -> <String, RandomAccessFile>
    HashMap<String, RandomAccessFile> deviceManager = new HashMap<String, RandomAccessFile>();
    char[] memory = new char[65536]; // String으로 수정해서 사용하여도 무방함.
    int[] register = new int[10];
    // double register_F;
    RandomAccessFile f;
    int j = 0, k = 0;

    SymbolTable symtabList = new SymbolTable();
    // 이외에도 필요한 변수 선언해서 사용할 것.
    String curDevice;
    String programName;
    String logText;
    int programLength;
    int firstInst;
    int targetAddr;
    int opStartAddr;
    int memStartAddr;

    Vector<String> instList = new Vector<>();

    /**
     * 메모리, 레지스터등 가상 리소스들을 초기화한다.
     */
    public void initializeResource() {
        for (int i = 0; i < REGISTER_NUMBER; i++) {
            setRegister(i, 0);
        }
        curDevice = "";
        programName = "";
        logText = "";
        programLength = 0;
        firstInst = 0;
        targetAddr = 0;
        opStartAddr = 0;
        memStartAddr = 0;
    }

    /**
     * deviceManager가 관리하고 있는 파일 입출력 stream들을 전부 종료시키는 역할. 프로그램을 종료하거나 연결을 끊을 때
     * 호출한다.
     */
    public void closeDevice() {
        try {
            for (RandomAccessFile i : deviceManager.values()) {
                i.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * 디바이스를 사용할 수 있는 상황인지 체크. TD명령어를 사용했을 때 호출되는 함수. 입출력 stream을 열고 deviceManager를
     * 통해 관리시킨다.
     *
     * @param devName 확인하고자 하는 디바이스의 번호,또는 이름
     */
    public void testDevice(String devName) {
        curDevice = devName;
        if (deviceManager.containsKey(devName)) {
```

```
            return;
        } else {
            try {
                RandomAccessFile temp = new RandomAccessFile(devName, "r");
                deviceManager.put(devName, temp);
            } catch (FileNotFoundException e) {
                System.out.println(devName + ": 그런 이름을 가진 디바이스가 없습니다. ");
                e.printStackTrace();
            }
        }
    }

    /**
     * 디바이스로부터 원하는 개수만큼의 글자를 읽어들인다. RD명령어를 사용했을 때 호출되는 함수.
     *
     * @param devName 디바이스의 이름
     * @param num     가져오는 글자의 개수
     * @return 가져온 데이터
     */
    public char[] readDevice(String devName, int num) {
        char[] result = new char[num];
        try {
            // 추가한 부분
            if (deviceManager.containsKey(devName)) {
                f = new RandomAccessFile(devName, "r");
                deviceManager.replace(devName, f);
            } else {
                f = new RandomAccessFile(devName, "r");
                deviceManager.put(devName, f);
            }
            f.seek(j++);
            for (int i = 0; i < num; i++)
                result[i] = (char) f.read();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return result;
    }

    /**
     * 디바이스로 원하는 개수 만큼의 글자를 출력한다. WD명령어를 사용했을 때 호출되는 함수.
     *
     * @param devName 디바이스의 이름
     * @param data    보내는 데이터
     * @param num     보내는 글자의 개수
     */
    public void writeDevice(String devName, char[] data, int num) {
        try {
            if (deviceManager.containsKey(devName)) {
                f = new RandomAccessFile(devName, "rw");
                deviceManager.replace(devName, f);
            }
            f.seek(k++);

            for (int i = 0; i < num; i++)
                f.write(data[i]);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * 메모리의 특정 위치에서 원하는 개수만큼의 글자를 가져온다.
     *
     * @param location 메모리 접근 위치 인덱스
     * @param num      데이터 개수
     * @return 가져오는 데이터
     */
    public char[] getMemory(int location, int num) {
        char[] result = new char[num];
        for (int i = 0; i < num; i++)
            result[i] = memory[location + i];
        return result;
    }

    /**
     * 메모리의 특정 위치에 원하는 개수만큼의 데이터를 저장한다.
     *
     * @param locate 접근 위치 인덱스
     * @param data   저장하려는 데이터
     * @param num    저장하는 데이터의 개수
     */
    public void setMemory(int locate, char[] data, int num) {
        for (int i = 0; i < num; i++)
            memory[locate + i] = data[i];

    }

    /**
     * 번호에 해당하는 레지스터가 현재 들고 있는 값을 리턴한다. 레지스터가 들고 있는 값은 문자열이 아님에 주의한다.
     *
     * @param regNum 레지스터 분류번호
     * @return 레지스터가 소지한 값
     */
    public int getRegister(int regNum) {
        return register[regNum];

    }

    /**
     * 모든 레지스터의 값을 배열 형태로 반환해주는 함수.
     *
     * @return 모든 레지스터의 값이 담긴 int형 배열
     */
```

