EMB0000243c3415

**시스템프로그래밍 2020 보고서**

**보고서 제출서약서**

**나는 숭실대학교 소프트웨어학부의 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 2020** |
| **과제명** | **프로젝트 2** |
| **담당교수** | **최 재 영 교 수** |
| **제출인** | ***소프트웨어학부 20150286 최유준* (출석번호 310번)** |
| **제출일** | **2020년6월 4일** |

**-------------------- 차 례 --------------------**

1. **동기/목적**
2. **설계/구현 아이디어**

**2.1 설계 아이디어**

**2.2 구현 아이디어**

1. **수행결과**
2. **결론 및 보충할 점**
3. **소스코드**

**1장 동기/목적**

이번 시스템 프로그래밍의 프로젝트는 첫 번째 프로젝트에서 완성한 objectCode를 토대로 프로그램의 목적인 COPY를 완수하는 것입니다. 그 전까지는 어셈블러가 기계어로 번역하는 과정을 공부하였다면 이번에는 그 과정을 토대로 만들어진 output을 새로운 실행의 input으로 이용하는 것입니다. 이 과제를 토대로 linker 및 loader를 직접 구현해보고 그 이론에 대해 좀더 확실히 공부하는 과정입니다.

**2장 설계/구현 아이디어**

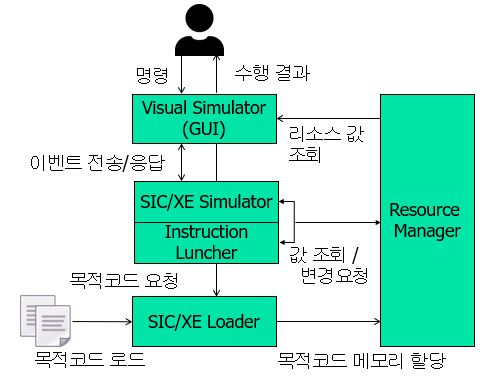
**2.1 설계 아이디어**

파일 전체 개요는 다음과 같습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

사용자가 명령을 내리면 Visual Simulator에서 GUI를 통해 내용을 보여줍니다. Visual Simulator는 SicLoader를 통해 목적코드를 요청합니다. 또한 SicSimulator과 InstLuncher를 통해 Resource Maneger에 접근하여 값을 조회하고 변경을 요청합니다.



**2.2 구현 아이디어**

Visual Simulator에서 Jframe을 extends하여 GUI를 제작합니다. File을 Open하면 SicSimulator에서 메모리 및 레지스터를 초기화하고 SicLoader가 line별로 file을 load합니다. OneStep을 이용할 경우 하나의 instruction마다 진행이 됩니다. Onestep에서 opcode를 통해 명령어의 종류를 분석하고 bool 변수인 extForm을 통해 3, 4형식을 확인합니다.

명령어 별로 instLuncher를 통해 진행한 후 다음 명령어로 넘어갑니다. AllStep을 클릭할 경우 남은 과정을 한번에 실행합니다. OneStep 과정에서 Memory와 static final Register 변수를 이용하여 다음 명령어를 찾고 실행합니다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

**3장 수행결과**

‘F1’ file스크린샷, 방이(가) 표시된 사진

자동 생성된 설명

**4장 결론 및 보충할 점**

링크 로더를 만들면서 프로젝트1보다 더욱더 난이도가 높았기에 개발과정이 더욱 험난하였습니다. 또한 그 과정에서 allStep이 제대로 되지 않는 문제가 발생했습니다. 이러한 문제점을 보완하여 더욱 완성도 높은 프로그램을 개발할 필요가 있습니다.

* **DEBUGING**

컴퓨터, 스크린샷, 모니터, 노트북이(가) 표시된 사진

자동 생성된 설명

**5장 소스코드**

**Visual Simulator.java**

package SP20\_simulator;

import java.awt.FileDialog;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JList;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTextField;

import javax.swing.border.TitledBorder;

/\*\*

 \* VisualSimulator는 사용자와의 상호작용을 담당한다.<br>

 \* 즉, 버튼 클릭등의 이벤트를 전달하고 그에 따른 결과값을 화면에 업데이트 하는 역할을 수행한다.<br>

 \* 실제적인 작업은 SicSimulator에서 수행하도록 구현한다.

 \*/

public class VisualSimulator extends JFrame {

    ResourceManager resourceManager = new ResourceManager();

    SicLoader sicLoader = new SicLoader(resourceManager);

    SicSimulator sicSimulator = new SicSimulator(resourceManager);

    private JTextField textFileName;

    private JTextField textProgName;

    private JTextField textStartAddrObProg;

    private JTextField textLengthProg;

    private JTextField textFirstInstAddr;

    private JTextField textStartAddrMemory;

    private JTextField textADec;

    private JTextField textAHex;

    private JTextField textXDec;

    private JTextField textXHex;

    private JTextField textLDec;

    private JTextField textLHex;

    private JTextField textPCDec;

    private JTextField textPCHex;

    private JTextField textSW;

    private JTextField textTargetAddr;

    private JTextField textBDec;

    private JTextField textBHex;

    private JTextField textSDec;

    private JTextField textSHex;

    private JTextField textTDec;

    private JTextField textTHex;

    private JTextField textF;

    private JTextField textDevice;

    private JList<String> listInst;

    private JList<String> listLog;

    private String filePath;

    public VisualSimulator() {

        super("SIC/XE Simulator");

        setBounds(100, 100, 720, 1100);

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setLayout(null);

        JLabel labelFilename = new JLabel("FileName:");

        labelFilename.setBounds(22, 20, 78, 21);

        add(labelFilename);

        textFileName = new JTextField();

        textFileName.setBounds(105, 17, 156, 27);

        textFileName.setColumns(10);

        add(textFileName);

        JButton btnOpen = new JButton("open");

        btnOpen.setBounds(266, 16, 73, 29);

        btnOpen.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e)

            {

                FileDialog fileD = new FileDialog(new JFrame(), "Open", FileDialog.LOAD);

                fileD.setVisible(true);

                filePath = fileD.getDirectory() + fileD.getFile();

                textFileName.setText(fileD.getFile());

                File file = new File(filePath);

                load(file);

                update();

            }

        });

        add(btnOpen);

        JPanel panel\_1 = new JPanel();

        panel\_1.setBorder(new TitledBorder(null, "H (Header Record)", TitledBorder.LEADING, TitledBorder.TOP, null, null));

        panel\_1.setBounds(17, 60, 322, 180);

        add(panel\_1);

        panel\_1.setLayout(null);

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

        lblProgramName.setBounds(17, 33, 136, 21);

        panel\_1.add(lblProgramName);

        JLabel lblStartAddressOf = new JLabel("Start Address of");

        lblStartAddressOf.setBounds(17, 69, 136, 21);

        panel\_1.add(lblStartAddressOf);

        JLabel lblObjectProgram = new JLabel("Object Program:");

        lblObjectProgram.setBounds(17, 91, 136, 21);

        panel\_1.add(lblObjectProgram);

        JLabel lblLengthOfProgram = new JLabel("Length of Program:");

        lblLengthOfProgram.setBounds(17, 141, 157, 21);

        panel\_1.add(lblLengthOfProgram);

        textProgName = new JTextField();

        textProgName.setBounds(170, 30, 136, 27);

        panel\_1.add(textProgName);

        textProgName.setColumns(10);

        textStartAddrObProg = new JTextField();

        textStartAddrObProg.setBounds(170, 85, 136, 27);

        panel\_1.add(textStartAddrObProg);

        textStartAddrObProg.setColumns(10);

        textLengthProg = new JTextField();

        textLengthProg.setBounds(180, 138, 126, 27);

        panel\_1.add(textLengthProg);

        textLengthProg.setColumns(10);

        JPanel panel = new JPanel();

        panel.setLayout(null);

        panel.setBorder(new TitledBorder(null, "E (End Record)", TitledBorder.LEADING, TitledBorder.TOP, null, null));

        panel.setBounds(356, 60, 322, 104);

        add(panel);

        JLabel lblAddressOfFirst = new JLabel("Address of First instruction");

        lblAddressOfFirst.setBounds(17, 33, 218, 21);

        panel.add(lblAddressOfFirst);

        JLabel lblInIbjectProgram = new JLabel("in Object Program:");

        lblInIbjectProgram.setBounds(17, 65, 157, 21);

        panel.add(lblInIbjectProgram);

        textFirstInstAddr = new JTextField();

        textFirstInstAddr.setColumns(10);

        textFirstInstAddr.setBounds(180, 62, 126, 27);

        panel.add(textFirstInstAddr);

