프로젝트 2

20181259

조수민

출석번호 130

프로젝트1 (sic/xe머신 어셈블러 pass2 구현)

동기 / 목적

지난 과제에서 구현한 어셈블러를 그대로 언어만 자바로 바꿔서 구현하는 프로젝트였습니다. 결국 내용이 같기 때문에 언어 연습 혹은 개념 복습이 목적인 것 같습니다.

설계 구현 아이디어

주어진 인터페이스에 맞추어 구현하려고 노력했습니다.

```
    public SymbolTable() {
        _sectionName = Optional.empty();
        _symbolMap = new LinkedHashMap<String, Symbol>();
    }

.
.
.
        public LiteralTable() {
        literalMap = new LinkedHashMap<String, Literal>();
    }
```

심볼 테이블과 리터럴 테이블입니다.

순서대로 출력하기 위해 LinkedHashMap을 사용했습니다.

심볼 테이블은 sectionName 멤버를 추가하여 해당 심볼이 속한 controlSection의 이름을 초기화 했습니다.

```
    public ObjectCode() {
        _sectionName = Optional.empty();
        _startAddress = Optional.empty();
        _programLength = Optional.empty();
        _initialPC = Optional.empty();

        _refers = new ArrayList<String>();
        _texts = new ArrayList<Text>();
        _mods = new ArrayList<Modification>();
        _defines = new ArrayList<Define>();
    }

        class Define {
        Define(String symbolName, int address) {
            this.symbolName = symbolName;
            this.address = address;
        }

        String symbolName;
        int address;
    }

    class Text {
```

```
        Text(int address, String value) {
            this.address = address;
            this.value = value;
        }

        int address;
        String value;
    }

    class Modification {
        Modification(int address, int sizeHalfByte, String symbolNameWithSign) {
            this.address = address;
            this.sizeHalfByte = sizeHalfByte;
            this.symbolNameWithSign = symbolNameWithSign;
        }

        int address;
        int sizeHalfByte;
        String symbolNameWithSign;
    }
```

오브젝트 코드는 게시판의 코드를 참고하여 작성하였습니다.

개인적으로 저번 과제에서 제가 C로 구현한 자료 구조들과 형태가 유사하여 흥미로웠습니다.

하지만 자바 숙련도가 떨어지기 때문에 쉽사리 이해하진 못했고 시간이 오래 걸렸습니다.

```
private void parseOperatorAndOperands(String[] parts, int start) {
        if (parts.length > start) {
            String operator = parts[start];

            // Clear previous flags
            _eBit = false;
            _nBit = true; // Default to true for SIC/XE unless specifically set to false
            _iBit = true; // Default to true for SIC/XE unless specifically set to false
            _pBit = true; // PC-relative by default unless explicitly set to false

            if (operator.contains("+")) {
                _eBit = true;  // Extended instruction
                _pBit = false;  // Extended instruction is not PC-relative
            }
```

```
            _operator = Optional.of(operator.replaceAll("[+@#]", ""));  // Clean operator
        }

        if (parts.length > start + 1) {
            _operands = new ArrayList<>(Arrays.asList(parts[start + 1].split(",\\s*")));
            if (_operands.size() > 0 && _operands.get(_operands.size() - 1).endsWith(",X")
                _xBit = true;
                _operands.set(_operands.size() - 1, _operands.get(_operands.size() - 1).r
            }

            //operands가 #으로 시작할 때
            if (_operands.size() > 0 && _operands.get(0).startsWith("#")) {
                _iBit = true;
                _nBit = false;
                _pBit = false;
                _operands.set(0, _operands.get(0).replace("#", ""));
```

```
            }

            //operands가 @으로 시작할 때
            if (_operands.size() > 0 && _operands.get(0).startsWith("@")) {
                _iBit = false;
                _nBit = true;
                _pBit = true;
                _operands.set(0, _operands.get(0).replace("@", ""));
            }
        }

        if (parts.length > start + 2) {
            _comment = Optional.of(parts[start + 2]);
        }
    }
```

token 클래스에서 operator 혹은 operand를 첫번쨰 character를 확인한 뒤 nixbpe 비트를 알맞게 초기화 해줍니다.

