

프로그램 보고서

나는 송실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.
나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

1. 나는 자력으로 보고서를 작성하였습니다.
2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다.
3. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.

교과목	시스템프로그래밍 2025
프로젝트 명	Project #2 - ObjectCpde 시뮬레이터 구현하기
교과목 교수	최 재 영
제출인	컴퓨터학부(학과) 학번: 20232872 성명: 김도원(출석번호:102)
제출일	2025년 6월 6일

차 례

1장 동기/목적

2장 설계/구현 아이디어

3장 수행결과

4장 결론 및 보충할 점

5장 소스코드(+주석)

1장 동기/목적

SIC/XE ObjectCode 시뮬레이터를 개발한다. 이전 프로젝트에서 제작한 SIC/XE 어셈블러들의 출력물로 나오는 ObjectCode를 이 프로젝트의 인풋으로 넣어 프로세스를 시뮬레이션한다. 시뮬레이션 과정은 GUI로 보여주며 이를 구현하는 Java 프로그램 단에서 메모리, 레지스터, 입출력 장치, 로더들을 가상으로 정의하고 사용한다. 이 프로젝트를 통해서 SIC/XE 명령어의 동작을 이해한다. 프로젝트 구현 과정에서 객체 지향 프로그래밍과 자바 프로그래밍을 학습할 수 있다.

2장 설계/구현 아이디어

우선 프로세스 실행과 과정 및 결과를 GUI로 보여주기 위해 GUI 모듈을 제작한다. 그다음 가상의 메모리 공간을 잡고 로더로 SIC/XE 오브젝트 코드의 텍스트 영역을 해당 공간에 올린다. 로드 후에는 Modification record로 지정된 메모리 위치의 값을 수정한다. GUI를 통해 프로세스를 실행하면 oneStep의 경우에는 가상 메모리의 명령어들을 하나씩 분리한다. 연산 모듈은 이 명령어를 분석해서 연산자와 피연산자, nixbpe에 따라 동작을 수행한다. 연산 과정에서 쓰이는 레지스터와 입출력 장치는 가상 장치 모듈에 구현한다. 실제 입출력 장치는 파일 입출력 방식으로 대체한다. 명령어 실행마다 결과를 GUI에 업데이트하고 다음 GUI 버튼 이벤트를 대기한다. allStep의 경우에는 이 oneStep을 프로세스 끝까지 반복한다.

설계/구현한 클래스는 총 10개이며 각각의 역할은 다음과 같다.

VisualSimulator : SIC/XE 시뮬레이터의 GUI를 표현한다. 프로그램 실행, 레지스터 상태, 명령어 로그, 헤더 정보 등의 프레임을 생성하고 명령어 실행마다 업데이트된 값을 GUI에 표시한다.

SicSimulator : 시뮬레이터의 핵심 역할인 메인 로직을 실행한다. 명령어를 한 줄씩 실행하는 oneStep, 모든 명령어를 한꺼번에 실행하는 allStep 기능을 제공한다. 스텝마다 가상의 메모리 공간을 읽어서 하나의 명령어 코드를 가져와서 InstLuncher를 통해 연산을 수행한다.

Program : 시뮬레이터가 로드한 프로그램의 정보를 보관하는 역할을 한다. 프로그램명, 시작 주소, 명령어 리스트, 수정 레코드 등을 포함한다. 이 정보들은 GUI의 헤더 영역에 표시된다.

Modification : 로딩된 프로그램의 수정 레코드 정보를 저장하는 역할을 한다. 수정할 주소와 길이 정보를 가지고 있으며, SicSimulator가 로드 시 이 정보를 참조해 특정 메모리값을 수정한다.

Command : 실제 실행 가능한 명령어 한 줄을 표현하는 구조체 역할을 한다. 위치, 코드, 포맷, 연산자, 피연산자 등의 정보를 담고 있다. 명령어 분석 및 실행 시 해당 정보를 기반으로 디코딩하고, Operator 클래스와 함께 처리된다.

Operator : SIC/XE 명령어 셋을 정의하고 있으며, 각 연산자의 opcode, 포맷, 이름 등을 보유하고 있다. 이 클래스는 명령어 실행 시 연산자 정보를 참조하거나 검증하는 데 사용된다.

SicLoader : SIC/XE 프로그램의 로딩 및 링킹 과정을 담당한다. 오브젝트 코드 파일을 분석하여 메모리에 적재하고 실행에 필요한 정보를 설정한다. 생성자에서 SicSimulator와 연결하여 로딩 결과를 시뮬레이터에 반영한다.

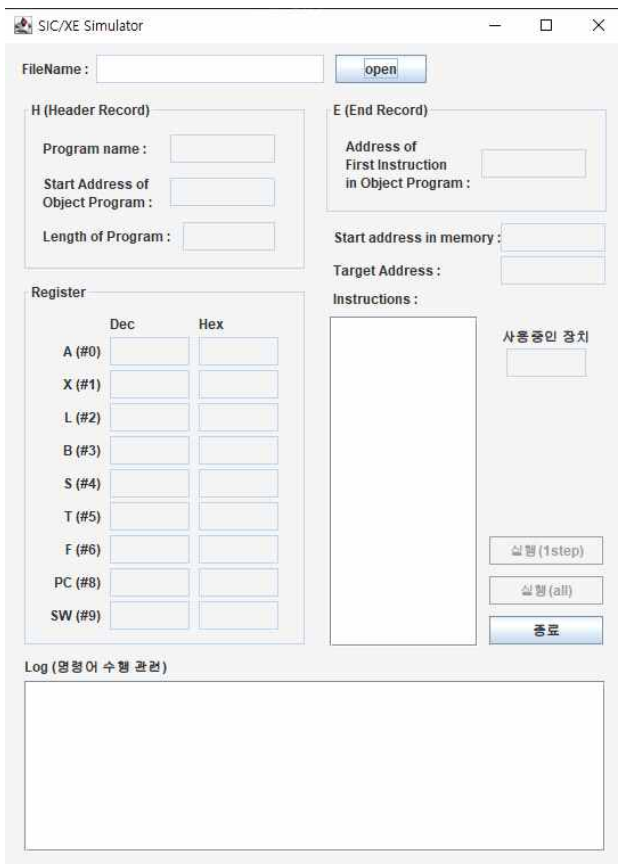
ResourceManager : 레지스터, 메모리, 입출력 장치를 통합적으로 관리한다. 64kb의 전체 메모리 블록과 SIC/XE 레지스터를 유지하며, 특정 메모리 주소의 값을 읽고 쓰는 기능과, 각 레지스터에 접근하여 값을 설정하거나 가져오는 기능을 제공합니다. 프로그램 실행 중 명령어 처리 시 SicSimulator와 InstLuncher 에서 이 클래스를 활용해 필요한 값을 읽고 조작하게 된다.

SymbolTable : 심볼 테이블을 다루는 클래스다. 심볼(label)과 해당 주소(location)를 저장하고 검색한다. 외부 참조 심볼(EXTREF)을 따로 저장해두었다가 다른 프로그램의 EXTDEF를 읽으면서 심볼과 주소값을 저장한다.

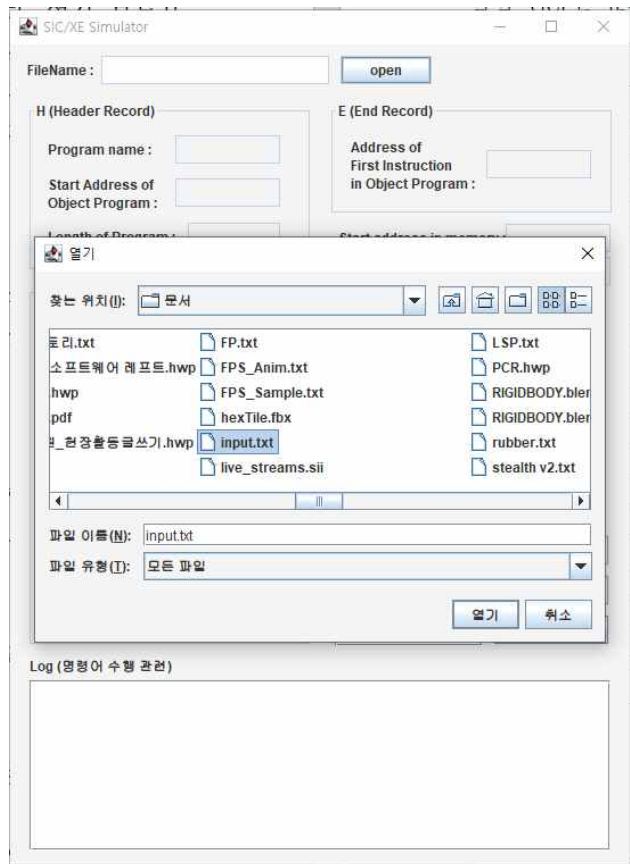
InstLuncher : 명령어(Command) 객체를 분석하고 해당 명령어에 맞는 실행 로직을 수행하는 기능을 담당한다. 이 클래스는 시뮬레이터의 자원(메모리, 레지스터, 장치 등)을 조작하거나 제어 분기, 시각적 반응을 수행한다.