        JLabel lblStartAddressIn = new JLabel("Start Address In Memory");

        lblStartAddressIn.setBounds(356, 179, 229, 21);

        add(lblStartAddressIn);

        textStartAddrMemory = new JTextField();

        textStartAddrMemory.setBounds(512, 213, 166, 27);

        add(textStartAddrMemory);

        textStartAddrMemory.setColumns(10);

        JPanel panel\_2 = new JPanel();

        panel\_2.setLayout(null);

        panel\_2.setBorder(new TitledBorder(null, "Register", TitledBorder.LEADING, TitledBorder.TOP, null, null));

        panel\_2.setBounds(17, 255, 322, 450);

        add(panel\_2);

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

        lblA.setBounds(17, 54, 65, 21);

        panel\_2.add(lblA);

        textADec = new JTextField();

        textADec.setColumns(10);

        textADec.setBounds(85, 51, 103, 27);

        panel\_2.add(textADec);

        textAHex = new JTextField();

        textAHex.setColumns(10);

        textAHex.setBounds(203, 51, 103, 27);

        panel\_2.add(textAHex);

        JLabel lblDec = new JLabel("Dec");

        lblDec.setBounds(121, 15, 43, 21);

        panel\_2.add(lblDec);

        JLabel lblHex = new JLabel("Hex");

        lblHex.setBounds(238, 15, 43, 21);

        panel\_2.add(lblHex);

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

        lblX.setBounds(17, 93, 65, 21);

        panel\_2.add(lblX);

        textXDec = new JTextField();

        textXDec.setColumns(10);

        textXDec.setBounds(85, 90, 103, 27);

        panel\_2.add(textXDec);

        textXHex = new JTextField();

        textXHex.setColumns(10);

        textXHex.setBounds(203, 90, 103, 27);

        panel\_2.add(textXHex);

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

        lblL.setBounds(17, 132, 65, 21);

        panel\_2.add(lblL);

        textLDec = new JTextField();

        textLDec.setColumns(10);

        textLDec.setBounds(85, 129, 103, 27);

        panel\_2.add(textLDec);

        textLHex = new JTextField();

        textLHex.setColumns(10);

        textLHex.setBounds(203, 129, 103, 27);

        panel\_2.add(textLHex);

        JLabel lblB = new JLabel("B (#3)");

        lblB.setBounds(17, 171, 65, 21);

        panel\_2.add(lblB);

        textBDec = new JTextField();

        textBDec.setColumns(10);

        textBDec.setBounds(85, 168, 103, 27);

        panel\_2.add(textBDec);

        textBHex = new JTextField();

        textBHex.setColumns(10);

        textBHex.setBounds(203, 168, 103, 27);

        panel\_2.add(textBHex);

        JLabel lblS = new JLabel("S (#4)");

        lblS.setBounds(17, 210, 65, 21);

        panel\_2.add(lblS);

        textSDec = new JTextField();

        textSDec.setColumns(10);

        textSDec.setBounds(85, 207, 103, 27);

        panel\_2.add(textSDec);

        textSHex = new JTextField();

        textSHex.setColumns(10);

        textSHex.setBounds(203, 207, 103, 27);

        panel\_2.add(textSHex);

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

        lblT.setBounds(17, 249, 65, 21);

        panel\_2.add(lblT);

        textTDec = new JTextField();

        textTDec.setColumns(10);

        textTDec.setBounds(85, 246, 103, 27);

        panel\_2.add(textTDec);

        textTHex = new JTextField();

        textTHex.setColumns(10);

        textTHex.setBounds(203, 246, 103, 27);

        panel\_2.add(textTHex);

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

        lblF.setBounds(17, 288, 65, 21);

        panel\_2.add(lblF);

        textF = new JTextField();

        textF.setColumns(10);

        textF.setBounds(85, 285, 221, 27);

        panel\_2.add(textF);

        JLabel lblPc = new JLabel("PC(#8)");

        lblPc.setBounds(17, 327, 65, 21);

        panel\_2.add(lblPc);

        textPCDec = new JTextField();

        textPCDec.setColumns(10);

        textPCDec.setBounds(85, 324, 103, 27);

        panel\_2.add(textPCDec);

        textPCHex = new JTextField();

        textPCHex.setColumns(10);

        textPCHex.setBounds(203, 324, 103, 27);

        panel\_2.add(textPCHex);

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

        lblSw.setBounds(17, 366, 65, 21);

        panel\_2.add(lblSw);

        textSW = new JTextField();

        textSW.setColumns(10);

        textSW.setBounds(85, 363, 221, 27);

        panel\_2.add(textSW);

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

        lblTargetAddress.setBounds(361, 266, 136, 21);

        add(lblTargetAddress);

        textTargetAddr = new JTextField();

        textTargetAddr.setColumns(10);

        textTargetAddr.setBounds(512, 263, 166, 27);

        add(textTargetAddr);

        JLabel lblInstructions = new JLabel("Instructions:");

        lblInstructions.setBounds(361, 302, 112, 21);

        add(lblInstructions);

        JScrollPane instScroll = new JScrollPane();

        instScroll.setBounds(356, 338, 174, 392);

        add(instScroll);

        listInst = new JList<>();

        instScroll.setViewportView(listInst);

        JLabel label\_1 = new JLabel("Using Device");

        label\_1.setBounds(547, 338, 131, 21);

        add(label\_1);

        textDevice = new JTextField();

        textDevice.setBounds(571, 374, 107, 27);

        add(textDevice);

        textDevice.setColumns(10);

        JButton btnRunOneStep = new JButton("Execute (1 step)");

        btnRunOneStep.setBounds(542, 614, 136, 29);

        btnRunOneStep.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e)

            {

                // 1 step 실행 버튼을 누른 경우,

                // 하나의 명령어를 실행시키는 oneStep() 메소드 호출 후,

                // 변경된 값을 GUI에 보여주기 위해 update() 메소드를 호출한다.

                oneStep();

                update();

            }

        });

        add(btnRunOneStep);

        JButton btnRunAll = new JButton("Execute (all)");

        btnRunAll.setBounds(542, 658, 136, 29);

        btnRunAll.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e)

            {

                // all step 실행 버튼을 누른 경우,

                // 모든 명령어를 실행시키는 allStep() 메소드 호출 후,

                // 변경된 값을 GUI에 보여주기 위해 update() 메소드를 호출한다.

                allStep();

                update();

            }

        });

        add(btnRunAll);

        JButton btnQuit = new JButton("Exit");

        btnQuit.setBounds(542, 702, 136, 29);

        btnQuit.addActionListener(new ActionListener() {

            @Override

            public void actionPerformed(ActionEvent e)

            {

                // 종료 버튼을 누른 경우,

                // 시뮬레이터 프로그램을 종료시킨다.

                System.exit(1);

            }

        });

        add(btnQuit);

        JLabel lblLog = new JLabel("Log (About Operating instruction):");

        lblLog.setBounds(17, 728, 204, 21);

        add(lblLog);

        JScrollPane logScroll = new JScrollPane();

        logScroll.setBounds(17, 752, 664, 227);

        add(logScroll);

        listLog = new JList<>();

        logScroll.setViewportView(listLog);

        setVisible(true);

    }

    /\*\*

     \* 프로그램 로드 명령을 전달한다.

     \*/

    public void load(File program){

        //...

        sicLoader.load(program);

        sicSimulator.load(program);

    };

    /\*\*

     \* 하나의 명령어만 수행할 것을 SicSimulator에 요청한다.

     \*/

    public void oneStep(){

        sicSimulator.oneStep();

    };

    /\*\*

     \* 남아있는 모든 명령어를 수행할 것을 SicSimulator에 요청한다.

     \*/

    public void allStep(){

        sicSimulator.allStep();

    };

    /\*\*

     \* 화면을 최신값으로 갱신하는 역할을 수행한다.

     \*/

    public void update(){

        // 실행한 명령어 리스트를 출력한다.

        String[] instStringList = sicSimulator.getInstList().toArray(new String[sicSimulator.getInstList().size()]);

        listInst.setListData(instStringList);

        // 실행한 명령어 log를 출력한다.