실행 결과

output_symtab.txt

```
1    COPY      0x0000
2    FIRST     0x0000     COPY
3    CLOOP     0x0003     COPY
4    ENDFIL    0x0017     COPY
5    RETADR    0x002A     COPY
6    LENGTH    0x002D     COPY
7    BUFFER    0x0033     COPY
8    BUFEND    0x1033     COPY
9    MAXLEN    0x1000     COPY
10   RDREC     REF
11   WRREC     REF
12
13
14   RDREC     0x0000
15   RLOOP     0x0009     RDREC
16   EXIT      0x0020     RDREC
17   INPUT     0x0027     RDREC
18   MAXLEN    0x0028     RDREC
19   BUFFER    REF
20   LENGTH    REF
21   BUFEND    REF
22
23
24   WRREC     0x0000
25   WLOOP     0x0006     WRREC
26   LENGTH    REF
27   BUFFER    REF
28
```

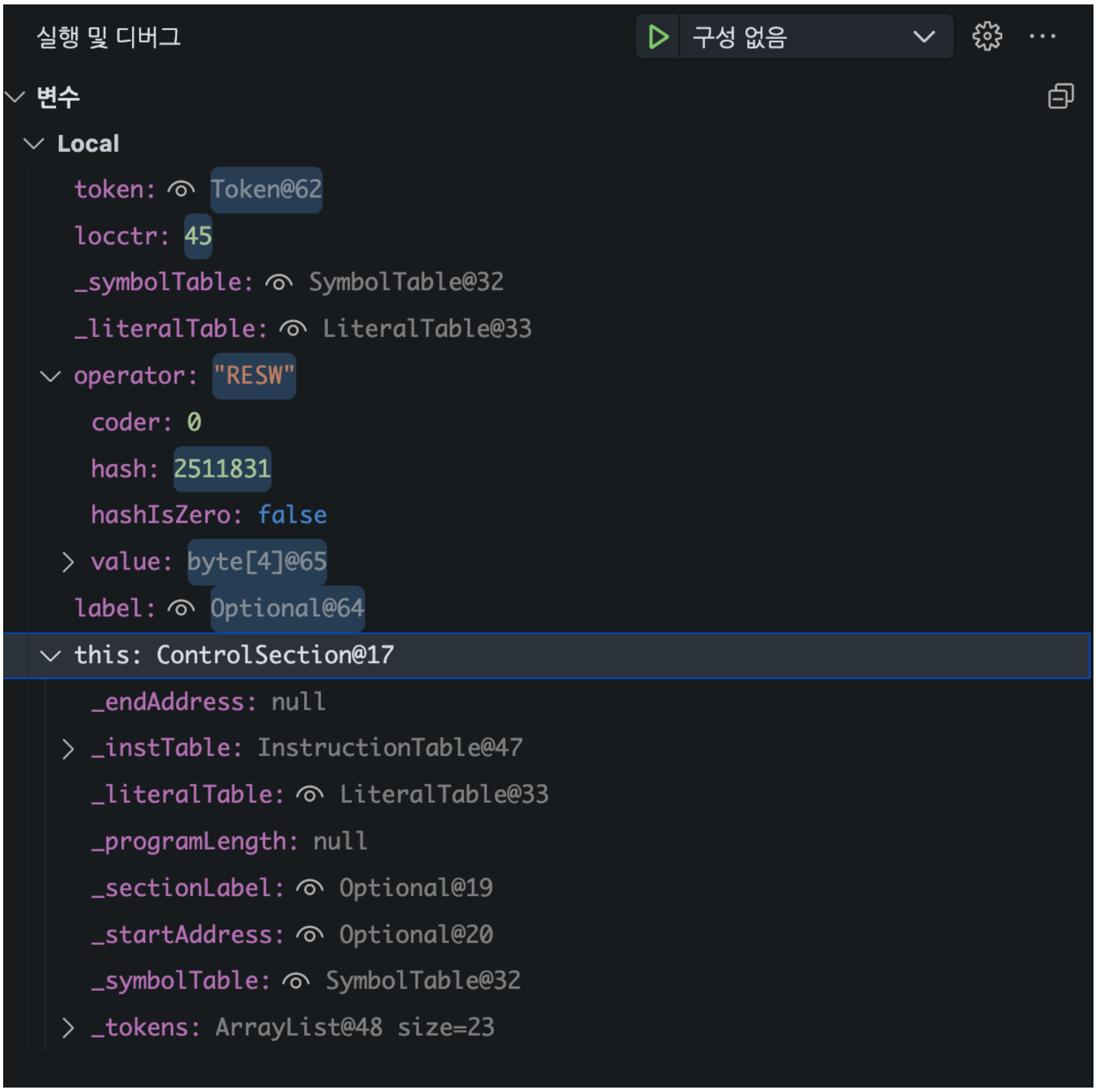