3장 수행결과

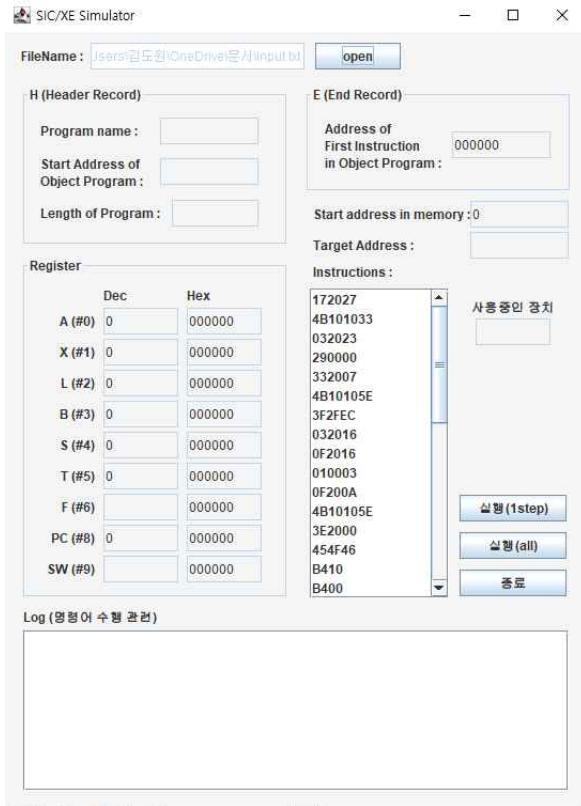
GUI의 파일 탐색 기능으로 오브젝트 코드 파일을 선택하면 해당 파일을 읽어서 프로그램을 실행한다. 명령어 수행마다 헤더, 레지스터, 명령어 로그, target address 등의 프로그램이랑 리소스 정보들을 GUI로 표시한다. 예시로 실행할 오브젝트 코드는 입력 장치의 데이터들을 출력 장치로 복사하는 역할을 한다.



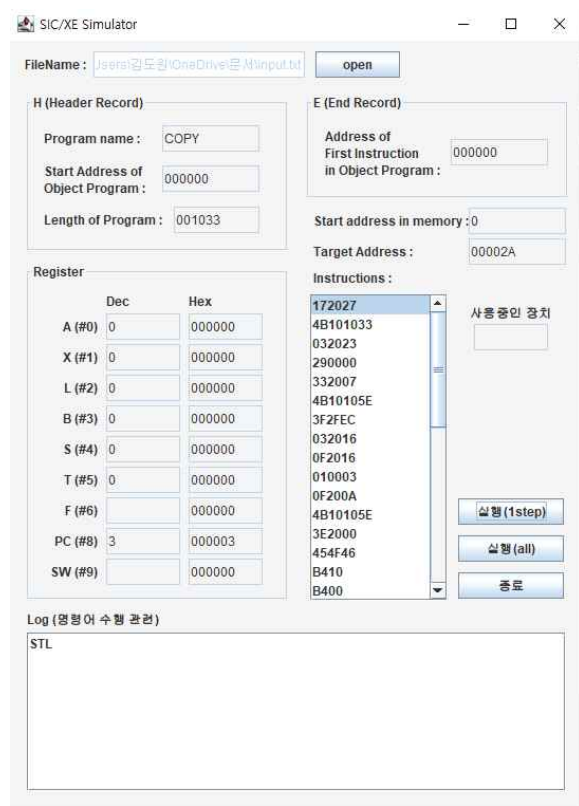
<프로그램 실행 시 초기 GUI 화면>



<open 버튼으로 파일 선택 화면>



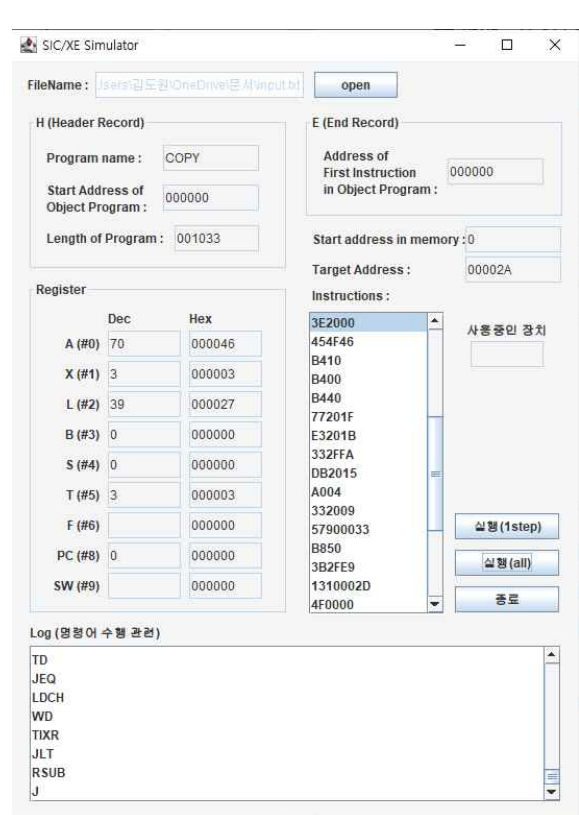
<파일 선택 후 명령어 로드 화면>



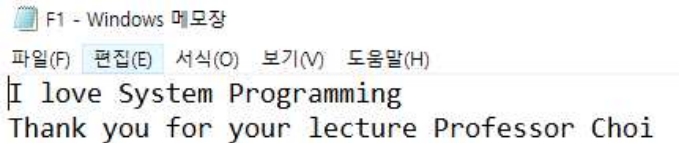
<step 버튼으로 한 명령어 실행 후 화면>



<여러 step 버튼 클릭 후 화면>

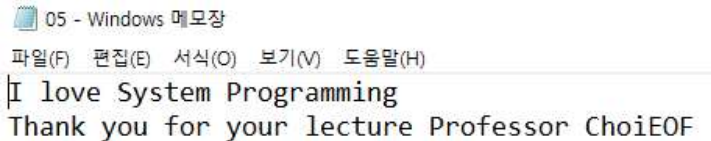


<all 버튼으로 프로그램 전체 실행 후 화면>



F1 - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
I love System Programming
Thank you for your lecture Professor Choi

<F1 : input device 대행 파일>



05 - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
I love System Programming
Thank you for your lecture Professor ChoiE0F

<05 : output device 대행 파일>

4장 결론 및 보충할 점

VisualSimulator, SicSimulator, Program, Modification, Command, Operator, SicLoader, ResourceManager, SymbolTable, InstLuncher 클래스들을 활용해서 명령어 로드와 실행을 거치는 과정을 GUI를 통해 보여주었다. 최종적으로 오브젝트 코드의 동작인 디바이스 간의 복사 프로세스를 마쳤다. 이 프로젝트를 완성하고 나서 추가 클래스 정의에서 보충할 점을 느꼈다.

4.1. Program 클래스 추가

GUI의 Header Record 부분에는 현재 수행 중인 프로그램의 정보가 띄어져야 한다. 헤더 정보를 보여줘야 할 때마다 매번 오브젝트 코드의 헤더 부분을 읽고 필요한 정보를 파싱하기에는 번거롭다. 그래서 오브젝트 코드를 로드하면서 헤더 부분의 프로그램 정보를 한 번만 추출해서 인스턴스 화한다. 그러면 저장된 인스턴스를 재사용하면서 헤더 정보를 보여주기가 편하다.

4.2. Modification 클래스 추가

수정 레코드를 통한 메모리 값 변경이 이뤄져야 한다. 하지만 한 프로그램의 오브젝트 코드 로드 후 곧바로 수정 레코드를 통한 메모리 변경을 시도한다면 다른 프로그램의 심볼은 저장되기 전이어서 문제가 생길 수 있다. 따라서 이 클래스를 선언해서 코드를 적재하는 중에 수정 레코드를 읽어서 수정 정보를 저장해서 인스턴스화 한다. 그리고 모든 프로그램의 로드가 끝난 후, 즉 프로그램마다 심볼이 다 저장되고 나서 Modification 인스턴스들을 순회하면 수정 레코드가 의도한 메모리 수정이 이루어지게 된다.

4.3. Command 클래스 추가

프로그램 실행 중에는 작동을 위해 명령어 파싱을 진행해야 한다. 적재된 메모리를 읽은 후 명령어 코드의 정보인 opcode, 형식, 코드, 위치 등의 정보를 한꺼번에 저장할 수 있는 Command 클래스가 필요하다. 연산을 수행하는 InstLuncher에서 해당 클래스 인스턴스를 받고 인스턴스의 여러 정보를 분석해서 그에 맞는 연산을 수행하게 된다.

5장 소스코드(+주석)

```
package SP25_simulator;

import java.awt.EventQueue;
import javax.swing.*.*;
import java.awt.*.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.List;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.AbstractMap;

/**
 * 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 {
    //리소스 매니저, 시뮬레이터, 로더
    ResourceManager resourceManager = new ResourceManager();
    SicSimulator sicSimulator = new SicSimulator(resourceManager, this);
    SicLoader sicLoader = new SicLoader(resourceManager, sicSimulator);

    //오브젝트 코드 명령어와 위치 목록, 스크롤 목록
    DefaultListModel<String> commandsModel;
    ArrayList<Integer> commandLocList;
    JList<String> commandScrollList;

    //oneStep, allStep 버튼
    JButton stepInstallBtn;
```

```

JButton allInstallBtn;

//레지스터 이름 -> 레지스터 인덱스 매핑
HashMap<String, Integer> registerField = new HashMap<String, Integer>();
//레지스터 10진수 목록
JTextField[] registerDecFields;
//레지스터 16진수 목록
JTextField[] registerHexFields;

//헤더 정보 : 프로그램 이름, 시작 위치, 길이
JTextField programNameField;
JTextField startAddrField;
JTextField lengthField;

//end record 정보
JTextField instAddressField;

//“Start address in memory“ 필드
JTextField memoryStartField;

//“Target address“ 필드
JTextField targetAddrField;

//명령어 로그 관련 목록 및 스크롤
DefaultListModel<String> logModel;
JList<String> logList;

//사용중인 장치 필드
JTextField deviceField;

//레지스터 인덱스 매핑 초기화 및 레지스터 배열 초기화
public VisualSimulator() {
    registerField.put("A", 0);
    registerField.put("X", 1);
    registerField.put("L", 2);
    registerField.put("B", 3);
    registerField.put("S", 4);
    registerField.put("T", 5);
}