        String[] logStringList = sicSimulator.getLogList().toArray(new String[sicSimulator.getLogList().size()]);

        listLog.setListData(logStringList);

        // 현재 실행중인 컨트롤 섹션 프로그램의 이름을 출력한다.

        textProgName.setText(resourceManager.getProgName(resourceManager.getCurrentSection()));

        // 현재 실행중인 컨트롤 섹션 프로그램의 실행 시작 주소를 출력한다.

        if(resourceManager.getCurrentSection() != 0)

        {

            textStartAddrObProg.setText(String.format("%06X", resourceManager.getProgStartAddr(resourceManager.getCurrentSection()) - resourceManager.getProgLength(resourceManager.getCurrentSection()-1)));

            textFirstInstAddr.setText(String.format("%06X", resourceManager.getProgStartAddr(resourceManager.getCurrentSection()) - resourceManager.getProgLength(resourceManager.getCurrentSection()-1)));

        }

        else

        {

            textStartAddrObProg.setText(String.format("%06X", resourceManager.getProgStartAddr(resourceManager.getCurrentSection())));

            textFirstInstAddr.setText(String.format("%06X", resourceManager.getProgStartAddr(resourceManager.getCurrentSection())));

        }

        // 해당 프로그램의 총 길이를 출력한다.

        int progLength = 0;

        for(int i = 0; i < resourceManager.getProgCount(); i++)

            progLength += resourceManager.getProgLength(i);

        textLengthProg.setText(Integer.toHexString(progLength));

        // 해당 섹션 프로그램의 메모리상에서의 실행 시작 주소를 출력한다.

        textStartAddrMemory.setText(String.format("%06X", resourceManager.getProgStartAddr(resourceManager.getCurrentSection())));

        //각 레지스터의 값을 출력한다.

        textADec.setText(String.format("%d", resourceManager.getRegister(SicSimulator.A\_REGISTER)));

        textAHex.setText(String.format("%06X", resourceManager.getRegister(SicSimulator.A\_REGISTER)));

        textXDec.setText(String.format("%d", resourceManager.getRegister(SicSimulator.X\_REGISTER)));

        textXHex.setText(String.format("%06X", resourceManager.getRegister(SicSimulator.X\_REGISTER)));

        textLDec.setText(String.format("%d", resourceManager.getRegister(SicSimulator.L\_REGISTER)));

        textLHex.setText(String.format("%06X", resourceManager.getRegister(SicSimulator.L\_REGISTER)));

        textPCDec.setText(String.format("%d", resourceManager.getRegister(SicSimulator.PC\_REGISTER)));

        textPCHex.setText(String.format("%06X", resourceManager.getRegister(SicSimulator.PC\_REGISTER)));

        textSW.setText(String.format("%06X", resourceManager.getRegister(SicSimulator.SW\_REGISTER)));

        textTargetAddr.setText(String.format("%06X", sicSimulator.getTargetAddr()));

        textBDec.setText(String.format("%d", resourceManager.getRegister(SicSimulator.B\_REGISTER)));

        textBHex.setText(String.format("%06X", resourceManager.getRegister(SicSimulator.B\_REGISTER)));

        textSDec.setText(String.format("%d", resourceManager.getRegister(SicSimulator.S\_REGISTER)));

        textSHex.setText(String.format("%06X", resourceManager.getRegister(SicSimulator.S\_REGISTER)));

        textTDec.setText(String.format("%d", resourceManager.getRegister(SicSimulator.T\_REGISTER)));

        textTHex.setText(String.format("%06X", resourceManager.getRegister(SicSimulator.T\_REGISTER)));

        //textF.setText(String.format("%f", resourceManager.getFRegister()));

        // 입출력으로 장치를 사용할 경우, 사용중인 장치 정보를 출력한다.

        textDevice.setText(sicSimulator.getDevice());

    };

    public static void main(String[] args) {

        new VisualSimulator();

    }

}

**SymbolTable.java**

package SP20\_simulator;

import java.util.ArrayList;

/\*\*

 \* symbol과 관련된 데이터와 연산을 소유한다.

 \* section 별로 하나씩 인스턴스를 할당한다.

 \*/

public class SymbolTable {

    ArrayList<String> symbolList;

    ArrayList<Integer> addressList;

    // 기타 literal, external 선언 및 처리방법을 구현한다.

    // 수정 사이즈를 담은 리스트

    ArrayList<Integer> modifSizeList;

    // 수정 모드, 즉 부호를 담은 리스트

    ArrayList<Character> modifModeList;

    // 수정할 섹션 프로그램 번호를 담은 리스트

    ArrayList<Integer> sectionList;

    public SymbolTable()

    {

        symbolList = new ArrayList<>();

        addressList = new ArrayList<>();

        modifSizeList = new ArrayList<>();

        modifModeList = new ArrayList<>();

        sectionList = new ArrayList<>();

    }

    /\*\*

     \* 새로운 Symbol을 table에 추가한다.

     \* @param symbol : 새로 추가되는 symbol의 label

     \* @param address : 해당 symbol이 가지는 주소값

     \* <br><br>

     \* 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) {

        String inputSymbol = symbol;

        if (!symbolList.contains(inputSymbol))

        {

            // 심볼과 인자로 들어온 주소값을 저장함

            symbolList.add(inputSymbol);

            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) {

        String inputSymbol = symbol;

        // List 상에 이미 저장되어있는 경우에만 수정이 가능

        if (symbolList.contains(inputSymbol))

        {

            // 저장되어있는 심볼의 위치를 찾아 인자로 받은 새로운 주소값을 넣어줌

            for (int index = 0; index < symbolList.size(); index++)

                if (inputSymbol.equals(symbolList.get(index)))

                {

                    symbolList.set(index, inputSymbol);

                    addressList.set(index, newaddress);

                    break;

                }

        }

    }

    /\*\*

     \* 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) {

        // 출력할 주소값 저장

        int address = 0;

        // 인자로 받은 심볼이 List 상에 있는 경우

        // 해당 심볼의 주소값을 찾아 address에 지정

        if (symbolList.contains(symbol))

        {

            for (int index = 0; index < symbolList.size(); index++)

                if (symbol.equals(symbolList.get(index)))

                {

                    address = addressList.get(index);

                    break;

                }

        }

        // 없는 경우 -1을 address에 지정

        else

            address = -1;

        // address 리턴

        return address;

    }

    public void putExSymbol(String symbol, int address, int modifSize, char modifMode, int section)

    {

        symbolList.add(symbol);

        addressList.add(address);

        modifSizeList.add(modifSize);

        modifModeList.add(modifMode);

        sectionList.add(section);

    }

    public int size()

    {

        return symbolList.size();

    }

    public String getSymbol(int index)//심볼 반환

    {

        return symbolList.get(index);

    }

    public int getaddress(int index)//주소 반환

    {

        return addressList.get(index);

    }

    public int getModifSize(int index)//수정할 사이즈 반환

    {

        return modifSizeList.get(index);

    }

    public char getModifMode(int index)//수정 형 반환

    {

        return modifModeList.get(index);

    }

    public int getSection(int index)//섹션 반환

    {

        return sectionList.get(index);

    }

}

**SicSimulator.java**

package SP20\_simulator;

import java.io.\*;

import java.util.\*;

import org.omg.CORBA.ARG\_IN;

/\*\*

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

 \* ResourceManager에 접근하여 작업을 수행한다.

 \*

 \* 작성중의 유의사항 : <br>

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

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

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

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

 \*

 \* <br><br>

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

 \*/

public class SicSimulator {

    ResourceManager rMgr;

    public static final int A\_REGISTER = 0;

    public static final int X\_REGISTER = 1;

    public static final int L\_REGISTER = 2;

    public static final int B\_REGISTER = 3;

    public static final int S\_REGISTER = 4;

    public static final int T\_REGISTER = 5;

    public static final int F\_REGISTER = 6;

    public static final int PC\_REGISTER = 8;

    public static final int SW\_REGISTER = 9;

    public static final int INIT\_RETADR = 0x4000;

    private int targetAddr;

    private String currentDevice;

    private List<String> instList = new ArrayList<>();

    private List<String> logList = new ArrayList<>();

    InstLuncher inst;

    public SicSimulator(ResourceManager resourceManager) {

        // 필요하다면 초기화 과정 추가

        this.rMgr = resourceManager;

        targetAddr = 0;

        currentDevice = "";

        this.inst = new InstLuncher(rMgr);

    }

    /\*\*

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

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

     \*/

    public void load(File program) {

        /\* 메모리 초기화, 레지스터 초기화 등\*/

        rMgr.initializeResource();

    }

    /\*\*

     \* 1개의 instruction이 수행된 모습을 보인다.