output_littab.txt

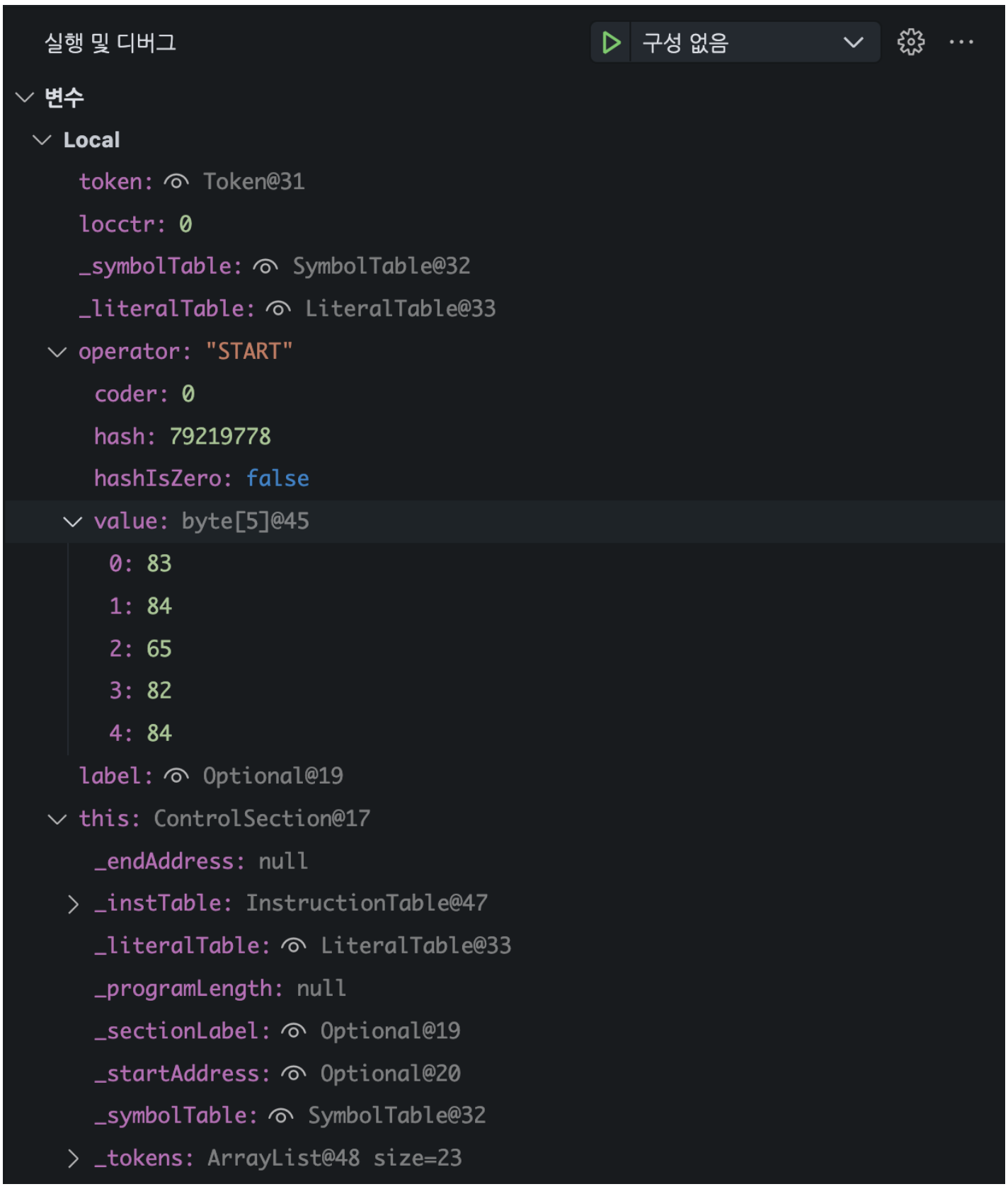
```
1    =C'EOF' 0x0030
2
3
4
5    =X'05' 0x001B
```

```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDREC WRREC
T0000001720274B1000000320232900003320074B1000003F2FEC0320160F20161D
T00001D0100000F200A4B1000003E2000454F4610
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E


HRDREC 00000000002B
RBUFFERLENGTHBUFEND
T000000B410B400B44077201FE3201B332FFADB2015A004332009571000000B8501D
T00001D3B2FE9131000004F0000F10B
M00001805+BUFFER
M00002105+LENGTH
E


HWRREC 00000000001C
RLENGTHBUFFER
T000000B41077100000E32012332FFA53100000DF2008B8503B2FEE4F00001B
M00000305+LENGTH
M00000D05+BUFFER
E
```