```



```

        registerField.put("F", 6);
        registerField.put("PC", 7);
        registerField.put("SW", 8);

        registerDecFields = new JTextField[registerField.size()];
        registerHexFields = new JTextField[registerField.size()];
        commandLocList = new ArrayList<>();
    }

    /**
     * 프로그램 로드 명령을 전달한다.
     */
    public void load(File program) {
        // ...
        sicLoader.load(program);
        sicSimulator.load(program);
        showAllRegisters();
        instAddressField.setText(String.format("%06X", sicSimulator.loadLocation));
        memoryStartField.setText(String.format("%X", sicSimulator.loadLocation));
    };

    /**
     * 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() {

```

```
};
```

```
//GUI 요소와 프레임 생성
```

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

```
    VisualSimulator visualSimulator = new VisualSimulator();
```

```
    //타이틀 프레임
```

```
    JFrame frame = new JFrame("SIC/XE Simulator");
```

```
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
    frame.setSize(560, 765);
```

```
    frame.setLocationRelativeTo(null);
```

```
    frame.setLayout(null); // 절대 좌표 배치
```

```
//파일 이름 레이블 생성
```

```
JLabel label = new JLabel("FileName :");
```

```
label.setBounds(15, 10, 60, 25);
```

```
frame.add(label);
```

```
//파일 이름 텍스트 필드 생성
```

```
JTextField textField = new JTextField();
```

```
textField.setBounds(80, 10, 200, 25);
```

```
textField.setEnabled(false);
```

```
frame.add(textField);
```

```
//파일 탐색 버튼 생성
```

```
JButton button = new JButton("open");
```

```
button.setBounds(290, 10, 80, 25);
```

```
frame.add(button);
```

```
//파일 탐색 버튼 클릭 시 동작
```

```
button.addActionListener(e -> {
```

```
    JFileChooser fileChooser = new JFileChooser();
```

```
    int result = fileChooser.showOpenDialog(frame); //파일 다이얼로그 열기
```

```
    if (result == JFileChooser.APPROVE_OPTION) {
```

```
        File selectedFile = fileChooser.getSelectedFile();
```

```
        textField.setText(selectedFile.getAbsolutePath());
```

```

        visualSimulator.load(selectedFile);
        visualSimulator.appendCommands();
        visualSimulator.stepInstallBtn.setEnabled(true);
        visualSimulator.allInstallBtn.setEnabled(true);
    }
});

```

//여러 하위 프레임들 추가

```

frame.add(visualSimulator.makeEndPanel());
frame.add(visualSimulator.makeHeaderPanel());
frame.add(visualSimulator.makeRegisterPanel());
frame.add(visualSimulator.makeAddressPanel());
frame.add(visualSimulator.makeInstList(), BorderLayout.WEST);
frame.add(visualSimulator.makeControlButtons());

```

//명령어 로그 프레임 생성

```

JLabel logLabel = new JLabel("Log (명령어 수행 관련)");
logLabel.setBounds(17, 535, 150, 25);
frame.add(logLabel);
visualSimulator.logModel = new DefaultListModel<>();
visualSimulator.logList = new JList<>(visualSimulator.logModel);
visualSimulator.logList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
JScrollPane logListScrollPane = new JScrollPane(visualSimulator.logList);
logListScrollPane.setBounds(17, 560, 150, 150);
frame.add(logListScrollPane, BorderLayout.WEST);

```

//최상단의 GUI 프레임 표시

```

frame.setVisible(true);
}

```

//명령어 코드 목록에 로드된 명령어 코드 추가

```

private void appendCommands() {
    ArrayList<Command> commandList =
sicSimulator.parseCommands(resourceManager.getMemory(0, resourceManager.offset));
    SwingUtilities.invokeLater() -> {
        for (Command command : commandList) {
            commandsModel.addElement(command.toString());
            commandLocList.add(command.getLocation());
        }
    }
}

```

```

    }
});
}

```

//헤더 레코드 영역 패널 생성

```
private JPanel makeHeaderPanel() {
```

```
    //그리드 형식의 패널 생성
```

```
    JPanel hPanel = new JPanel(new GridBagLayout());
```

```
    hPanel.setBorder(BorderFactory.createTitledBorder("H (Header Record)"));
```

```
    GridBagConstraints hGbc = new GridBagConstraints();
```

```
    hGbc.insets = new Insets(5, -5, 5, 5);
```

```
    hGbc.anchor = GridBagConstraints.WEST;
```

//레이블 생성

```
hGbc.gridx = 0; hGbc.gridy = 0;
```

```
hPanel.add(new JLabel("Program name :"), hGbc);
```

//텍스트 필드 생성

```
hGbc.gridx = 1; hGbc.gridy = 0; hGbc.anchor = GridBagConstraints.EAST;
```

```
programNameField = new JTextField(8);
```

```
programNameField.setPreferredSize(new  
Dimension(programNameField.getPreferredSize().width, 25));
```

```
programNameField.setEditable(false);
```

```
hPanel.add(programNameField, hGbc);
```

//레이블 생성

```
hGbc.gridx = 0; hGbc.gridy = 1; hGbc.anchor = GridBagConstraints.WEST;
```

```
hPanel.add(new JLabel("<html>Start Address of<br>Object Program :</html>"), hGbc);
```

//텍스트 필드 생성

```
hGbc.gridx = 1; hGbc.gridy = 1; hGbc.anchor = GridBagConstraints.EAST;
```

```
startAddrField = new JTextField(8);
```

```
startAddrField.setPreferredSize(new  
25));
```

```
startAddrField.setEditable(false);
```

```
hPanel.add(startAddrField, hGbc);
```

//레이블 생성

```
hGbc.gridx = 0; hGbc.gridy = 2; hGbc.anchor = GridBagConstraints.WEST;
hPanel.add(new JLabel("Length of Program :"), hGbc);
```

```
//텍스트 필드 생성
```

```
hGbc.gridx = 1; hGbc.gridy = 2; hGbc.anchor = GridBagConstraints.EAST;
lengthField = new JTextField(7);
lengthField.setPreferredSize(new Dimension(lengthField.getPreferredSize().width, 25));
lengthField.setEditable(false);
hPanel.add(lengthField, hGbc);
```

```
hPanel.setBounds(15, 50, 250, 150);
return hPanel;
}
```

```
//end 레코드 영역 패널 생성
```

```
private JPanel makeEndPanel() {
```

```
    //그리드 형식의 패널 생성
```

```
    JPanel ePanel = new JPanel(new GridBagLayout());
    ePanel.setBorder(BorderFactory.createTitledBorder("E (End Record)"));
    GridBagConstraints eGbc = new GridBagConstraints();
    eGbc.insets = new Insets(5, 5, 5, 5);
    eGbc.anchor = GridBagConstraints.WEST;
```

```
//레이블 생성
```

```
eGbc.gridx = 0; eGbc.gridy = 0;
ePanel.add(new JLabel("<html>Address of<br>First Instruction<br>in Object Program :
</html>"), eGbc);
```

```
//텍스트 필드 생성
```

```
eGbc.gridx = 1; eGbc.gridy = 0; eGbc.anchor = GridBagConstraints.EAST;
instAddressField = new JTextField(8);
instAddressField.setPreferredSize(new
Dimension(instAddressField.getPreferredSize().width, 25));
instAddressField.setEditable(false);
ePanel.add(instAddressField, eGbc);

ePanel.setBounds(280, 50, 250, 100);
return ePanel;
```

```
}
```

```
private JPanel makeRegisterPanel() {
```

```
    //그리드 형식의 패널 생성
```

```
    JPanel rPanel = new JPanel(new GridBagLayout());
```

```
    rPanel.setBorder(BorderFactory.createTitledBorder("Register"));
```

```
    GridBagConstraints rGbc = new GridBagConstraints();
```

```
    rGbc.insets = new Insets(2, 4, 2, 4);
```

```
    rGbc.anchor = GridBagConstraints.WEST;
```

```
    //타이틀 레이블 생성
```

```
    rGbc.gridx = 1; rGbc.gridy = 0;
```

```
    rPanel.add(new JLabel("Dec"), rGbc);
```

```
    rGbc.gridx = 2; rGbc.gridy = 0;
```

```
    rPanel.add(new JLabel("Hex"), rGbc);
```

```
    int yPos = 1;
```

```
    // <레지스터 번호, 레지스터 이름> 페어로 레지스터별 영역 생성
```

```
    for (Map.Entry<Integer, String> entry : resourceManager.registerNum.entrySet()) {
```

```
        //레이블 생성
```

```
        rGbc.gridx = 0; rGbc.gridy = yPos; rGbc.anchor = GridBagConstraints.EAST;
```

```
        JLabel regLabel = new JLabel(String.format("%s (%#d)", entry.getValue(),  
entry.getKey()));
```

```
        rPanel.add(regLabel, rGbc);
```

```
        //10진수 텍스트 필드 생성
```

```
        rGbc.gridx = 1; rGbc.gridy = yPos; rGbc.anchor = GridBagConstraints.EAST;
```

```
        JTextField decField = new JTextField(6);
```

```
        decField.setPreferredSize(new Dimension(decField.getPreferredSize().width, 25));
```

```
        decField.setEditable(false);
```

```
        rPanel.add(decField, rGbc);
```

```
        registerDecFields[yPos - 1] = decField;
```

```
        //16진수 텍스트 필드 생성
```

```
        rGbc.gridx = 2; rGbc.gridy = yPos; rGbc.anchor = GridBagConstraints.EAST;
```

```
        JTextField hexField = new JTextField(6);
```

```
        hexField.setPreferredSize(new Dimension(hexField.getPreferredSize().width, 25));
```

```
        hexField.setEditable(false);
```

```

        rPanel.add(hexField, rGbc);
        registerHexFields[yPos - 1] = hexField;
        yPos++;
    }