     \*/

    public void oneStep() {

        char [] bytes = rMgr.getMemory(rMgr.getRegister(PC\_REGISTER), 2);

        int temp = (bytes[0] >>> 4) + (bytes[0] & 15);

        int opcode = temp;

        boolean extForm = false;

        boolean pcRelative = false;

        boolean immediate = false;

        boolean indirect = false;

        int registerNum = 0;

        int difference = 0;

        char [] instruction = new char[1];

        // 상위 1바이트를 가지고

        // indirect나 immediate addressing 표시

        if((temp & 2) == 2)

        {

            opcode -= 2;

            indirect = true;

        }

        if((temp & 1) == 1)

        {

            opcode -= 1;

            immediate = true;

        }

        // 두번째 바이트를 가지고

        // 확장된 4형식 명령어를 사용하였는지,

        // PC relative를 사용하였는지 표시

        temp = (bytes[1] >>> 8);

        extForm = (temp & 1) == 1;

        pcRelative = (temp & 2) == 2;

        // target address 초기화

        targetAddr = 0;

        // 분석된 opcode의 값에 따라서 동작을 달리함

        switch(opcode)

        {

            case 0x14:  // STL 명령어: L 레지스터 값을 해당 주소에 저장하는 명령어

                addLog("STL");

                if(extForm)  // 4형식 명령어인 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.STL(opcode, extForm, pcRelative);

                }

                else  // 3형식 명령어인 경우

                {

                    instruction = inst.STL(opcode, extForm, pcRelative);

                }

                break;

            case 0x48: // JSUB 명령어: 주소값으로 들어온 곳으로 이동

                addLog("JSUB");

                if(extForm)  // 4형식 명령어인 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.JSUB(opcode, extForm, pcRelative);

                }

                else   // 3형식 명령어인 경우,

                {

                    instruction = inst.JSUB(opcode, extForm, pcRelative);

                }

                break;

            case 0x00:  // LDA 명령어: 해당 피연산자 주소에 저장된 값을 A 레지스터로 가져옴

                addLog("LDA");

                if(extForm)  // 4형식인 명령어인 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.LDA(opcode, extForm, pcRelative, immediate);

                }

                else  // 3형식 명령어인 경우,

                {

                    instruction = inst.LDA(opcode, extForm, pcRelative, immediate);

                }

                break;

            case 0x28: // COMP 명령어: A레지스터 값과 명령어에 주어진 값과 비교한다.

                addLog("COMP");

                if(extForm)  // 4형식 명령어를 사용하는 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.COMP(opcode, extForm, immediate, difference);

                }

                else  // 3형식 명령어를 사용하는 경우,

                {

                    instruction = inst.COMP(opcode, extForm, immediate, difference);

                }

                break;

            case 0x4c: // RSUB 명령어: L 레지스터에 저장되어있는 주소로 이동(호출 시점 다음 명령어로 돌아감)

                addLog("RSUB");

                instruction = inst.RSUB(opcode);

                currentDevice = "";

                rMgr.setCurrentSection();

                break;

            case 0x50:  // LDCH 명령어: 해당 주소의 값을 A레지스터 하위 1바이트에 불러온다.

                logList.add("LDCH");

                if(extForm)  // 4형식 명령어를 사용하는 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.LDCH(opcode, extForm, pcRelative);

                }

                else // 3형식 명령어를 사용하는 경우,

                {

                    instruction = inst.LDCH(opcode, extForm, pcRelative);

                }

                break;

            case 0xdc: // WD 명령어: 지정된 기기(또는 파일)에 A 레지스터 하위 1바이트의 값을 출력한다.

                addLog("WD");

                if(extForm)  // 4형식 명령어를 사용하는 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.WD(opcode, extForm, pcRelative);

                }//, Integer.toHexString(rMgr.getRegister[A\_REGISTER] & 255), 1

                else  // 3형식 명령어를 사용하는 경우,

                {

                    instruction = inst.WD(opcode, extForm, pcRelative);

                }//, Integer.toHexString(rMgr.getRegister[A\_REGISTER] & 255), 1

                break;

            case 0x3c: // 피연산자로 들어온 주소로 프로그램 흐름을 이동한다.

                addLog("J");

                if(extForm)  // 4형식인 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.J(opcode, extForm, pcRelative, indirect, immediate);

                }

                else  // 3형식인 경우,

                {

                    instruction = inst.J(opcode, extForm, pcRelative, indirect, immediate);

                }

                break;

            case 0x0c:

                addLog("STA"); //STA 명령어: A 레지스터에 저장된 값을 지정된 주소로 저장한다.

                if(extForm)  // 4형식 명령어인 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.STA(opcode, extForm, pcRelative);

                }

                else  // 3형식 명령어인 경우,

                {

                    instruction = inst.STA(opcode, extForm, pcRelative);

                }

                break;

            case 0xb4:  // CLEAR 명령어: 해당 레지스터의 값을 0으로 초기화 시키는 명령어

                logList.add("CLEAR");

                instruction = inst.CLEAR(opcode);

                registerNum = instruction[1] >>> 8;

                rMgr.setRegister(registerNum, 0);

                break;

            case 0x74:  // LDT 명령어: 해당 피연산자의 값을 T 레지스터에 저장한다.

                addLog("LDT");

                if(extForm)

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.LDT(opcode, extForm, pcRelative);

                }

                else

                {

                    instruction = inst.LDT(opcode, extForm, pcRelative);

                }

                break;

            case 0xe0: // TD 명령어: 해당 이름의 기기(또는 파일)의 입출력 스트림을 확인한다.

                addLog("TD");

                if(extForm)  // 4형식 명령어인 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.TD(opcode, extForm);

                    char[] deviceInfo = rMgr.getMemory(targetAddr, 1);

                    String deviceName = String.format("%X%X", deviceInfo[0] >> 8, deviceInfo[0] & 15);

                    currentDevice = deviceName;

                    rMgr.testDevice(deviceName);

                }

                else  // 3형식 명령어인 경우,

                {

                    //instruction = inst.TD(opcode, extForm);

                    instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

                    targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >>> 8) << 4) + (instruction[2] & 15);

                    rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

                    if(pcRelative)

                        targetAddr += rMgr.getRegister(PC\_REGISTER);

                    char[] deviceInfo = rMgr.getMemory(targetAddr, 1);

                    String deviceName = String.format("%X%X", deviceInfo[0] >> 8, deviceInfo[0] & 15);

                    currentDevice = deviceName;

                    rMgr.testDevice(deviceName);

                }

                break;

            case 0xd8:  // RD 명령어: 해당 기기(또는 파일)에서 문자 하나를 읽어 A레지스터에 저장한다.

                addLog("RD");

                if(extForm)

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = rMgr.getMemory(rMgr.getRegister(PC\_REGISTER), 4);

                    targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >> 8) << 4) + (instruction[3] & 15);

                    rMgr.setRegister(PC\_REGISTER, rMgr.getRegister(PC\_REGISTER) + 4);

                    //instruction = inst.RD(opcode, extForm);

                    char[] deviceInfo = rMgr.getMemory(targetAddr, 1);

                    String deviceName = String.format("%X%X", deviceInfo[0] >> 8, deviceInfo[0] & 15);

                    rMgr.setRegister(A\_REGISTER, rMgr.readDevice(deviceName));

                }

                else

                {

                    //instruction = inst.RD(opcode, extForm);

                    instruction = rMgr.getMemory(rMgr.getRegister(PC\_REGISTER), 3);

                    targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >>> 8) << 4) + (instruction[2] & 15);

                    rMgr.setRegister(PC\_REGISTER, rMgr.getRegister(PC\_REGISTER) + 3);

                    if(pcRelative)

                        targetAddr += rMgr.getRegister(PC\_REGISTER);

                    char[] deviceInfo = rMgr.getMemory(targetAddr, 1);

                    String deviceName = String.format("%X%X", deviceInfo[0] >> 8, deviceInfo[0] & 15);

                    rMgr.setRegister(A\_REGISTER, rMgr.readDevice(deviceName));

                }

                break;

            case 0xa0:  // COMPR 명령어: 두 레지스터 값을 비교한다.

                addLog("COMPR");

                //instruction = inst.COMPR(opcode);

                instruction = rMgr.getMemory(rMgr.getRegister(PC\_REGISTER), 2);

                rMgr.setRegister(PC\_REGISTER, rMgr.getRegister(PC\_REGISTER) + 2);

                registerNum = instruction[1] >>> 8;

                int compareRegister = instruction[1] & 15;

                difference = rMgr.getRegister(registerNum) - rMgr.getRegister(compareRegister);

                rMgr.setRegister(SW\_REGISTER, difference);

                break;

            case 0x54:  // STCH 명령어: A레지스터 하위 1바이트에 저장된 문자를 지정된 주소에 저장한다.