개인적으로 아쉬운 과제입니다. 시험기간과 겹치면서 시간이 너무 부족했습니다. 구현하지 못한 내용이 좀 있습니다. 결론 및 보충할 점에서 이어서 작성

디버깅


실행 및 디버그
구성 없음
변수
Local
token: Token@62
locctr: 45
_symbolTable: SymbolTable@32
_literalTable: LiteralTable@33
operator: "RESW"
coder: 0
hash: 2511831
hashIsZero: false
value: byte[4]@65
label: Optional@64
this: ControlSection@17
_endAddress: null
_instTable: InstructionTable@47
_literalTable: LiteralTable@33
_programLength: null
_sectionLabel: Optional@19
_startAddress: Optional@20
_symbolTable: SymbolTable@32
_tokens: ArrayList@48 size=23



결론 및 보충할 점

몇가지 구현하지 못한 내용이 있습니다.

1. 오브젝트 코드 한 줄의 길이를 출력하지 못했습니다.
2. 오브젝트 코드의 형식이 매끄럽지 못합니다. 이는 toString을 수정해서 고칠 수 있습니다.
3. EQU의 operand 연산을 완벽히 구현하지 못했습니다. 때문에 BUFFEND - BUFFER를 모디피케이션으로 쓰지 못했습니다.
4. 에러 처리에 소극적이었습니다.

다 크리티컬한 버그가 아니고, 시간만 있다면 충분히 고칠 수 있는 문제들입니다. 욕심 많은 과목이었는데 아쉽습니다.

개인적으로 자바 숙련도가 떨어지기 때문에 주어진 인터페이스 내용을 이해하는데도 시간이 많이 걸렸습니다. 그래도 최선을 다했다고 생각합니다.

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

import org.w3c.dom.Text;
```

```
public class ObjectCode {
    public ObjectCode() {
        _sectionName = Optional.empty();
        _startAddress = Optional.empty();
        _programLength = Optional.empty();
        _initialPC = Optional.empty();

        _refers = new ArrayList<String>();
        _texts = new ArrayList<Text>();
        _mods = new ArrayList<Modification>();
        _defines = new ArrayList<Define>();
    }

    public void setSectionName(String sectionName) {
        _sectionName = Optional.of(sectionName);
    }

    public void setStartAddress(int address) {
        _startAddress = Optional.of(address);
    }

    public void setProgramLength(int length) {
        _programLength = Optional.of(length);
    }

    public void addDefineSymbol(String symbolName, int address) {
        _defines.add(new Define(symbolName, address));
    }

    public void addReferSymbol(String symbolName) {
        _refers.add(symbolName);
    }

    public void addText(int address, String value) {
        _texts.add(new Text(address, value));
    }

    public void addModification(String symbolNameWithSign, int address, int sizeHalfByte)
        _mods.add(new Modification(address, sizeHalfByte, symbolNameWithSign));
    }

    public void setInitialPC(int address) {
        _initialPC = Optional.of(address);
    }

    /**
     * ObjectCode 객체를 String으로 변환한다. Assembler.java에서 오브젝트 코드를 출력하는 데에 사용된
     */
    @Override
    public String toString() {
        if (_sectionName.isEmpty() || _startAddress.isEmpty() || _programLength.isEmpty()
            throw new RuntimeException("illegal operation");

        String sectionName = _sectionName.get();
        int startAddress = _startAddress.get();
        int programLength = _programLength.get();

        String header = String.format("H%-6s%06X%06X\n", sectionName, startAddress, progr

        // Define Record
```

```
        StringBuilder define = new StringBuilder();
        if (!_defines.isEmpty()) {
            define.append("D");
            for (Define def : _defines) {
                define.append(String.format("%-6s%06X", def.symbolName, def.address));
            }
            define.append("\n");
        }