```
    public int[] getRegister() {
        return register;
    }

    /**
     * 번호에 해당하는 레지스터에 새로운 값을 입력한다. 레지스터가 들고 있는 값은 문자열이
아님에 주의한다.
     *
     * @param regNum 레지스터의 분류번호
     * @param value  레지스터에 집어넣는 값
     */
    public void setRegister(int regNum, int value) {
        register[regNum] = value;
    }

    /**
     * 주로 레지스터와 메모리간의 데이터 교환에서 사용된다. int값을 char[]형태로 변경한다.
     *
     * @param data
     * @return char[] 형태로 변경된 data
     */
    public char[] intToChar(int data) {
        String temp = Integer.toString(data);
        return temp.toCharArray();
    }

    /**
     * 주로 레지스터와 메모리간의 데이터 교환에서 사용된다. char[]값을 int형태로 변경한다.
     *
     * @param data
     * @return int 자료형으로 변환된 data
     */
    public int byteToInt(char[] data) {
        String temp = String.valueOf(data);
        return Integer.parseInt(temp);
    }
}
```

SymbolTable.java

```
import java.util.ArrayList;

/**
 * symbol과 관련된 데이터와 연산을 소유한다. section 별로 하나씩 인스턴스를 할당한다.
 */
public class SymbolTable {
   ArrayList<String>  symbolList=new ArrayList<String>();
   ArrayList<Integer>  addressList=new ArrayList<Integer>();
   // 기타 literal, external 선언 및 처리방법을 구현한다.

   /**
    * 새로운 Symbol을 table에 추가한다.
    *
    * @param symbol   : 새로 추가되는 symbol의 label
    * @param address : 해당 symbol이 가지는 주소값
    *                  주의 : 만약 중복된 symbol이 putSymbol을 통해서 입력된다면 이는
프로그램 코드에 문제가 있음을
    *                  나타낸다. 매칭되는 주소값의 변경은 modifySymbol()을 통해서
이루어져야 한다.
    */
   public void putSymbol(String symbol, int address) {
      if (search(symbol)  ==  -1) {
         this.symbolList.add(symbol);
         this.addressList.add(address);
      }
   }

   /**
    * 기존에 존재하는 symbol 값에 대해서 가리키는 주소값을 변경한다.
    *
    * @param symbol       : 변경을 원하는 symbol의 label
    * @param newaddress : 새로 바꾸고자 하는 주소값
    */
   public void modifySymbol(String symbol, int newaddress) {
      int index  =  this.symbolList.indexOf(symbol);
      if (index !=  -1)
         this.addressList.set(index,  newaddress);
   }