                addLog("STCH");

                if(extForm)  // 4형식 명령어인 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.STCH(opcode, extForm, pcRelative);

                }

                else  // 3형식 명령어인 경우,

                {

                    instruction = inst.STCH(opcode, extForm, pcRelative);

                }

                break;

            case 0xb8:  //TIXR 명령어: X 레지스터 값을 1 올리고 피연산자로 들어온 레지스터의 값과 비교한다.

                addLog("TIXR");

                instruction = inst.TIXR();

                registerNum = instruction[1] >>> 8;

                rMgr.setRegister(X\_REGISTER, rMgr.getRegister(X\_REGISTER)+1);

                difference = rMgr.getRegister(X\_REGISTER) - rMgr.getRegister(registerNum);

                rMgr.setRegister(SW\_REGISTER, difference);

                break;

            case 0x38:  // JLT 명령어: 비교 후 작다면 명시된 주소로 이동한다.

                addLog("JLT");

                if(extForm)  // 4형식 명령어인 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.JLT(extForm, pcRelative);

                }

                else  // 3형식 명령어인 경우,

                {

                    instruction = inst.JLT(extForm, pcRelative);

                }

                break;

            case 0x10:  // STX 명령어: X레지스터의 값을 지정된 주소에 저장한다.

                addLog("STX");

                if(extForm)  // 4형식 명령어인 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.STX(extForm, pcRelative);

                }

                else  // 3형식 명령어인 경우,

                {

                    instruction = inst.STX(extForm, pcRelative);

                }

                break;

            case 0x30:  // JEQ 명령어: 비교한 두 값이 같은 경우 지정된 주소로 이동한다.

                addLog("JEQ");

                if(extForm)  // 4형식 명령어인 경우,

                {

                    logList.set(logList.size()-1, "+" + logList.get(logList.size()-1));

                    instruction = inst.JEQ(extForm, pcRelative);

                }

                else // 3형식 명령어인 경우,

                {

                    instruction = inst.JEQ(extForm, pcRelative);

                }

                break;

        }

        // instruction 출력을 위해 4비트에 해당하는 값을 문자로 다시 변환

        // 변환한 instruction을 리스트에 추가한다.

        char[] outputInst = new char[instruction.length\*2];

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

        {

            outputInst[i \* 2] = (char)((instruction[i] >> 8) + '0');

            outputInst[i \* 2 + 1] = (char)((instruction[i] & 255) + '0');

            if((instruction[i] >> 8) >= 10)

                outputInst[i \* 2] += 7;

            if((instruction[i] & 255) >= 10)

                outputInst[i \* 2 + 1] += 7;

        }

        instList.add(new String(outputInst, 0, outputInst.length));

    }

    /\*\*

     \* 남은 모든 instruction이 수행된 모습을 보인다.

     \*/

    public void allStep() {

        while(true)

        {

            oneStep();

            // 초기 지정한 L 레지스터 값과 PC 레지스터 값이 같아지면 작동을 멈춘다.

            if(rMgr.getRegister(PC\_REGISTER) == INIT\_RETADR)

                break;

        }

    }

    /\*\*

     \* 각 단계를 수행할 때 마다 관련된 기록을 남기도록 한다.

     \*/

    public void addLog(String log) {

        logList.add(log);

    }

//log반환

    public List<String> getLogList()

    {

        return logList;

    }

//inst반환

    public List<String> getInstList()

    {

        return instList;

    }

//타겟주소 반환(목적지주소)

    public int getTargetAddr()

    {

        return targetAddr;

    }

//현재 디바이스

    public String getDevice()

    {

        return currentDevice;

    }

}

**SicLoader.java**

package SP20\_simulator;

import java.io.\*;

/\*\*

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

 \* <br><br>

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

 \* - program code를 메모리에 적재시키기<br>

 \* - 주어진 공간만큼 메모리에 빈 공간 할당하기<br>

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

 \*/

public class SicLoader {

    ResourceManager rMgr;

    private int currentSection;

    public SicLoader(ResourceManager resourceManager) {

        // 필요하다면 초기화

        setResourceManager(resourceManager);

        currentSection = 0;

    }

    /\*\*

     \* Loader와 프로그램을 적재할 메모리를 연결시킨다.

     \* @param rMgr

     \*/

    public void setResourceManager(ResourceManager resourceManager) {

        this.rMgr=resourceManager;

    }

    /\*\*

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

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

     \* @param objectCode 읽어들인 파일

     \*/

    public void load(File objectCode){

        try

        {

            // 인자로 들어온 이름의 파일을 열어 오브젝트 코드를 읽어 들인다.

            FileReader fileReader = new FileReader(objectCode);

            BufferedReader bufReader = new BufferedReader(fileReader);

            String line = "";

            // 파일의 끝에 도달할 때까지 한 줄씩 읽어 들인다.

            while((line = bufReader.readLine()) != null)

            {

                // 읽어들인 문자열이 비어있다면 다음 라인을 읽는다.

                if(line.length() == 0)

                    continue;

                // 문자열의 처음 글자로 들어가 있는 레코드 표시로 처리를 달리한다.

                switch(line.charAt(0))

                {

                    // Header Record인 경우,

                    // 해당 섹션 프로그램의 이름과 시작주소, 프로그램 길이, 프로그램의 메모리상의 시작주소 등을 저장한다.

                    case 'H':

                        int progNameLength = line.length()-13;

                        String programName = line.substring(1, progNameLength);

                        rMgr.setProgName(programName, currentSection);

                        rMgr.setProgStartAddr(line.substring(progNameLength+1, progNameLength+7), currentSection);

                        rMgr.setProgLength(line.substring(line.length()-6, line.length()), currentSection);

                        rMgr.symtabList.putSymbol(programName, rMgr.getProgStartAddr(currentSection));

                        break;

                    // Define Record인 경우,

                    // 해당 symbol과 그 주소를 테이블에 저장한다.

                    case 'D':

                        int symNameLength = 0;

                        int symNameStart = 1;

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

                        {

                            if(line.charAt(i) == '0')

                            {

                                String symbol = line.substring(symNameStart, symNameStart+symNameLength);

                                int address = Integer.parseInt(line.substring(symNameStart+symNameLength, symNameStart+symNameLength+6), 16);

                                rMgr.symtabList.putSymbol(symbol, address);

                                symNameStart += symNameLength + 6;

                                i += 5;

                                symNameLength = 0;

                                continue;

                            }

                            symNameLength++;

                        }

                        break;

                    // Refer Record인 경우, 넘어간다.

                    case 'R':

                        break;

                    // Text Record인 경우,

                    // 시작주소부터 명시된 길이만큼 메모리에 오브젝트 코드를 로드한다.

                    // 오브젝트 코드를 로드하기 전, 한 char형에 두 글자가 담기도록 packing 과정을 거친다.

                    case 'T':

                        int currentAddr = Integer.parseInt(line.substring(1, 7), 16) + rMgr.getProgStartAddr(currentSection);

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

                        char[] packedOpcode = packing(line.substring(9, line.length()).toCharArray());

                        rMgr.setMemory(currentAddr, packedOpcode, codeLength);

                        break;

                    // Modification Record인 경우,

                    // 수정을 위한 EXTAB에 수정할 주소, 수정할 부분의 개수, 주소를 더할 것인지 뺄 것인지, 심볼 정보를 저장한다.

                    case 'M':

                        int modifLocation = Integer.parseInt(line.substring(1, 7), 16) + +rMgr.getProgStartAddr(currentSection);

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

                        char modifMode = line.charAt(9);

                        String symbol = line.substring(10, line.length());

                        rMgr.extabList.putExSymbol(symbol, modifLocation, modifSize, modifMode, currentSection);

                        break;

                    // End Record인 경우,

                    // 컨트롤 섹션을 표시하는 currentSection의 값을 올린다.

                    case 'E':

                        currentSection++;

                        break;

                }

            }

            // EXTAB에 저장되어있는 수정 정보들을 가지고

            // 메모리에 올라가있는 명령어를 수정한다.

            for(int i = 0; i < rMgr.extabList.size(); i++)

            {

                String symbol = rMgr.extabList.getSymbol(i);

                int modifSize = rMgr.extabList.getModifSize(i);

                char modifMode = rMgr.extabList.getModifMode(i);