        // Reference Record
        StringBuilder refer = new StringBuilder();
        if (!_refers.isEmpty()) {
            refer.append("R");
            for (String ref : _refers) {
                refer.append(String.format("%-6s", ref));
            }
            refer.append("\n");
        }

        // Text Records
        StringBuilder text = new StringBuilder();
        int currentRecordLength = 0;
        int currentRecordStartAddress = 0;
        for (Text tx : _texts) {
            if (currentRecordLength == 0) {
                currentRecordStartAddress = tx.address;
                text.append(String.format("T%06X", currentRecordStartAddress));
            }
            if (currentRecordLength + (tx.value.length() / 2) > 0x1D) {
                text.append(String.format("%02X", currentRecordLength));
                text.append(tx.value.substring(0, 0x1D * 2 - currentRecordLength));
                text.append("\n");
                tx.value = tx.value.substring(0x1D * 2 - currentRecordLength);
                currentRecordLength = 0;
                text.append(String.format("T%06X", tx.address + (0x1D * 2 - currentRecord
            }
            text.append(tx.value);
            currentRecordLength += tx.value.length() / 2;
            if (currentRecordLength >= 0x1D) {
                text.append("\n");
                currentRecordLength = 0;
            }
        }
        if (currentRecordLength > 0) {
            text.append(String.format("%02X", currentRecordLength)).append("\n");
        }

        // Modification Records
        StringBuilder modification = new StringBuilder();
        for (Modification mod : _mods) {
            modification.append(String.format("M%06X%02X+%s\n", mod.address, mod.sizeHalf
        }

        // End Record
        String end = "E" + _initialPC.map(x -> String.format("%06X", x)).orElse("") + "\n

        return header + define.toString() + refer.toString() + text.toString() + modifica
    }
```

```
    class Define {
        Define(String symbolName, int address) {
            this.symbolName = symbolName;
            this.address = address;
        }

        String symbolName;
        int address;
    }

    class Text {
        Text(int address, String value) {
            this.address = address;
            this.value = value;
        }

        int address;
        String value;
    }
```

```
    class Modification {
        Modification(int address, int sizeHalfByte, String symbolNameWithSign) {
            this.address = address;
            this.sizeHalfByte = sizeHalfByte;
            this.symbolNameWithSign = symbolNameWithSign;
        }

        int address;
        int sizeHalfByte;
        String symbolNameWithSign;
    }

    // TODO: private field 선언.
    private Optional<String> _sectionName;
    private Optional<Integer> _startAddress;
    private Optional<Integer> _programLength;
    private Optional<Integer> _initialPC;
    private ArrayList<Define> _defines;

    private ArrayList<String> _refers;
    private ArrayList<Text> _texts;
    private ArrayList<Modification> _mods;
}
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.stream.Collectors;

public class SymbolTable {
    /**
     * 심볼 테이블 객체를 초기화한다.
     */
```

```
    public SymbolTable() {
        _sectionName = Optional.empty();
        _symbolMap = new LinkedHashMap<String, Symbol>();
    }

    /**
     * EQU를 제외한 명령어/지시어에 label이 포함되어 있는 경우, 해당 label을 심볼 테이블에 추가한다.
     *
     * @param label   라벨
     * @param address 심볼의 주소
     * @throws RuntimeException (TODO: exception 발생 조건을 작성하기)
     */
    public void putLabel(String label, int address) throws RuntimeException {
        // TODO: EQU를 제외한 명령어/지시어의 label로 생성되는 심볼을 추가하기.

        Symbol symbol = new Symbol(label, address);
        _symbolMap.put(label, symbol);
    }

    /**
     * EQU에 label이 포함되어 있는 경우, 해당 label을 심볼 테이블에 추가한다.
     *
     * @param label    라벨
     * @param locctr   locctr 값
     * @param equation equation 문자열
     * @throws RuntimeException equation 파싱 오류
     */
    public void putLabel(String label, int locctr, String equation) throws RuntimeExcepti
        if (equation.equals("*")) {
            putLabel(label, locctr);
            return;
        }

        try {
            int result = evaluateExpression(equation);
            _symbolMap.put(label, new Symbol(label, result));
        } catch (Exception e) {
            throw new RuntimeException("Failed to evaluate expression: " + equation, e);
        }
    }


    private int evaluateExpression(String equation) throws RuntimeException {
        // 정규표현식을 이용하여 피연산자와 연산자를 분리합니다.
        Matcher m = Pattern.compile("([a-zA-Z]+)|([0-9]+)|([+\\-*/])").matcher(equation);
        int result = 0;
        String lastOperator = "+"; // 초기 연산자는 '+'로 설정하여 첫 번째 값을 그대로 사용합니다.