```

```

rPanel.setBounds(15, 210, 250, 320);
return rPanel;
}

```

```

private JPanel makeAddressPanel() {

```

```

    //그리드 형식의 패널 생성

```

```

    JPanel aPanel = new JPanel(new GridBagLayout());

```

```

    GridBagConstraints aGbc = new GridBagConstraints();

```

```

    aGbc.insets = new Insets(2, 0, 2, 0);

```

```

    aGbc.anchor = GridBagConstraints.WEST;

```

```

    //레이블 생성

```

```

    aGbc.gridx = 0; aGbc.gridy = 0; aGbc.anchor = GridBagConstraints.WEST;

```

```

    aPanel.add(new JLabel("Start address in memory :"), aGbc);

```

```

    //텍스트 필드 생성

```

```

    aGbc.gridx = 1; aGbc.gridy = 0; aGbc.anchor = GridBagConstraints.EAST;

```

```

    memoryStartField = new JTextField(8);

```

```

    memoryStartField.setPreferredSize(new
Dimension(startAddrField.getPreferredSize().width, 25));

```

```

    memoryStartField.setEditable(false);

```

```

    aPanel.add(memoryStartField, aGbc);

```

```

    //레이블 생성

```

```

    aGbc.gridx = 0; aGbc.gridy = 1; aGbc.anchor = GridBagConstraints.WEST;

```

```

    aPanel.add(new JLabel("Target Address :"), aGbc);

```

```

    //텍스트 필드 생성

```

```

    aGbc.gridx = 1; aGbc.gridy = 1; aGbc.anchor = GridBagConstraints.EAST;

```

```

    targetAddrField = new JTextField(8);

```

```

    targetAddrField.setPreferredSize(new Dimension(targetAddrField.getPreferredSize().width,
25));

```

```

    targetAddrField.setEditable(false);

```

```
aPanel.add(targetAddrField, aGbc);
```

```
//스크롤 목록의 레이블 생성
```

```
aGbc.gridx = 0; aGbc.gridy = 2; aGbc.anchor = GridBagConstraints.WEST;
```

```
aPanel.add(new JLabel("Instructions :"), aGbc);
```

```
aPanel.setBounds(280, 150, 255, 90);
```

```
return aPanel;
```

```
}
```

```
private JScrollPane makeInstList() {
```

```
    //그리드 형식의 패널 생성
```

```
    GridBagConstraints iGbc = new GridBagConstraints();
```

```
    iGbc.insets = new Insets(0, 0, 2, 0);
```

```
    iGbc.anchor = GridBagConstraints.WEST;
```

```
//명령어 코드 스크롤 목록 생성
```

```
commandsModel = new DefaultListModel<>();
```

```
commandScrollList = new JList<>(commandsModel);
```

```
commandScrollList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
JScrollPane scrollPane = new JScrollPane(commandScrollList);
```

```
scrollPane.setBounds(285, 240, 130, 290);
```

```
return scrollPane;
```

```
}
```

```
private JPanel makeControlButtons() {
```

```
    //그리드 형식의 패널 생성
```

```
    JPanel ePanel = new JPanel(new GridBagLayout());
```

```
    GridBagConstraints eGbc = new GridBagConstraints();
```

```
    eGbc.insets = new Insets(3, 0, 3, 0);
```

```
    eGbc.anchor = GridBagConstraints.CENTER;
```

```
//레이블 생성
```

```
eGbc.gridx = 0; eGbc.gridy = 0;
```

```
ePanel.add(new JLabel("사용중인 장치"), eGbc);
```


//텍스트 필드 생성

```
eGbc.gridx = 0; eGbc.gridy = 1; eGbc.anchor = GridBagConstraints.CENTER;
eGbc.insets = new Insets(0, 0, 135, 0);
deviceField = new JTextField(6);
deviceField.setHorizontalAlignment(JTextField.CENTER);
deviceField.setPreferredSize(new Dimension(deviceField.getPreferredSize().width, 25));
deviceField.setEditable(false);
ePanel.add(deviceField, eGbc);
```

//oneStep 버튼 생성

```
eGbc.gridx = 0; eGbc.gridy = 2; eGbc.anchor = GridBagConstraints.CENTER;
eGbc.insets = new Insets(5, 0, 5, 0);
stepInstallBtn = new JButton("실행(1step)");
stepInstallBtn.setPreferredSize(new Dimension(100, 25));
stepInstallBtn.setEnabled(false);
ePanel.add(stepInstallBtn, eGbc);
```

//oneStep 버튼 클릭 이벤트

```
stepInstallBtn.addActionListener(e -> {
    oneStep();
});
```

//allStep 버튼 생성

```
eGbc.gridx = 0; eGbc.gridy = 3; eGbc.anchor = GridBagConstraints.CENTER;
allInstallBtn = new JButton("실행(all)");
allInstallBtn.setPreferredSize(new Dimension(100, 25));
allInstallBtn.setEnabled(false);
ePanel.add(allInstallBtn, eGbc);
```

//allStep 버튼 클릭 이벤트

```
allInstallBtn.addActionListener(e -> {
    allStep();
});
```

//종료 버튼 생성

```
eGbc.gridx = 0; eGbc.gridy = 4; eGbc.anchor = GridBagConstraints.CENTER;
JButton exitBtn = new JButton("종료");
exitBtn.setPreferredSize(new Dimension(100, 25));
ePanel.add(exitBtn, eGbc);
```

```

//종료 버튼 클릭 이벤트
exitBtn.addActionListener(e -> {
    System.exit(0);
});

ePanel.setBounds(420, 235, 110, 310);
return ePanel;
}

```

```

//리소스 매니저의 모든 레지스터값 불러와서 GUI에 업데이트
public void showAllRegisters() {
    for (String register : registerField.keySet()) {
        showRegisterValue(register, resourceManager.registers.get(register));
    }
}

```

```

//지정한 레지스터의 값만 불러와서 GUI에 업데이트
public void showRegisterValue(String name, int value) {
    int regidx = registerField.get(name);
    if(!name.equals("F") && !name.equals("SW"))
        registerDecFields[regidx].setText(Integer.toString(value));
    registerHexFields[regidx].setText(String.format("%06X", value));
}

```

```

//프로그램 헤더 정보 불러와서 GUI에 업데이트
public void showHeader(Program program) {
    programNameField.setText(program.getName());
    startAddrField.setText(String.format("%06X", program.getStart()));
    lengthField.setText(String.format("%06X", program.getLength()));
}

```

```

//target address 업데이트
public void showTargetAddr() {
    targetAddrField.setText(String.format("%06X",
resourceManager.getRegisterValue("TA")));
}

```

```

//명령어 실행 로그 추가
public void addOperatorLog(String operator) {
    logModel.addElement(operator);
    logList.ensureIndexIsVisible(logModel.size() - 1);
}

```

```

//실행중인 명령어 코드를 스크롤 목록에서 선택 표시
public void selectCommand(Command command) {
    for(int i = 0; i < commandsModel.size(); ++i) {
        if(commandLocList.get(i) == command.getLocation()) {
            commandScrollList.setSelectedIndex(i);
            commandScrollList.ensureIndexIsVisible(i);
        }
    }
}

```

```

//사용중인 장치 정보 업데이트
public void setUsingDevice(String device) {
    if(device == null)
        deviceField.setText("");
    else
        deviceField.setText(device);
}

```

```

}

```

```

package SP25_simulator;

import java.io.File;
import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Iterator;

import javax.management.modelmbean.ModelMBeanOperationInfo;
import javax.xml.transform.Source;

/**
 * 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 또
 * 한 허용됨. 3) 모든 void 타입의 리턴값은
 * 유저의 필요에 따라 다른 리턴 타입으로 변경 가능. 4) 파일, 또는 콘솔창에 한글을 출력시
 * 키지 말 것. (채점상의 이유. 주석에 포함된
 * 한글은 상관 없음)
 *
 *
 * + 제공하는 프로그램 구조의 개선방법을 제안하고 싶은 분들은 보고서의 결론 뒷부분에 첨
 * 부 바랍니다. 내용에 따라 가산점이 있을 수
 * 있습니다.
 */
public class SicSimulator {
    ResourceManager rMgr;
    VisualSimulator vsim;
    InstLuncher instLuncher;
    ArrayList<Program> sections;
    Program currentSection;
    int loadLocation = -1;
    boolean isProcessing = true;

```

```

public SicSimulator(ResourceManager resourceManager, VisualSimulator visualSimulator)
{
    // 필요하다면 초기화 과정 추가
    sections = new ArrayList<>();
    this.rMgr = resourceManager;
    this.vsim = visualSimulator;
    instLuncher = new InstLuncher(resourceManager, vsim);
}

/**
 * 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 */

    //프로그램의 EXTREF를 다른 프로그램의 EXTDEF를 읽어서 절대 위치를 저장
    for(Program section : sections) {
        for(String extref : section.getExtrefs()) {
            for(Program otherSection : sections) {
                if(section.getName().equals(otherSection.getName()))
                    continue;

                int location = otherSection.searchSymbol(extref);
                if(location >= 0) {
                    section.addSymbol(extref, location);
                    break;
                }
            }
        }
    }

    //프로그램의 Modification record를 읽어서 메모리값 변경
    for(Program section : sections) {
        for(Modification mod : section.getModifications()) {
            int start = section.getStart() + mod.getOffset();
            int modByteSize = mod.getLength() / 2;
            if(mod.getLength() % 2 == 1)