                String modifAddr = "000000";

                if(modifSize == 5)

                    modifAddr = String.format("%05X", rMgr.symtabList.search(symbol));

                else if(modifSize == 6)

                    modifAddr = String.format("%06X", rMgr.symtabList.search(symbol));

                char[] packedAddr = packing(modifAddr.toCharArray());

                rMgr.modifMemory(rMgr.extabList.getaddress(i), packedAddr, packedAddr.length, modifMode);

            }

            bufReader.close();

        }

        catch (FileNotFoundException e)

        {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }

        catch (IOException e)

        {

        }

    }

    //바이트로 묶어주는 함수

    private char[] packing(char[] inputData)

    {

        int length = (inputData.length / 2) + (inputData.length % 2);

        char[] outputData = new char[length];

        int upByte = 0;

        int downByte = 0;

        if (inputData.length % 2 == 0)

        {

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

            {

                upByte = inputData[i \* 2] - '0';

                downByte = inputData[i \* 2 + 1] - '0';

                if (upByte >= 10)

                    upByte -= 7;

                if (downByte >= 10)

                    downByte -= 7;

                outputData[i] = (char) ((upByte << 8) + downByte);

            }

        }

        else

        {

            downByte = (inputData[0] - '0');

            if(downByte >= 10)

                downByte -= 7;

            outputData[0] = (char) downByte;

            for (int i = 1; i < length; i++)

            {

                upByte = inputData[i \* 2 - 1] - '0';

                downByte = inputData[i \* 2] - '0';

                if (upByte >= 10)

                    upByte -= 7;

                if (downByte >= 10)

                    downByte -= 7;

                outputData[i] = (char) ((upByte << 8) + downByte);

            }

        }

        return outputData;

    }

}

**ResourceManager.java**

package SP20\_simulator;

import java.io.\*;

import java.util.ArrayList;

import java.util.\*;

/\*\*

 \* ResourceManager�� ��ǻ���� ���� ���ҽ����� �����ϰ� �����ϴ� Ŭ�����̴�.

 \* ũ�� �װ����� ���� �ڿ� ������ �����ϰ�, �̸� ������ �� �ִ� �Լ����� �����Ѵ�.<br><br>

 \*

 \* 1) ������� ���� �ܺ� ��ġ �Ǵ� device<br>

 \* 2) ���α׷� �ε� �� ������ ���� �޸� ����. ���⼭�� 64KB�� �ִ밪���� ��´�.<br>

 \* 3) ������ �����ϴµ� ����ϴ� �������� ����.<br>

 \* 4) SYMTAB �� simulator�� ���� �������� ���Ǵ� �����͵��� ���� ������.

 \* <br><br>

 \* 2���� simulator������ ����Ǵ� ���α׷��� ���� �޸𸮰����� �ݸ�,

 \* 4���� simulator�� ������ ���� �޸� �����̶�� ������ ���̰� �ִ�.

 \*/

public class ResourceManager{

    /\*\*

     \* ����̽��� ���� ����� ��ġ���� �ǹ� ������ ���⼭�� ���Ϸ� ����̽��� ��ü�Ѵ�.<br>

     \* ��, 'F1'�̶�� ����̽��� 'F1'�̶�� �̸��� ������ �ǹ��Ѵ�. <br>

     \* deviceManager�� ����̽��� �̸��� �Է¹޾��� �� �ش� �̸��� ���� ����� ���� Ŭ������ �����ϴ� ������ �Ѵ�.

     \* ���� ���, 'A1'�̶�� ����̽����� ������ read���� ������ ���, hashMap�� <"A1", scanner(A1)> ���� �������μ� �̸� ������ �� �ִ�.

     \* <br><br>

     \* ������ ���·� ����ϴ� �� ���� ����Ѵ�.<br>

     \* ���� ��� key������ String��� Integer�� ����� �� �ִ�.

     \* ���� ������� ���� ����ϴ� stream ���� �������� ����, �����Ѵ�.

     \* <br><br>

     \* �̰͵� �����ϸ� �˾Ƽ� �����ؼ� ����ص� �������ϴ�.

     \*/

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

    char[] memory = new char[65536]; // String���� �����ؼ� ����Ͽ��� ������.

    int[] register = new int[10];

    double register\_F;

    SymbolTable symtabList = new SymbolTable();

    private int currentSection;

    private int readPointer = 0;

    SymbolTable extabList = new SymbolTable();

    private ArrayList<String> progNameList = new ArrayList<String>();

    private ArrayList<Integer> progLengthList = new ArrayList<Integer>();

    private ArrayList<Integer> progStartAddrList = new ArrayList<Integer>();

    // �̿ܿ��� �ʿ��� ���� �����ؼ� ����� ��.

    /\*\*

     \* �޸�, �������͵� ���� ���ҽ����� �ʱ�ȭ�Ѵ�.

     \*/

    public void initializeResource(){

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

            register[i] = 0;

        register\_F = 0;

        currentSection = 0;

        register[SicSimulator.X\_REGISTER] = progStartAddrList.get(currentSection);

        register[SicSimulator.L\_REGISTER] = SicSimulator.INIT\_RETADR;

    }

    /\*\*

     \* deviceManager�� �����ϰ� �ִ� ���� ����� stream���� ���� �����Ű�� ����.

     \* ���α׷��� �����ϰų� ������ ���� �� ȣ���Ѵ�.

     \*/

    public void closeDevice() {

        Iterator<String> it = deviceManager.keySet().iterator();

        while (it.hasNext())

        {

            String key = it.next();

            Object stream = deviceManager.get(key);

            try

            {

                if (stream instanceof FileReader)

                {

                    ((FileReader) stream).close();

                }

                else if (stream instanceof FileWriter)

                {

                    ((FileWriter) stream).close();

                }

            }

            catch (IOException e)

            {

                // TODO Auto-generated catch block

                e.printStackTrace();

            }

        }

    }

    /\*\*

     \* ����̽��� ����� �� �ִ� ��Ȳ���� üũ. TD���ɾ ������� �� ȣ��Ǵ� �Լ�.

     \* ����� stream�� ���� deviceManager�� ���� ������Ų��.

     \* @param devName Ȯ���ϰ��� �ϴ� ����̽��� ��ȣ,�Ǵ� �̸�

     \*/

    public void testDevice(String devName) {

        String str = devName+".txt";

        File file = new File(str);

        if(file != null){

            deviceManager.put(devName,(File)file);

            register[SicSimulator.SW\_REGISTER] = 1;

            setRegister(SicSimulator.PC\_REGISTER, getRegister(SicSimulator.PC\_REGISTER) + 3);

        }

    }

    /\*\*

     \* ����̽��κ��� ���ϴ� ������ŭ�� ���ڸ� �о���δ�. RD���ɾ ������� �� ȣ��Ǵ� �Լ�.

     \* @param devName ����̽��� �̸�

     \* @param num �������� ������ ����

     \* @return ������ ������

     \*/

    public char readDevice(String devName){

        char input = ' ';

        try

        {

            File file = (File)deviceManager.get(devName);

            FileReader fileReader = new FileReader(file);

            int inputChar = 0;

            int index = 0;

            while(index <= readPointer)

            {

                inputChar = fileReader.read();

                index++;

            }

            if (inputChar != -1)

            {

                input = (char) inputChar;

            }

            else

                input = 0;

            readPointer++;

            fileReader.close();

        }

        catch (FileNotFoundException e)

        {

        }

        catch (IOException e)

        {

            System.out.println(e);

        }

        return input;

    }

    /\*\*

     \* ����̽��� ���ϴ� ���� ��ŭ�� ���ڸ� ����Ѵ�. WD���ɾ ������� �� ȣ��Ǵ� �Լ�.

     \* @param devName ����̽��� �̸�

     \* @param data ������ ������

     \* @param num ������ ������ ����

     \*/

    public void writeDevice(String devName){

        try

        {

            File file = (File)deviceManager.get(devName);

            FileWriter fileWriter = new FileWriter(file, true);

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

                fileWriter.write((char)(register[SicSimulator.A\_REGISTER] & 255));

                fileWriter.close();

            //}

            //, char[] data, int num

        }

        catch (FileNotFoundException e)

        {

            // ������ ã�� ������ ���� ���� �ڵ鸵

        }

        catch (IOException e)

        {

            System.out.println(e);

        }

    }

    /\*\*

     \* �޸��� Ư�� ��ġ���� ���ϴ� ������ŭ�� ���ڸ� �����´�.