        while (m.find()) {
            String token = m.group();
            if (token.matches("[+\\-*/]")) { // 연산자인 경우
                lastOperator = token;
            } else {
                int value;
                if (Character.isDigit(token.charAt(0))) {
                    value = Integer.parseInt(token); // 숫자인 경우
                } else {
                    Optional<Integer> optValue = getAddress(token);
                    if (optValue.isEmpty()) {
                        throw new RuntimeException("Failed to find symbol: " + token);
```

```
                }
                value = optValue.get(); // 라벨의 주소값을 가져옵니다.
            }

            switch (lastOperator) {
                case "+": result += value; break;
                case "-": result -= value; break;
                case "*": result *= value; break;
                case "/": result /= value; break;
            }
        }
    }

    return result;
}

/**
 * EXTREF에 operand가 포함되어 있는 경우, 해당 operand를 심볼 테이블에 추가한다.
 *
 * @param refer operand에 적힌 하나의 심볼
 * @throws RuntimeException (TODO: exception 발생 조건을 작성하기)
 */
public void putRefer(String refer) throws RuntimeException {
    // TODO: EXTREF의 operand로 생성되는 심볼을 추가하기.
    // EXTREF의 operand는 주소를 가지지 않으므로, 주소값을 출력할 때 REF라고 저장해야함
    //순서는 맨 마지막에 출력돼야함

    Symbol symbol = new Symbol(refer, -1);
    _symbolMap.put(refer, symbol);

}

/**
 * 심볼 테이블에서 심볼을 찾는다.
 *
 * @param name 찾을 심볼 명칭
 * @return 심볼. 없을 경우 empty
 */
public Optional<Symbol> searchSymbol(String name) {
    // TODO: symbolMap에서 name에 해당하는 심볼을 찾아 반환하기.
    return Optional.empty();
}

/**
 * 심볼 테이블에서 심볼을 찾아, 해당 심볼의 주소를 반환한다.
 *
 * @param symbolName 찾을 심볼 명칭
 * @return 심볼의 주소. 없을 경우 empty
 */
public Optional<Integer> getAddress(String symbolName) {

    if (_symbolMap.containsKey(symbolName)) {
        return Optional.of(_symbolMap.get(symbolName).getAddress());
    } else {
        return Optional.empty();
    }
}

public void setSectionName(Optional<String> sectionName) {
    _sectionName = sectionName;
```

```
    }

    public String getSectionName() {
        return _sectionName.orElse("");
    }

    /**
     * 심볼 테이블을 String으로 변환한다. Assembler.java에서 심볼 테이블을 출력하기 위해 사용한다.
     */
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        List<String> refList = new ArrayList<>();
        String lastSectionLabel = null; // 마지막으로 사용된 섹션 라벨을 추적합니다.
        boolean isFirst = true; // 첫 번째 요소를 추적하는 플래그

        for (Map.Entry<String, Symbol> entry : _symbolMap.entrySet()) {
            Symbol symbol = entry.getValue();
            if (symbol.getAddress() == -1) {
                // REF로 표시된 항목들은 리스트에 추가하여 나중에 처리합니다.
                refList.add(String.format("%-8s REF\n", entry.getKey()));
            } else {
                // 첫 번째 심볼 출력 시 섹션 라벨을 출력하지 않습니다.
                if (isFirst) {
                    sb.append(String.format("%-8s 0x%04X\n", entry.getKey(), symbol.getAd
                    isFirst = false; // 첫 번째 요소를 처리한 후 플래그를 false로 설정합니다.
                } else {
                    // 현재 심볼의 섹션 라벨이 이전과 다르면 업데이트합니다.
                    if (!this.getSectionName().equals(lastSectionLabel)) {
                        lastSectionLabel = this.getSectionName(); // 섹션 라벨을 업데이트합니
                    }
                    // 섹션 라벨이 있는 경우에만 섹션 라벨을 추가합니다.
                    if (lastSectionLabel.isEmpty()) {
                        sb.append(String.format("%-8s 0x%04X\n", entry.getKey(), symbol.g
                    } else {
                        sb.append(String.format("%-8s 0x%04X    %s\n", entry.getKey(), sy
                    }
                }
            }
        }
        // REF로 표시된 항목들을 마지막에 추가합니다.
        for (String ref : refList) {
            sb.append(ref);
        }