```

```

        modByteSize += 1;

        int location = section.searchSymbol(mod.getSymbol());
        long mask = (1L << (mod.getLength() * 4)) - 1;
        location = location & (int)mask;
        byte[] addition = new byte[modByteSize];

        for(int i = 0; i < modByteSize; ++i) {
            addition[i] = (byte)((location >> 8 * (modByteSize - 1 - i)) &
0xFF);
        }
        if(mod.getIsNegative())
            rMgr.subMemory(start, addition, modByteSize);
        else {
            rMgr.addMemory(start, addition, modByteSize);
        }
    }
}

/**
 * 1개의 instruction이 수행된 모습을 보인다.
 */
public void oneStep() {
    if(isProcessing == false)
        return;

    //PC 불러오고 명령어 길이만큼 더하기
    int pc = rMgr.getRegisterValue("PC");
    byte[] bytes = rMgr.getMemory(pc, 4);
    Command command = new Command(bytes, pc);
    rMgr.addRegisterValue("PC", command.getLength() / 2);

    //명령어 실행
    isProcessing = instLuncher.processOperator(command);

    //현재 실행중인 프로그램 탐색
    for(Program program : sections) {
        int programStart = program.getStart();
        int programEnd = program.getStart() + program.getLength();

```

```

        if(pc >= programStart && pc < programEnd) {
            currentSection = program;
            break;
        }
    }

    //GUI 업데이트
    String operatorString = Command.operators.get(command.getOperator()).name;
    vsim.selectCommand(command);
    vsim.addOperatorLog(operatorString);
    vsim.showHeader(currentSection);
    vsim.showAllRegisters();
    vsim.showTargetAddr();
}

/**
 * 남은 모든 instruction이 수행된 모습을 보인다.
 */
public void allStep() {
    //프로세스가 끝날때까지 oneStep 실행
    while(isProcessing) {
        oneStep();
    }
}

/**
 * 각 단계를 수행할 때 마다 관련된 기록을 남기도록 한다.
 */
public void addLog(String log) {
}

//메모리에서 받은 바이트들로 다수의 명령어 코드 추출
public ArrayList<Command> parseCommands(byte[] bytes) {
    if(bytes == null || bytes.length <= 2)
        return null;

    ArrayList<Command> commandList = new ArrayList<Command>();

```

```

        int i = 0;
        while(i < bytes.length) {
            Command command = new Command(Arrays.copyOfRange(bytes, i,
bytes.length), i);

            if(command.getLength() <= 0) {
                ++i;
                continue;
            }
            commandList.add(command);
            i += command.getLength() / 2;
        }
        return commandList;
    }
}

```

```

//새로운 프로그램의 end record를 읽었을때 해당 프로그램 추가
public void addProgram(Program program) {
    sections.add(program);
    if(currentSection == null)
        currentSection = program;
}
}

```

```

//프로그램(섹션) 클래스
class Program {
    private String name;
    private int start;
    private int length;
    private SymbolTable symTable;
    private ArrayList<Modification> modifications;

    Program(String name, int start, int length) {
        modifications = new ArrayList<>();
        symTable = new SymbolTable();
        this.name = name;
    }
}

```



```

        this.start = start;
        this.length = length;
        addSymbol(name, start);
    }

    public String getName() {
        return name;
    }

    public int getStart() {
        return start;
    }

    public int getLength() {
        return length;
    }

    public ArrayList<Modification> getModifications() {
        return modifications;
    }

    public ArrayList<String> getExtrefs() {
        return symTable.extrefList;
    }

    //심볼 추가
    public void addSymbol(String symbol, int absLocation) {
        symTable.putSymbol(symbol, absLocation);
    }

    //심볼 탐색
    public int searchSymbol(String symbol) {
        return symTable.search(symbol);
    }

    //수정 레코드 추가
    public void addModification(Modification mod) {
        modifications.add(mod);
    }

```

```
//EXTREF 추가
public void addExtref(String extref) {
    symTable.putExternalref(extref);
}
}
```

//수정 레코드 클래스

```
class Modification {
    private int offset;
    private int length;
    private boolean isNegative;
    private String symbol;

    public Modification(int offset, int length, boolean isNegative, String symbol) {
        this.offset = offset;
        this.length = length;
        this.isNegative = isNegative;
        this.symbol = symbol;
    }

    public int getOffset() {
        return offset;
    }

    public int getLength() {
        return length;
    }

    public boolean getIsNegative() {
        return isNegative;
    }

    public String getSymbol() {
        return symbol;
    }
}
```

//명령어 클래스

```
class Command {
```

```
    private int data;
```

```
    private int length;
```

```
    private byte operator;
```

```
    private int location;
```

```
    //명령어 opcode와 명령어 정보 매핑
```

```
    static HashMap<Byte, Integer> formats;
```

```
    static HashMap<Byte, Operator> operators;
```

```
    //nixbpe 플래그
```

```
    final static int N_FLAG = 5;
```

```
    final static int I_FLAG = 4;
```

```
    final static int X_FLAG = 3;
```

```
    final static int B_FLAG = 2;
```

```
    final static int P_FLAG = 1;
```

```
    final static int E_FLAG = 0;
```

```
    //메모리에서 읽은 바이트들로 하나의 명령어 생성
```

```
    public Command(byte[] bytes, int offset) {
```

```
        if(bytes == null || bytes.length <= 1)
```

```
            return;
```

```
        operator = parseOperator(bytes);
```

```
        Integer format = formats.get(operator);
```

```
        if(format == null)
```

```
            return;
```

```
        byte second_byte = bytes[1];
```

```
        if(isRightNthBitSet(bytes[0], 0) || isRightNthBitSet(bytes[0], 1)) {
```

```
            if(format != 3)
```

```
                return;
```

```

        if(isRightNthBitSet(second_byte, 4))
            length = 8;
        else {
            length = 6;
        }
    }
    else if(format == 2) {
        length = 4;
    }
    for(int i = 0; i < length / 2; ++i) {
        data |= (bytes[i] & 0xFF) << ((length / 2 - 1 - i) * 8);
    }
    this.location = offset;
}

```

```

public int getLength() {
    return length;
}

```

```

public byte getOperator() {
    return operator;
}

```

```

public int getLocation() {
    return location;
}

```

```

public int getData() {
    return data;
}

```

```

//명령어 길이만큼 실제 코드값 data 문자열 반환ㄴ
public String toString() {
    long mask = (1L << (length * 4)) - 1;
    int lower_bits = data & (int)mask;
    return String.format("%0" + length + "X", lower_bits);
}

```

```

//메모리에서 읽은 바이트들로 opcode 파싱
private static byte parseOperator(byte[] bytes) {
    if(bytes == null || bytes.length <= 0)
        return 0;

    byte operatorByte = bytes[0];
    for(int i = 0; i < 2; ++i) {
        operatorByte = clearRightNthBit(operatorByte, i);
    }
    return operatorByte;
}

```

```

//유틸리티, 오른쪽 n번째 비트가 1인지
private static boolean isRightNthBitSet(byte data, int n) {
    byte mask = (byte)(1 << n);
    return (data & mask) != 0;
}

```

```

//유틸리티, 오른쪽 n번째 비트를 0으로
private static byte clearRightNthBit(byte data, int n) {
    byte mask = (byte)(~(1 << n));
    return (byte)(data & mask);
}

```

```

//유틸리티, nixbpe 값이 있는지
public boolean hasFlag(int flag) {
    int flagOffset = (length - 3) * 4 + flag;
    int mask = (1 << flagOffset);
    return (data & mask) != 0;
}

```

//부호 여부에 따른 피연산자값 반환

```
public int getOperandValue(boolean signed) {
    int operandBitCnt = (length - 3) * 4;
    long mask = (1 << operandBitCnt) - 1;
    int operandValue = data & (int)mask;
    if ((operandValue & (1 << (operandBitCnt - 1))) != 0 && signed)
        return operandValue - (1 << operandBitCnt);
    return operandValue;
}
```