     \* @param location �޸� ���� ��ġ �ε���

     \* @param num ������ ����

     \* @return �������� ������

     \*/

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

        char[] result = new char[num];

        for (int i = location; i < location + num; i++)

        {

            result[i-location] = memory[i];

        }

        return result;

    }

    /\*\*

     \* �޸��� Ư�� ��ġ�� ���ϴ� ������ŭ�� �����͸� �����Ѵ�.

     \* @param locate ���� ��ġ �ε���

     \* @param data �����Ϸ��� ������

     \* @param num �����ϴ� �������� ����

     \*/

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

        for (int i = locate; i < locate + num; i++)

        {

            memory[i] = data[i - locate];

            System.out.print(data[i - locate] >> 8);

            System.out.print(" ");

            System.out.print(data[i - locate] & 255);

        }

    }

    /\*\*

     \* ��ȣ�� �ش��ϴ� �������Ͱ� ���� ��� �ִ� ���� �����Ѵ�. �������Ͱ� ��� �ִ� ���� ���ڿ��� �ƴԿ� �����Ѵ�.

     \* @param regNum �������� �з���ȣ

     \* @return �������Ͱ� ������ ��

     \*/

    public int getRegister(int regNum){

        return register[regNum];

    }

    /\*\*

     \* ��ȣ�� �ش��ϴ� �������Ϳ� ���ο� ���� �Է��Ѵ�. �������Ͱ� ��� �ִ� ���� ���ڿ��� �ƴԿ� �����Ѵ�.

     \* @param regNum ���������� �з���ȣ

     \* @param value �������Ϳ� ����ִ� ��

     \*/

    public void setRegister(int regNum, int value){

        register[regNum] = value;

    }

    /\*\*

     \* �ַ� �������Ϳ� �޸𸮰��� ������ ��ȯ���� ���ȴ�. int���� char[]���·� �����Ѵ�.

     \* @param data

     \* @return

     \*/

    public char[] intToChar(int data){

        char[] inputData = String.format("%X", data).toCharArray();

        int length = (inputData.length / 2) + (inputData.length % 2);

        char[] outputData = new char[length];

        int upByte = 0;

        int downByte = 0;

        if (inputData.length % 2 == 0)

        {

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

            {

                upByte = inputData[i \* 2] - '0';

                downByte = inputData[i \* 2 + 1] - '0';

                if (upByte >= 10)

                    upByte -= 7;

                if (downByte >= 10)

                    downByte -= 7;

                outputData[i] = (char) ((upByte << 8) + downByte);

            }

        }

        else

        {

            downByte = (inputData[0] - '0');

            if(downByte >= 10)

                downByte -= 7;

            outputData[0] = (char) downByte;

            for (int i = 1; i < length; i++)

            {

                upByte = inputData[i \* 2 - 1] - '0';

                downByte = inputData[i \* 2] - '0';

                if (upByte >= 10)

                    upByte -= 7;

                if (downByte >= 10)

                    downByte -= 7;

                outputData[i] = (char) ((upByte << 8) + downByte);

            }

        }

        return outputData;

    }

    /\*\*

     \* �ַ� �������Ϳ� �޸𸮰��� ������ ��ȯ���� ���ȴ�. char[]���� int���·� �����Ѵ�.

     \* @param data

     \* @return

     \*/

    public int byteToInt(char[] data){

        int result = 0;

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

        {

            result = result << 4;

            result += (data[i] >> 8);

            result = result << 4;

            result += (data[i] & 255);

        }

        return result;

    }

    //프로그램이름 저장

    public void setProgName(String progName, int sectionNum)

    {

        progNameList.add(sectionNum, progName);

    }

    //프로그램 시작주소 저장

    public void setProgStartAddr(String startAddr, int sectionNum)

    {

        int addr = Integer.parseInt(startAddr, 16);

        if (sectionNum > 0)

        {

            addr += progStartAddrList.get(sectionNum - 1) + progLengthList.get(sectionNum - 1);

        }

        progStartAddrList.add(sectionNum, addr);

    }

    //프로그램 길이 저장

    public void setProgLength(String length, int sectionNum)

    {

        progLengthList.add(sectionNum, Integer.parseInt(length, 16));

    }

    //프로그램 시작주소 반환

    public int getProgStartAddr(int sectionNum)

    {

        return progStartAddrList.get(sectionNum);

    }

    //프로그램 이름 반환

    public String getProgName(int sectionNum)

    {

        return progNameList.get(sectionNum);

    }

    //프로그램 길이 반환

    public int getProgLength(int section)

    {

        return progLengthList.get(section);

    }

    //이름크기 반환

    public int getProgCount()

    {

        return progNameList.size();

    }

    //현재 섹션

    public int getCurrentSection()

    {

        return currentSection;

    }

    //현재 섹션 저장

    public void setCurrentSection()

    {

        int  i = 0;

        for(i = 0; i < getProgCount(); i++)

        {

            if(getProgStartAddr(i) <= getRegister(SicSimulator.PC\_REGISTER) && getRegister(SicSimulator.PC\_REGISTER) < getProgStartAddr(i) + getProgLength(i))

            {

                currentSection = i;

                break;

            }

        }

        if(i == getProgCount())

            currentSection = 0;

    }

    //수정데이터 메모리

    public void modifMemory(int locate, char[] data, int num, char modifMode)

    {

        if (modifMode == '+')

        {

            for (int i = locate; i < locate + num; i++)

            {

                memory[i] += data[i - locate];

            }

        }

        else if (modifMode == '-')

        {

            for (int i = locate; i < locate + num; i++)

            {

                memory[i] -= data[i - locate];

            }

        }

    }

}

**InstLuncher.java**

package SP20\_simulator;

// instruction에 따라 동작을 수행하는 메소드를 정의하는 클래스

public class InstLuncher {

    ResourceManager rMgr;

    int targetAddr;

    SicSimulator sim;

    char [] instruction;

    public InstLuncher(ResourceManager resourceManager) {

        this.rMgr = resourceManager;

        instruction = new char[1];

    }

    // instruction 별로 동작을 수행하는 메소드를 정의

    // ex) public void add(){...}

    public char[] STL(int opcode, boolean extForm, boolean pcRelative) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >>> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >>> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            char[] data = new char[3];

            rMgr.setMemory(targetAddr, data, 3);

            data = rMgr.intToChar(rMgr.getRegister(SicSimulator.L\_REGISTER));

            rMgr.modifMemory(targetAddr + (3 - data.length), data, data.length, '+');

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >>> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if(pcRelative)

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

            char[] data = new char[3];

            rMgr.setMemory(targetAddr, data, 3);

            data = rMgr.intToChar(rMgr.getRegister(SicSimulator.L\_REGISTER));

            rMgr.modifMemory(targetAddr + (3 - data.length), data, data.length, '+');

        }

        return instruction;

    }

    public char[] JSUB(int opcode, boolean extForm, boolean pcRelative) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            rMgr.setRegister(SicSimulator.L\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER));

            rMgr.setRegister(SicSimulator.PC\_REGISTER, targetAddr);

            rMgr.setCurrentSection();

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            rMgr.setRegister(SicSimulator.L\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER));

            if(pcRelative)

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, targetAddr);

            rMgr.setCurrentSection();

        }

        return instruction;

    }

    public char[] LDA(int opcode, boolean extForm, boolean pcRelative, boolean immediate) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            char[] data = rMgr.getMemory(targetAddr, 3);

            rMgr.setRegister(SicSimulator.A\_REGISTER, rMgr.byteToInt(data));

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if(pcRelative) // PC relative를 사용하는 경우,

            {

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

                char[] data = rMgr.getMemory(targetAddr, 3);

                rMgr.setRegister(SicSimulator.A\_REGISTER, rMgr.byteToInt(data));

            }

            else if(immediate)  // immediate를 사용하는 경우,

            {

                rMgr.setRegister(SicSimulator.A\_REGISTER, targetAddr);

            }

        }

        return instruction;

    }

    public char[] COMP(int opcode, boolean extForm, boolean immediate, int difference) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >>> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >>> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            if(immediate) {

                difference = rMgr.getRegister(SicSimulator.A\_REGISTER) - targetAddr;

                rMgr.setRegister(SicSimulator.SW\_REGISTER, difference);

                System.out.println("SW: " + rMgr.getRegister(SicSimulator.SW\_REGISTER));

            }

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >>> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if(immediate)

            {

                difference = rMgr.getRegister(SicSimulator.A\_REGISTER) - targetAddr;

                rMgr.setRegister(SicSimulator.SW\_REGISTER, difference);