        return sb.toString();
    }



    /** 심볼 테이블. key: 심볼 명칭, value: 심볼 객체 */
    private HashMap<String, Symbol> _symbolMap;
    private Optional<String> _sectionName;
}

class Symbol {
    /**
     * 심볼 객체를 초기화한다.
     *
```

```
     * @param name    심볼 명칭
     * @param address 심볼의 절대 주소
     */
    public Symbol(String name, int address /* , 추가로 선언한 field들... */) {
        // TODO: 심볼 객체 초기화.

        _name = name;
        _address = address;
    }

    /**
     * 심볼 명칭을 반환한다.
     *
     * @return 심볼 명칭
     */
    public String getName() {
        return _name;
    }

    /**
     * 심볼의 주소를 반환한다.
     *
     * @return 심볼 주소
     */
    public int getAddress() {
        return _address;
    }

    // TODO: 추가로 선언한 field에 대한 getter 작성하기.

    /**
     * 심볼을 String으로 변환한다.
     */
    @Override
    public String toString() {
        // TODO: 심볼을 String으로 표현하기.
        //address는 16진수
        return String.format("%s\t%s", _name, Integer.toHexString(_address));
    }

    /** 심볼의 명칭 */
    private String _name;

    /** 심볼의 주소 */
    private int _address;

    // TODO: 추가로 필요한 field 선언
}
```

```
import java.util.ArrayList;
import java.util.Optional;
import java.util.stream.Collectors;
import java.util.Arrays;

public class Token {
    /**
```

```
     * 소스 코드 한 줄에 해당하는 토큰을 초기화한다.
     *
     * @param input 소스 코드 한 줄에 해당하는 문자열
     * @throws RuntimeException 잘못된 형식의 소스 코드 파싱 시도.
     */

    public Token(String input) throws RuntimeException {

        if (input == null || input.isEmpty()) {
            throw new IllegalArgumentException("Input string must not be null or empty.")
        }



        String[] parts = input.split("\t", -1);

        // 첫번째 요소가 . 이면 전부 주석
        if (parts[0].startsWith(".")) {
            _comment = Optional.of(parts[0]);
            return;
        }

        if (parts[0].isEmpty()) {
            parseOperatorAndOperands(parts, 1);
        } else {
            _label = Optional.of(parts[0]);
            parseOperatorAndOperands(parts, 1);
        }

        // System.out.println(this.toString());
    }

    private void parseOperatorAndOperands(String[] parts, int start) {
        if (parts.length > start) {
            String operator = parts[start];

            // Clear previous flags
            _eBit = false;
            _nBit = true; // Default to true for SIC/XE unless specifically set to false
            _iBit = true; // Default to true for SIC/XE unless specifically set to false
            _pBit = true; // PC-relative by default unless explicitly set to false

            if (operator.contains("+")) {
                _eBit = true;  // Extended instruction
                _pBit = false;  // Extended instruction is not PC-relative
            }



            _operator = Optional.of(operator.replaceAll("[+@#]", ""));  // Clean operator
        }

        if (parts.length > start + 1) {
            _operands = new ArrayList<>(Arrays.asList(parts[start + 1].split(",\\s*")));
            if (_operands.size() > 0 && _operands.get(_operands.size() - 1).endsWith(",X")
                _xBit = true;
                _operands.set(_operands.size() - 1, _operands.get(_operands.size() - 1).r
            }

            //operands가 #으로 시작할 때
            if (_operands.size() > 0 && _operands.get(0).startsWith("#")) {
```

```
                _iBit = true;
                _nBit = false;
                _pBit = false;
                _operands.set(0, _operands.get(0).replace("#", ""));
            }

            //operands가 @으로 시작할 때
            if (_operands.size() > 0 && _operands.get(0).startsWith("@")) {
                _iBit = false;
                _nBit = true;
                _pBit = true;
                _operands.set(0, _operands.get(0).replace("@", ""));
            }
        }

        if (parts.length > start + 2) {
            _comment = Optional.of(parts[start + 2]);
        }
    }

    public void setSize(int size) {
        _size = size;
    }

    public int getSize() {
        return _size;
    }

    /**
     * label 문자열을 반환한다.
     *
     * @return label 문자열. 없으면 empty <code>Optional</code>.
     */
    public Optional<String> getLabel() {
        return _label;
    }