//정적 매핑 변수 초기화

```
static {
    formats = new HashMap<>();
    formats.put((byte) 0x18, 3);
    formats.put((byte) 0x58, 3);
    formats.put((byte) 0x90, 2);
    formats.put((byte) 0x40, 3);
    formats.put((byte) 0xB4, 2);
    formats.put((byte) 0x28, 3);
    formats.put((byte) 0x88, 3);
    formats.put((byte) 0xA0, 2);
    formats.put((byte) 0x24, 3);
    formats.put((byte) 0x64, 3);
    formats.put((byte) 0x9C, 2);
    formats.put((byte) 0x3C, 3);
    formats.put((byte) 0x30, 3);
    formats.put((byte) 0x34, 3);
    formats.put((byte) 0x38, 3);
    formats.put((byte) 0x48, 3);
    formats.put((byte) 0x00, 3);
    formats.put((byte) 0x68, 3);
    formats.put((byte) 0x50, 3);
    formats.put((byte) 0x70, 3);
    formats.put((byte) 0x08, 3);
    formats.put((byte) 0x6C, 3);
    formats.put((byte) 0x74, 3);
}
```

```
formats.put((byte) 0x04, 3);
formats.put((byte) 0xD0, 3);
formats.put((byte) 0x20, 3);
formats.put((byte) 0x60, 3);
formats.put((byte) 0x98, 2);
formats.put((byte) 0x44, 3);
formats.put((byte) 0xD8, 3);
formats.put((byte) 0x4C, 3);
formats.put((byte) 0xA4, 2);
formats.put((byte) 0xA8, 2);
formats.put((byte) 0xEC, 3);
formats.put((byte) 0x0C, 3);
formats.put((byte) 0x78, 3);
formats.put((byte) 0x54, 3);
formats.put((byte) 0x80, 3);
formats.put((byte) 0xD4, 3);
formats.put((byte) 0x14, 3);
formats.put((byte) 0x7C, 3);
formats.put((byte) 0xE8, 3);
formats.put((byte) 0x84, 3);
formats.put((byte) 0x10, 3);
formats.put((byte) 0x1C, 3);
formats.put((byte) 0x5C, 3);
formats.put((byte) 0x94, 2);
formats.put((byte) 0xB0, 2);
formats.put((byte) 0xE0, 3);
formats.put((byte) 0x2C, 3);
formats.put((byte) 0xB8, 2);
formats.put((byte) 0xDC, 3);
```

```
operators = new HashMap<>();
operators.put((byte) 0x18, new Operator("ADD", (byte) 0x18, 3));
operators.put((byte) 0x58, new Operator("ADDF", (byte) 0x58, 3));
operators.put((byte) 0x90, new Operator("ADDR", (byte) 0x90, 2));
operators.put((byte) 0x40, new Operator("AND", (byte) 0x40, 3));
operators.put((byte) 0xB4, new Operator("CLEAR", (byte) 0xB4, 2));
operators.put((byte) 0x28, new Operator("COMP", (byte) 0x28, 3));
operators.put((byte) 0x88, new Operator("COMPF", (byte) 0x88, 3));
operators.put((byte) 0xA0, new Operator("COMPR", (byte) 0xA0, 2));
operators.put((byte) 0x24, new Operator("DIV", (byte) 0x24, 3));
```

```
operators.put((byte) 0x64, new Operator("DIVF", (byte) 0x64, 3));
operators.put((byte) 0x9C, new Operator("DIVR", (byte) 0x9C, 2));
operators.put((byte) 0x3C, new Operator("J", (byte) 0x3C, 3));
operators.put((byte) 0x30, new Operator("JEQ", (byte) 0x30, 3));
operators.put((byte) 0x34, new Operator("JGT", (byte) 0x34, 3));
operators.put((byte) 0x38, new Operator("JLT", (byte) 0x38, 3));
operators.put((byte) 0x48, new Operator("JSUB", (byte) 0x48, 3));
operators.put((byte) 0x00, new Operator("LDA", (byte) 0x00, 3));
operators.put((byte) 0x68, new Operator("LDB", (byte) 0x68, 3));
operators.put((byte) 0x50, new Operator("LDCH", (byte) 0x50, 3));
operators.put((byte) 0x70, new Operator("LDF", (byte) 0x70, 3));
operators.put((byte) 0x08, new Operator("LDL", (byte) 0x08, 3));
operators.put((byte) 0x6C, new Operator("LDS", (byte) 0x6C, 3));
operators.put((byte) 0x74, new Operator("LDT", (byte) 0x74, 3));
operators.put((byte) 0x04, new Operator("LDX", (byte) 0x04, 3));
operators.put((byte) 0xD0, new Operator("LPS", (byte) 0xD0, 3));
operators.put((byte) 0x20, new Operator("MUL", (byte) 0x20, 3));
operators.put((byte) 0x60, new Operator("MULF", (byte) 0x60, 3));
operators.put((byte) 0x98, new Operator("MULR", (byte) 0x98, 2));
operators.put((byte) 0x44, new Operator("OR", (byte) 0x44, 3));
operators.put((byte) 0xD8, new Operator("RD", (byte) 0xD8, 3));
operators.put((byte) 0x4C, new Operator("RSUB", (byte) 0x4C, 3));
operators.put((byte) 0xA4, new Operator("SHIFTL", (byte) 0xA4, 2));
operators.put((byte) 0xA8, new Operator("SHIFTR", (byte) 0xA8, 2));
operators.put((byte) 0xEC, new Operator("SSK", (byte) 0xEC, 3));
operators.put((byte) 0x0C, new Operator("STA", (byte) 0x0C, 3));
operators.put((byte) 0x78, new Operator("STB", (byte) 0x78, 3));
operators.put((byte) 0x54, new Operator("STCH", (byte) 0x54, 3));
operators.put((byte) 0x80, new Operator("STF", (byte) 0x80, 3));
operators.put((byte) 0xD4, new Operator("STI", (byte) 0xD4, 3));
operators.put((byte) 0x14, new Operator("STL", (byte) 0x14, 3));
operators.put((byte) 0x7C, new Operator("STS", (byte) 0x7C, 3));
operators.put((byte) 0xE8, new Operator("STSW", (byte) 0xE8, 3));
operators.put((byte) 0x84, new Operator("STT", (byte) 0x84, 3));
operators.put((byte) 0x10, new Operator("STX", (byte) 0x10, 3));
operators.put((byte) 0x1C, new Operator("SUB", (byte) 0x1C, 3));
operators.put((byte) 0x5C, new Operator("SUBF", (byte) 0x5C, 3));
operators.put((byte) 0x94, new Operator("SUBR", (byte) 0x94, 2));
operators.put((byte) 0xB0, new Operator("SVC", (byte) 0xB0, 2));
operators.put((byte) 0xE0, new Operator("TD", (byte) 0xE0, 3));
```



```

        operators.put((byte) 0x2C, new Operator("TIX", (byte) 0x2C, 3));
        operators.put((byte) 0xB8, new Operator("TIXR", (byte) 0xB8, 2));
        operators.put((byte) 0xDC, new Operator("WD", (byte) 0xDC, 3));
    }
}

```

//연산자 클래스

```

class Operator {
    String name;
    byte opcode;
    int format;

    Operator(String name, byte opcode, int format) {
        this.name = name;
        this.opcode = opcode;
        this.format = format;
    }
}

```

```

package SP25_simulator;

```

```

import java.io.*;
import java.util.ArrayList;

```

```

/**

```

* 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;
SicSimulator sicSimulator;
int programStart = 0;
```

```
public SicLoader(ResourceManager resourceManager, SicSimulator sicSimulator) {
    // 필요하다면 초기화
    setResourceManager(resourceManager);
    resourceManager.initializeResource();
    this.sicSimulator = sicSimulator;
}
```

```
/**
 * Loader와 프로그램을 적재할 메모리를 연결시킨다.
 *
 * @param rMgr
 */
```

```
public void setResourceManager(ResourceManager resourceManager) {
    this.rMgr = resourceManager;
}
```

```
/**
 * object code를 읽어서 load과정을 수행한다. load한 데이터는 resourceManager가 관
리하는 메모리에 올라가도록
 * 한다. load과정에서 만들어진 symbol table 등 자료구조 역시 resourceManager에 전
달한다.
```

```
 *
 * @param objectCode 읽어들이는 파일
 */
public void load(File objectCode) {
    ArrayList<String> lines = new ArrayList<>();
    try (BufferedReader reader = new BufferedReader(new
FileReader(objectCode))) {
        String line;
        while ((line = reader.readLine()) != null) {
            lines.add(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```

Program program = null;
int programLength = 0;
for (String line : lines) {
    if(line.length() <= 0)
        continue;
    if(line.charAt(0) == 'H') { //헤더 레코드, 프로그램
        programLength = Integer.parseInt(line.substring(13, 19), 16);
        int programOffset = Integer.parseInt(line.substring(7, 13), 16);
        String programName = line.substring(1, 7).trim();
        program = new Program(programName, programStart +
programOffset, programLength);
    }
    else if(line.charAt(0) == 'T') { //텍스트 레코드, 가상 메모리 공간에
        loadTextCode(line);
    }
    else if(line.charAt(0) == 'M') { //수정 레코드, 수정 레코드 클래스로
        저장하고 로드 후 메모리 변경
        storeMods(line, program);
    }
    else if(line.charAt(0) == 'R') { //EXTREF 레코드, 프로그램 심볼 테이블에 임시 저장
        storeExtrefs(line, program);
    }
    else if(line.charAt(0) == 'D') { //EXTDEF 레코드, 프로그램 심볼 테이블에 위치와 함께 저장
        storeExtdefs(line, program);
    }
    else if(line.charAt(0) == 'E') { //End 레코드, 여태 저장한 프로그램 정보를 시뮬레이터에 추가
        programStart += programLength;
        sicSimulator.addProgram(program);
        if(line.length() > 1) {
            sicSimulator.loadLocation
Integer.parseInt(line.substring(1, 16));
        }
    }
}
}

```

//수정 레코드 줄을 읽어서 수정 레코드 클래스로 파싱

```
private void storeMods(String line, Program program) {  
    int offset = Integer.parseInt(line.substring(1, 7), 16);  
    int length = Integer.parseInt(line.substring(7, 9), 16);  
    boolean isNegative = (line.charAt(9) == '-');  
    String symbol = line.substring(10);  
    program.addModification(new Modification(offset, length, isNegative, symbol));  
}
```

//EXTREF 레코드 줄을 읽어서 프로그램 심볼 테이블에 임시 저장, 추후에 절대위치와 함께 삽입

```
private void storeExtrefs(String line, Program program) {  
    int i = 1;  
    while(i < line.length()) {  
        String name = line.substring(i, i + 6);  
        i += 6;  
        program.addExtref(name.trim());  
    }  
}
```

//EXTDEF 레코드 줄을 읽어서 프로그램 심볼 테이블에 위치와 함께 저장

```
private void storeExtdefs(String line, Program program) {  
    int i = 1;  
    while(i < line.length()) {  
        String name = line.substring(i, i + 6);  
        i += 6;  
        int location = Integer.parseInt(line.substring(i, i + 6), 16);  
        i += 6;  
        location += program.getStart();  
        program.addSymbol(name.trim(), location);  
    }  
}
```

//텍스트 레코드 줄을 읽어서 명령어 코드 부분을 리소스 매니저에 로드

```
private void loadTextCode(String line) {
    int start = programStart + Integer.parseInt(line.substring(1, 7), 16);
    int length = Integer.parseInt(line.substring(7, 9), 16);
    String texts = line.substring(9);

    byte[] bytes = new byte[length];
    for(int i = 0; i < length; ++i) {
        bytes[i] = (byte)Integer.parseInt(texts.substring(2 * i, 2 * i + 2), 16);
    }

    rMgr.setMemory(start, bytes, length);
}
}
```

```
package SP25_simulator;
```

```
import java.io.EOFException;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
```

```
/**
```

```
 * ResourceManager는 컴퓨터의 가상 리소스들을 선언하고 관리하는 클래스이다. 크게 네가지
의 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.