                System.out.println("SW: " + rMgr.getRegister(SicSimulator.SW\_REGISTER));

            }

        }

        return instruction;

    }

    public char[] RSUB(int opcode) {

        instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

        rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.L\_REGISTER));

        return instruction;

    }

    public char[] LDCH(int opcode, boolean extForm, boolean pcRelative) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >>> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >>> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            char[] data = rMgr.getMemory(targetAddr + rMgr.getRegister(SicSimulator.X\_REGISTER), 1);

            rMgr.setRegister(SicSimulator.A\_REGISTER, rMgr.byteToInt(data));

            System.out.println((char)rMgr.getRegister(SicSimulator.A\_REGISTER));

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >>> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if(pcRelative)  // PC relative를 사용하는 경우,

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

            char[] data = rMgr.getMemory(targetAddr + rMgr.getRegister(SicSimulator.X\_REGISTER), 1);

            rMgr.setRegister(SicSimulator.A\_REGISTER, rMgr.byteToInt(data));

            System.out.println((char)rMgr.getRegister(SicSimulator.A\_REGISTER));

        }

        return instruction;

    }

    public char[] WD(int opcode, boolean extForm, boolean pcRelative) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >>> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >>> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            char[] deviceInfo = rMgr.getMemory(targetAddr, 1);

            String deviceName = String.format("%X%X", deviceInfo[0] >> 8, deviceInfo[0] & 15);

            System.out.println(deviceName);

            rMgr.writeDevice(deviceName);

            System.out.print(Integer.toBinaryString(rMgr.getRegister(SicSimulator.A\_REGISTER)));

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >>> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if(pcRelative)

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

            char[] deviceInfo = rMgr.getMemory(targetAddr, 1);

            String deviceName = String.format("%X%X", deviceInfo[0] >> 8, deviceInfo[0] & 15);

            System.out.println(deviceName);

            rMgr.writeDevice(deviceName);

            System.out.print(Integer.toBinaryString(rMgr.getRegister(SicSimulator.A\_REGISTER)));

        }

        return instruction;

    }

    public char[] J(int opcode, boolean extForm, boolean pcRelative, boolean indirect, boolean immediate) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >> 8) << 4) + (instruction[3] & 15);

            if((instruction[1] & 15) == 15)  // 음수인 경우(상위 8비트가 F인 경우)

                targetAddr += (0xFFF << 20); // 상위 나머지 비트들도 F로 채워 음수값을 만든다.

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, targetAddr);

            rMgr.setCurrentSection();

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >> 8) << 4) + (instruction[2] & 15);

            if((instruction[1] & 15) == 15)   // 음수인 경우(상위 8비트가 F인 경우)

                targetAddr += (0xFFFFF << 12);  // 상위 나머지 비트들도 F로 채워 음수값을 만든다.

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if(pcRelative)

            {

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

                if(indirect & !immediate)

                {

                    targetAddr = rMgr.byteToInt(rMgr.getMemory(targetAddr, 3));

                }

                rMgr.setRegister(SicSimulator.PC\_REGISTER, targetAddr);

            }

            rMgr.setCurrentSection();

        }

        return instruction;

    }

    public char[] STA(int opcode, boolean extForm, boolean pcRelative) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            char[] data = new char[3];

            rMgr.setMemory(targetAddr, data, 3);

            data = rMgr.intToChar(rMgr.getRegister(SicSimulator.A\_REGISTER));

            rMgr.modifMemory(targetAddr + (3 - data.length), data, data.length, '+');

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if(pcRelative)

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

            char[] data = new char[3];

            rMgr.setMemory(targetAddr, data, 3);

            data = rMgr.intToChar(rMgr.getRegister(SicSimulator.A\_REGISTER));

            rMgr.modifMemory(targetAddr + (3 - data.length), data, data.length, '+');

        }

        return instruction;

    }

    public char[] CLEAR(int opcode) {

        instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 2);

        rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 2);

        return instruction;

    }

    public char[] LDT(int opcode, boolean extForm, boolean pcRelative) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            char[] data = rMgr.getMemory(targetAddr, 3);

            rMgr.setRegister(SicSimulator.T\_REGISTER, rMgr.byteToInt(data));

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if(pcRelative)

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

            char[] data = rMgr.getMemory(targetAddr, 3);

            rMgr.setRegister(SicSimulator.T\_REGISTER, rMgr.byteToInt(data));

        }

        return instruction;

    }

    public char[] TD(int opcode, boolean extForm) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >>> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

        }

        return instruction;

    }

   /\* public char[] RD(int opcode, boolean extForm) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >>> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

        }

        return instruction;

    }\*/

   /\* public char[] COMPR(int opcode) {

        instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 2);

        rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 2);

        return instruction;

    }\*/

    public char[] STCH(int opcode, boolean extForm, boolean pcRelative) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            char [] data = rMgr.intToChar(rMgr.getRegister(SicSimulator.A\_REGISTER) & 255);

            rMgr.setMemory(targetAddr + rMgr.getRegister(SicSimulator.X\_REGISTER), data, 1);

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if(pcRelative)

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

            char [] data = rMgr.intToChar(rMgr.getRegister(SicSimulator.A\_REGISTER) & 255);

            rMgr.setMemory(targetAddr + rMgr.getRegister(SicSimulator.X\_REGISTER), data, 1);

        }

        return instruction;

    }

    public char[] TIXR() {

        instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 2);

        rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 2);

        return instruction;

    }

    public char[] JLT(boolean extForm, boolean pcRelative) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >>> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >>> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            if((instruction[1] & 15) == 15)

                targetAddr += (0xFFF << 20);

            if(rMgr.getRegister(SicSimulator.SW\_REGISTER) < 0)

            {

                rMgr.setRegister(SicSimulator.PC\_REGISTER, targetAddr);

            }

            rMgr.setCurrentSection();

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >>> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if((instruction[1] & 15) == 15)

                targetAddr += (0xFFFFF << 12);

            if(pcRelative)

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

            if(rMgr.getRegister(SicSimulator.SW\_REGISTER) < 0)

            {

                rMgr.setRegister(SicSimulator.PC\_REGISTER, targetAddr);

            }

            rMgr.setCurrentSection();

        }

        return instruction;

    }

    public char[] STX(boolean extForm, boolean pcRelative) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >>> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >>> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            char[] data = new char[3];

            rMgr.setMemory(targetAddr, data, 3);

            data = rMgr.intToChar(rMgr.getRegister(SicSimulator.X\_REGISTER));

            rMgr.modifMemory(targetAddr + (3 - data.length), data, data.length, '+');

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >>> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if(pcRelative)

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

            char[] data = new char[3];

            rMgr.setMemory(targetAddr, data, 3);

            data = rMgr.intToChar(rMgr.getRegister(SicSimulator.X\_REGISTER));

            rMgr.modifMemory(targetAddr + (3 - data.length), data, data.length, '+');

        }

        return instruction;

    }

    public char[] JEQ(boolean extForm, boolean pcRelative) {

        if(extForm) {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 4);

            targetAddr = ((instruction[1] & 15) << 16) + ((instruction[2] >>> 8) << 12) + ((instruction[2] & 15) << 8) + ((instruction[3] >>> 8) << 4) + (instruction[3] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 4);

            if((instruction[1] & 15) == 15)

                targetAddr += (0xFFF << 20);

            if(rMgr.getRegister(SicSimulator.SW\_REGISTER) == 0)

            {

                rMgr.setRegister(SicSimulator.PC\_REGISTER, targetAddr);

            }

            rMgr.setCurrentSection();

        }

        else {

            instruction = rMgr.getMemory(rMgr.getRegister(SicSimulator.PC\_REGISTER), 3);

            targetAddr = ((instruction[1] & 15) << 8) + ((instruction[2] >>> 8) << 4) + (instruction[2] & 15);

            rMgr.setRegister(SicSimulator.PC\_REGISTER, rMgr.getRegister(SicSimulator.PC\_REGISTER) + 3);

            if((instruction[1] & 15) == 15)

                targetAddr += (0xFFFFF << 12);

            if(pcRelative)

                targetAddr += rMgr.getRegister(SicSimulator.PC\_REGISTER);

            if(rMgr.getRegister(SicSimulator.SW\_REGISTER) == 0)

            {

                rMgr.setRegister(SicSimulator.PC\_REGISTER, targetAddr);

            }

            rMgr.setCurrentSection();

        }

        return instruction;

    }

}