   /**
    * 인자로 전달된 symbol이 어떤 주소를 지칭하는지 알려준다.
    *
    * @param symbol : 검색을 원하는 symbol의 label
    * @return symbol이 가지고 있는 주소값. 해당 symbol이 없을 경우 -1 리턴
    */
   public int search(String symbol) {
      int address  =  -1;
      int index  =  this.symbolList.indexOf(symbol);
      if (index !=  -1)
         address  =  this.addressList.get(index);
      return address;
   }
```

SicLoader.java

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.StringTokenizer;

/**
 * SicLoader는 프로그램을 해석해서 메모리에 올리는 역할을 수행한다. 이 과정에서 linker의 역할 또한 수행한다.
 *
 * SicLoader가 수행하는 일을 예를 들면 다음과 같다. - program code를 메모리에 적재시키기 - 주어진 공간만큼 메모리에 빈
 * 공간 할당하기 - 과정에서 발생하는 symbol, 프로그램 시작주소, control section 등 실행을 위한 정보 생성 및 관리
 */
public class SicLoader {
    ResourceManager rMgr;
    private int programAddr;
    private int csAddr;
    private int execAddr;
    ArrayList<Integer> length;
    ArrayList<String> csName;
    ArrayList<String> endLineAddr;

    public SicLoader(ResourceManager resourceManager) {
        setResourceManager(resourceManager);
    }

    /**
     * Loader와 프로그램을 적재할 메모리를 연결시킨다.
     *
     * @param resourceManager 리소스 매니저
     */
    public void setResourceManager(ResourceManager resourceManager) {
        this.rMgr = resourceManager;
    }

    /**
     * object code를 읽어서 load과정을 수행한다. load한 데이터는 resourceManager가 관리하는 메모리에 올라가도록
     * 한다. load과정에서 만들어진 symbol table 등 자료구조 역시 resourceManager에 전달한다.
     *
     * @param objectCode 읽어들인 파일
     */
    public void load(File objectCode) {
        length = new ArrayList<Integer>();
        csName = new ArrayList<String>();
        endLineAddr = new ArrayList<String>();
        pass1(objectCode);
        pass2(objectCode);
        // object code 해석 -> resourceManager setMemory 해주고, symbolTable set해주면됨
    };

    public void setSection(int currentAddr) {
        int section = 0;
        int len = 0;
        for (section = 0; section < length.size(); section++) {
            len += length.get(section);
            if (currentAddr < len)
                break;
        }
        rMgr.programName = csName.get(section);
        rMgr.programLength = length.get(section);
    }

    // 각 Control Section별로 프로그램의 이름, 길이를 저장하고 D레코드의 Symbol들을 SymbolTable에 저장한다.
    private void pass1(File objectCode) {
        SymbolTable symbolTable = rMgr.symtabList;
        csAddr = programAddr;
        try {
            Scanner s = new Scanner(objectCode);

            while (s.hasNext()) {

                String line = s.nextLine();
                if (line.equals("₩n") || line.equals(""))
                    continue;
                String[] headLine = line.split("₩t");
                int controlsection_Length = Integer.parseInt(headLine[1].substring(6), 16);
                length.add(controlsection_Length);
                String sectionName = headLine[0].substring(1);
                csName.add(sectionName);
                symbolTable.symbolList.add(sectionName);
                symbolTable.addressList.add(csAddr);

                line = s.nextLine();
                while (line.charAt(0) != 'E') { // 하나의 Control Section이 끝날 때까지
                    if (line.charAt(0) == 'D') { // 다른 테이블에서 참조하게 될 symbol일 때
                        StringTokenizer strtok = new StringTokenizer(line.substring(1), "0123456789");
                        String[] extdef = new String[strtok.countTokens() * 2];
                        int j = 0;
                        while (strtok.hasMoreTokens()) {
                            extdef[j] = strtok.nextToken();
                            j += 2;
                        }
                        j = 0;

                        int offset = 1;
                        for (int i = 0; i < extdef.length - 2; i += 2) {
                            offset += extdef[i].length();
                            extdef[i + 1] = line.substring(offset, offset + 6);
                            offset += 6;
                        }
```

```
            for (int i = 1; i < extdef.length - 2; i += 2) {
               String symName = extdef[i - 1];
               int symAddr = Integer.parseInt(extdef[i], 16);
               symbolTable.symbolList.add(symName);
               symbolTable.addressList.add(symAddr + csAddr);
            }
         }
         line = s.nextLine();
      }
      endLineAddr.add(line.substring(1));
      csAddr += controlsection_Length;
    }
    s.close();
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  }
}

// T레코드인 경우 리소스 매니저에 ObjectCode를 한 줄씩 메모리 영역에 올려준다.
// M레코드인 경우 해당 Symbol의 실제주소값을 저장하여 저장한다.
private void pass2(File objectCode) {
  csAddr = programAddr;
  execAddr = programAddr;
  try {
    Scanner s = new Scanner(objectCode);

    while (s.hasNext()) {
      String line = s.nextLine();
      if (line.equals("₩n") || line.equals(""))
        continue;
      String[] headLine = line.split("₩t");
      int controlsection_Length = Integer.parseInt(headLine[1].substring(6), 16);
      line = s.nextLine();
      execAddr = csAddr;
      while (line.charAt(0) != 'E') {
        if (line.charAt(0) == 'T') {
          char[] temp = new char[60];
          int templen = 0;
          String textLine = line.substring(9);
          for (int i = 0; i < textLine.length(); i += 2) {
            String strByte = textLine.substring(i, i + 2);
            temp[templen++] = (char) Integer.parseInt(strByte, 16);
          }
          rMgr.setMemory(execAddr, temp, templen);
          execAddr += Integer.parseInt(line.substring(7, 9), 16);
        }

        else if (line.charAt(0) == 'M') { // M레코드일 때 : 레코드의 값 변경 필요.
          int addr = -1;
          String symName = line.substring(10);

          addr = rMgr.symtabList.search(symName);

          char[] modByte = new char[3];
          for (int i = 2; i >= 0; i--) {
            modByte[i] = (char) ((addr >> (8 * Math.abs(i - 2))) & 0xFF);
          }
          if (line.charAt(9) == '+') {
            int modAddr = Integer.parseInt(line.substring(1, 7), 16);
            char[] changed = rMgr.getMemory(csAddr + modAddr, 3);
            for (int i = 0; i < 3; i++)
              changed[i] += modByte[i];

            rMgr.setMemory(csAddr + modAddr, changed, 3);
          } else if (line.charAt(9) == '-') {
            int modAddr = Integer.parseInt(line.substring(1, 7), 16);
            char[] changed = rMgr.getMemory(csAddr + modAddr, 3);
            for (int i = 0; i < 3; i++)
              changed[i] -= modByte[i];

            rMgr.setMemory(csAddr + modAddr, changed, 3);
          }

        }
        line = s.nextLine();
      }
      csAddr += controlsection_Length;
    }
    s.close();
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  }
}

void setStartAddr(int startAddr) {
  this.programAddr = startAddr;
}

int getStartAddr() {
  return programAddr;
}
```