*/

```
    HashMap<String, RandomAccessFile> deviceManager = new HashMap<>();
```

```
    byte[] memory = new byte[65536]; // String으로 수정해서 사용하여도 무방함. char  
    2byte 이므로 char => byte
```

```
    //레지스터 이름, 레지스터 값 매핑
```

```
    HashMap<String, Integer> registers = new HashMap<String, Integer>();
```

```
    //레지스터 이름, 레지스터 번호 매핑
```

```
    HashMap<Integer, String> registerNum = new HashMap<Integer, String>();
```

```
    int[] register = new int[10];
```

```
    double register_F;
```

```
    //마지막 텍스트가 저장되고난 뒤의 파일 오프셋 위치
```

```
    int offset = 0;
```

```
    SymbolTable symtabList;
```

```
    // 이외에도 필요한 변수 선언해서 사용할 것.
```

```
/**
 * 메모리, 레지스터등 가상 리소스들을 초기화한다.
 */
```

```
public void initializeResource() {
    //레지스터 매핑 변수들 초기화
    Arrays.fill(memory, (byte) 0);
    registers.put("A", 0);
    registers.put("X", 0);
    registers.put("L", 0);
    registers.put("B", 0);
    registers.put("S", 0);
    registers.put("T", 0);
    registers.put("F", 0);
    registers.put("PC", 0);
    registers.put("SW", 0);
    registers.put("TA", 0);

    registerNum.put(0, "A");
    registerNum.put(1, "X");
    registerNum.put(2, "L");
    registerNum.put(3, "B");
    registerNum.put(4, "S");
    registerNum.put(5, "T");
    registerNum.put(6, "F");
    registerNum.put(8, "PC");
    registerNum.put(9, "SW");
}
```

```
//레지스터 이름으로 레지스터 값 반환
public int getRegisterValue(String register) {
    if(register.equals("CH")) {
        return (registers.get("A") & 0xFF);
    }
    return registers.get(register);
}
```

```

//레지스터 이름으로 레지스터 값 세팅
public void setRegisterValue(String register, int num) {
    if(register.equals("CH")) {
        int newARegValue = (registers.get("A") & 0xFFFFFFFF00) | (num &
0xFF);

        registers.put("A", newARegValue);
    }
    registers.put(register, num);
}

```

```

//레지스터 이름으로 레지스터 값 더하기
public int addRegisterValue(String register, int addition) {
    if(register.equals("CH")) {
        return 0;
    }
    return registers.put(register, registers.get(register) + addition);
}

```

```

/**
 * deviceManager가 관리하고 있는 파일 입출력 stream들을 전부 종료시키는 역할. 프
로그램을 종료하거나 연결을 끊을 때
 * 호출한다.
 */

```

```

public void closeDevice() {
    for(RandomAccessFile file : deviceManager.values()) {
        try {
            file.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

```

```

/**
 * 디바이스를 사용할 수 있는 상황인지 체크. TD명령어를 사용했을 때 호출되는 함
수. 입출력 stream을 열고 deviceManager를

```



```

* 통해 관리시킨다.
*
* @param devName 확인하고자 하는 디바이스의 번호,또는 이름
*/
public void testDevice(String devName) {
    if(deviceManager.get(devName) != null)
        return;
    try {
        deviceManager.put(devName, new RandomAccessFile(devName, "rw"));
    } catch (FileNotFoundException e) {
        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 byte readDevice(String devName) {
    try {
        return deviceManager.get(devName).readByte();
    } catch (IOException e) {
        return 0;
    }
}

```

/**
* 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, byte data) {
    try {

```

```

        RandomAccessFile file = deviceManager.get(devName);
        if(file.getFilePointer() == 0) {
            file.setLength(0);
        }
        file.writeByte(data);
    } 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 byte[] getMemory(int location, int num) {
    byte[] data = new byte[num];
    for(int i = location; i < location + num; ++i) {
        if(i < 0 || i >= memory.length) {
            return data;
        }
        data[i - location] = memory[i];
    }
    return data;
}

```

//메모리 영역의 바이트 배열을 정수로 변화 후 리턴

```

public int getMemoryToInt(int location, int num) {
    int memoryValue = 0;

    byte[] bytesLoad = getMemory(location, Math.min(num, 4));
    for(int i = 0; i < bytesLoad.length; ++i) {
        int shifts = (bytesLoad.length - 1 - i) * 8;
        memoryValue |= (bytesLoad[i] & 0xFF) << shifts;
    }
}

```

```

        return memoryValue;
    }

    /**
     * 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, byte[] data, int num) {
        for(int i = locate; i < locate + num; ++i) {
            if(i < 0 || i >= memory.length) {
                return;
            }
            memory[i] = data[i - locate];
        }
        offset = locate + num;
    }

```

```

//정수에서 바이트 배열로 변화 후 메모리 영역에 저장
public void setMemoryFromInt(int locate, int memoryValue, int num) {
    byte[] bytesStore = new byte[Math.min(num, 4)];

    for(int i = 0; i < bytesStore.length; ++i) {
        int shifts = (bytesStore.length - 1 - i) * 8;
        bytesStore[i] = (byte)((memoryValue >> shifts) & 0xFF);
    }
    setMemory(locate, bytesStore, bytesStore.length);
}

```

```

//해당 메모리 영역에 addition만큼 더하기, 수정 레코드 부분에서 주로 사용
public void addMemory(int locate, byte[] addition, int num) {
    for(int i = locate; i < locate + num; ++i) {
        if(i < 0 || i >= memory.length) {
            return;
        }
    }
}

```

```

    }
    memory[i] += addition[i - locate];
}
}

```

//해당 메모리 영역에 sub만큼 빼기, 수정 레코드 부분에서 주로 사용

```

public void subMemory(int locate, byte[] sub, int num) {
    for(int i = locate; i < locate + num; ++i) {
        if(i < 0 || i >= memory.length) {
            return;
        }
        memory[i] -= sub[i - locate];
    }
}

```

/**

* 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) {
    String regName = registerNum.get(regNum);
    return registers.get(regName);
}

```

/**

* 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) {
    String regName = registerNum.get(regNum);
    registers.put(regName, value);
}

```

```

    }

    /**
     * 주로 레지스터와 메모리간의 데이터 교환에서 사용된다. int값을 char[]형태로 변경
한다.
     *
     * @param data
     * @return
     */
    public char[] intToChar(int data) {
        return null;
    }

    /**
     * 주로 레지스터와 메모리간의 데이터 교환에서 사용된다. char[]값을 int형태로 변경
한다.
     *
     * @param data
     * @return
     */
    public int byteToInt(byte[] data) {
        return 0;
    }
}

```

```
package SP25_simulator;
```

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

```

public class InstLuncher {
    ResourceManager rMgr;
    VisualSimulator vsim;

    public InstLuncher(ResourceManager resourceManager, VisualSimulator visualSimulator) {
        this.rMgr = resourceManager;
        this.vsim = visualSimulator;
    }
}

```

```
}
```

```
// instruction 별로 동작을 수행하는 메소드를 정의
```

```
// ex) public void add(){...}
```

용

//두 값을 비교하고 결과에 따라 SW 레지스터 세팅, COMP COMR TIX TIXR 에서 주로 사

```
private void compareAndSetStatusReg(int value1, int value2) {  
    if(value1 < value2) {  
        rMgr.setRegisterValue("SW", -1);    //작으면 -1 지정  
    }  
    else if(value1 > value2) {  
        rMgr.setRegisterValue("SW", 1);    //크면 1 지정  
    }  
    else {  
        rMgr.setRegisterValue("SW", 0);    //같으면 0 지정  
    }  
}
```

```
//2형식의 레지스터 관련 명령어의 연산을 수행
```

```
private boolean processRegOperator(Command command) {  
    String opString = Command.operators.get(command.getOperator()).name;  
    int firstRegNum = (command.getData() >> 4) & 0xF;  
    int secondRegNum = (command.getData() >> 0) & 0xF;  
  
    if(opString.equals("CLEAR")) { //CLEAR, 레지스터 값 초기화  
        rMgr.setRegister(firstRegNum, 0);  
    }  
    else if(opString.equals("TIXR")) { //TIXR, X 레지스터 1 증가하고 값 비교  
        rMgr.setRegisterValue("X", rMgr.getRegisterValue("X") + 1);  
        int xReg = rMgr.getRegisterValue("X");  
        int otherReg = rMgr.getRegister(firstRegNum);  
        compareAndSetStatusReg(xReg, otherReg);  
    }  
    else if(opString.equals("COMPR")) { //COMPR, 두 레지스터의 값 비교  
        int firstReg = rMgr.getRegister(firstRegNum);  
        int secondReg = rMgr.getRegister(secondRegNum);  
        compareAndSetStatusReg(firstReg, secondReg);  
    }  
}
```

```

    }
    return true;
}

```

//명령어 정보에 따라 target address 설정하기

```

private int setTargetAddresss(Command command) {
    int targetAddr = rMgr.getRegisterValue("TA");
    //e, p, b 주소 지정 방식에 따라 target address 지정
    if(command.hasFlag(Command.E_FLAG)) {
        targetAddr = command.getOperandValue(false);
    }
    else if(command.hasFlag(Command.P_FLAG)) {
        targetAddr = command.getOperandValue(true) +
rMgr.getRegisterValue("PC");
    }
    else if(command.hasFlag(Command.B_FLAG)) {
        targetAddr = command.getOperandValue(false) +
rMgr.getRegisterValue("B");
    }
    else {
        targetAddr = command.getOperandValue(true);
    }

    //x 플래그에 따라 x 레지스터 값 target address에 더하기
    if(command.hasFlag(Command.X_FLAG)) {
        targetAddr += rMgr.getRegisterValue("X");
    }
    rMgr.setRegisterValue("TA", targetAddr);
    return targetAddr;
}

```

//3, 4형식의 명령어의 연산을 수행

```

private boolean processNormalOperator(Command command) {
    int targetAddr = setTargetAddresss(command);
    String opString = Command.operators.get(command.getOperator()).name;
    //실제 연산 실행

```

```

        if(opString.length() >= 3 && opString.substring(0, 2).equals("LD")) { //LD*,
메모리->레지스터로 값 이동
            String reg = opString.substring(2);
            ld(command, reg);
        }
        else if(opString.length() >= 3 && opString.substring(0, 2).equals("ST")) { //ST*,
레지스터->메모리로 값 이동
            String reg = opString.substring(2);
            st(command, reg);
        }
        else if(opString.equals("TIX")) { //TIX, X 레지스터 1 증가하고 값 비교
            tix(command);
        }
        else if(opString.equals("COMP")) { //COMP, A 레지스터의 값과 피연산자
값 비교
            comp(command);
        }
        else if(opString.equals("JEQ")) { //SW == 0(직전 비교값이 같으면) 분기
            jeq(command);
        }
        else if(opString.equals("JGT")) { //SW == 1(직전 비교값이 크면) 분기
            jgt(command);
        }
        else if(opString.equals("JLT")) { //SW == 1(직전 비교값이 작으면) 분기
            jlt(command);
        }
        else if(opString.equals("J")) { //해당 위치로 분기
            j(command);
            if(rMgr.getRegisterValue("PC") == 0)
                return false;
        }
        else if(opString.equals("JSUB")) { //L 레지스터 리턴 주소 저장해두고 서
브 루틴으로 분기
            jsub(command);
        }
        else if(opString.equals("RSUB")) { //L 레지스터의 리턴 주소로 분기
            rsub(command);
        }
        else if(opString.equals("TD")) { //test device, 이 프로젝트에서는 파일 스트림

```

생성

트 읽기

트 쓰기

```
        td(command);
    }
    else if(opString.equals("RD")) { //read device, 이 프로젝트에서는 파일 한 바이트 읽기
        rd(command);
    }
    else if(opString.equals("WD")) { //write device, 이 프로젝트에서는 파일 한 바이트 쓰기
        wd(command);
    }
    return true;
}
```

```
//LD*
private void ld(Command command, String reg) {
    int targetAddr = rMgr.getRegisterValue("TA");
    int loadValue = 0;
    if(!command.hasFlag(Command.N_FLAG) &&
command.hasFlag(Command.I_FLAG)) {
        loadValue = targetAddr;
    }
    else {
        int loadSize = reg.equals("CH") ? 1 : 3;
        loadValue = rMgr.getMemoryToInt(targetAddr, loadSize);
    }
    rMgr.setRegisterValue(reg, loadValue);
}
```

```
//ST*
private void st(Command command, String reg) {
    int targetAddr = rMgr.getRegisterValue("TA");
    int regValue = rMgr.getRegisterValue(reg);
    int storeSize = reg.equals("CH") ? 1 : 3;
    rMgr.setMemoryFromInt(targetAddr, regValue, storeSize);
}
```

```

//TIX
private void tix(Command command) {
    int targetAddr = rMgr.getRegisterValue("TA");
    int compValue = 0;
    if(!command.hasFlag(Command.N_FLAG)                                &&
command.hasFlag(Command.I_FLAG)) {
        compValue = targetAddr;
    }
    else {
        compValue = rMgr.getMemoryToInt(targetAddr, 3);
    }
    rMgr.setRegisterValue("X", rMgr.getRegisterValue("X") + 1);
    compareAndSetStatusReg(rMgr.getRegisterValue("X"), compValue);
}

```

```

//COMP
private void comp(Command command) {
    int targetAddr = rMgr.getRegisterValue("TA");
    int compValue = 0;
    if(!command.hasFlag(Command.N_FLAG)                                &&
command.hasFlag(Command.I_FLAG)) {
        compValue = targetAddr;
    }
    else {
        compValue = rMgr.getMemoryToInt(targetAddr, 3);
    }
    compareAndSetStatusReg(rMgr.getRegisterValue("A"), compValue);
}

```

```

//JEQ
private void jeq(Command command) {
    int targetAddr = rMgr.getRegisterValue("TA");
    if(rMgr.getRegisterValue("SW") != 0)
        return;
}

```

```

        rMgr.setRegisterValue("PC", targetAddr);
    }

```

//JGT

```

private void jgt(Command command) {
    int targetAddr = rMgr.getRegisterValue("TA");
    if(rMgr.getRegisterValue("SW") != 1)
        return;
    rMgr.setRegisterValue("PC", targetAddr);
}

```

//JLT

```

private void jlt(Command command) {
    int targetAddr = rMgr.getRegisterValue("TA");
    if(rMgr.getRegisterValue("SW") != -1)
        return;
    rMgr.setRegisterValue("PC", targetAddr);
}

```

//J

```

private void j(Command command) {
    int targetAddr = rMgr.getRegisterValue("TA");
    if(command.hasFlag(Command.N_FLAG)                                &&
!command.hasFlag(Command.I_FLAG)) {
        int memValue = rMgr.getMemoryToInt(targetAddr, 3);
        rMgr.setRegisterValue("PC", memValue);
    }
    else {
        rMgr.setRegisterValue("PC", targetAddr);
    }
}

```

//JSUB

```
private void jsub(Command command) {  
    int targetAddr = rMgr.getRegisterValue("TA");  
    rMgr.setRegisterValue("L", rMgr.getRegisterValue("PC"));  
    rMgr.setRegisterValue("PC", targetAddr);  
}
```

//RSUB

```
private void rsub(Command command) {  
    rMgr.setRegisterValue("PC", rMgr.getRegisterValue("L"));  
}
```

//TD

```
private void td(Command command) {  
    int targetAddr = rMgr.getRegisterValue("TA");  
    int deviceNum = rMgr.getMemoryToInt(targetAddr, 1) & 0xFF;  
    String device = String.format("%02X", deviceNum);  
    rMgr.testDevice(device);  
    rMgr.setRegisterValue("SW", 1);  
    vsim.setUsingDevice(device);  
}
```

//RD

```
private void rd(Command command) {  
    int targetAddr = rMgr.getRegisterValue("TA");  
    int deviceNum = rMgr.getMemoryToInt(targetAddr, 1) & 0xFF;  
    String device = String.format("%02X", deviceNum);  
    byte dataRead = rMgr.readDevice(device);  
    rMgr.setRegisterValue("CH", dataRead);  
    vsim.setUsingDevice(device);  
}
```

```

//WD
private void wd(Command command) {
    int targetAddr = rMgr.getRegisterValue("TA");
    int deviceNum = rMgr.getMemoryToInt(targetAddr, 1) & 0xFF;
    String device = String.format("%02X", deviceNum);
    byte dataWrite = (byte)(rMgr.getRegisterValue("CH") & 0xFF);
    rMgr.writeDevice(device, dataWrite);
    vsim.setUsingDevice(device);
}

```

//명령어의 형식에 따라 연산 방식 지정, sicSimulator 에서 사용

```

public boolean processOperator(Command command) {
    vsim.setUsingDevice(null);
    if(command.getLength() == 4) {
        return processRegOperator(command);
    }
    else {
        return processNormalOperator(command);
    }
}
}

```

```
package SP25_simulator;
```

```
import java.util.ArrayList;
```

```
/**
```

```
 * symbol과 관련된 데이터와 연산을 소유한다. section 별로 하나씩 인스턴스를 할당한다.
```

```
 */
```

```
public class SymbolTable {
```

```
    ArrayList<String> symbolList;
```

```
    ArrayList<Integer> addressList;
```

```
    ArrayList<String> extrefList;
```

```
    // 기타 literal, external 선언 및 처리방법을 구현한다.
```

```

/**
 * 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 SymbolTable() {
    symbolList = new ArrayList<>();
    addressList = new ArrayList<>();
    extrefList = new ArrayList<>();
}

```

//EXTREF 추가

```

public void putExternalref(String extref) {
    extrefList.add(extref);
}

```

//심볼과 주소 추가

```

public void putSymbol(String symbol, int address) {
    symbolList.add(symbol);
    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) {

```

```
}
```

```
/**
```

```
 * 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;
```

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

```
        if(symbolList.get(i).equals(symbol))
```

```
            return addressList.get(i);
```

```
    }
```

```
    // ...
```

```
    return address;
```

```
}
```

```
}
```