    /**
     * operator 문자열을 반환한다.
     *
     * @return operator 문자열. 없으면 empty <code>Optional</code>.
     */
    public Optional<String> getOperator() {
        return _operator;
    }

    /**
     * operand 문자열 배열을 반환한다.
     *
     * @return operand 문자열 배열
     */
    public ArrayList<String> getOperands() {
        return _operands;
    }

    /**
     * comment 문자열을 반환한다.
     *
     * @return comment 문자열. 없으면 empty <code>Optional</code>.
     */
```

```
    public Optional<String> getComment() {
        return _comment;
    }

    /**
     * 토큰의 iNdirect bit가 1인지 여부를 반환한다.
     *
     * @return N bit가 1인지 여부
     */
    public boolean isN() {
        return _nBit;
    }

    /**
     * 토큰의 Immediate bit가 1인지 여부를 반환한다.
     *
     * @return I bit가 1인지 여부
     */
    public boolean isI() {
        return _iBit;
    }

    /**
     * 토큰의 indeX bit가 1인지 여부를 반환한다.
     *
     * @return X bit가 1인지 여부
     */
    public boolean isX() {
        return _xBit;
    }

    /**
     * 토큰의 Pc relative bit가 1인지 여부를 반환한다.
     *
     * @return P bit가 1인지 여부
     */
    public boolean isP() {
        return _pBit;
    }

    /**
     * 토큰의 Extra bit가 1인지 여부를 반환한다.
     *
     * @return E bit가 1인지 여부
     */
    public boolean isE() {
        return _eBit;
    }

    public void setAddress(int address) {
        _address = Optional.of(address);
    }

    public Optional<Integer> getAddress() {
        return _address;
    }

    /**
     * StringToken 객체의 정보를 문자열로 반환한다. 디버그 용도로 사용한다.
     */
```

```
    @Override
    public String toString() {
        String label = _label.map(x -> "<" + x + ">").orElse("(no label)");
        String operator = (isE() ? "+" : "") + _operator.map(x -> "<" + x + ">").orElse("
        String operand = (isN() && !isI() ? "@" : "") + (isI() && !isN() ? "#" : "")
                + (_operands.isEmpty() ? "(no operand)"
                        : "<" + _operands.stream().collect(Collectors.joining("/")) + ">"
                + (isX() ? (_operands.isEmpty() ? "X" : "/X") : "");
        String comment = _comment.map(x -> "<" + x + ">").orElse("(no comment)");

        String formatted = String.format("%-12s\t%-12s\t%-18s\t%s", label, operator, oper
        return formatted;
    }

    private Optional<String> _label = Optional.empty();
    private Optional<String> _operator = Optional.empty();
    private ArrayList<String> _operands = new ArrayList<>();
    private Optional<String> _comment = Optional.empty();
    private Optional<Integer> _address = Optional.empty();

    private boolean _nBit = true;
    private boolean _iBit = true;
    private boolean _xBit = false;
    // private boolean _bBit; /** base relative는 구현하지 않음 */
    private boolean _pBit = true;
    private boolean _eBit = false;

    private int _size = 0;
}
```

```
import java.util.ArrayList;
import java.util.Optional;
import java.util.stream.Collectors;


public class ControlSection {
    /**
     * pass1 작업을 수행한다. 기계어 목록 테이블을 통해 소스 코드를 토큰화하고, 심볼 테이블 및 리터럴 테이
     *
     * @param instTable 기계어 목록 테이블
     * @param input     하나의 control section에 속하는 소스 코드. 마지막 줄은 END directive를 강
     *                  추가하였음.
     * @throws RuntimeException 소스 코드 컴파일 오류
     */
    public ControlSection(InstructionTable instTable, ArrayList<String> input) throws Run
        _instTable = instTable; // 기계어 목록 테이블을 필드에 할당
        _symbolTable = new SymbolTable(); // 심볼 테이블 초기화
        _literalTable = new LiteralTable(); // 리터럴 테이블 초기화

        _tokens = input.stream()
                .map(Token::new) // String을 Token으로 변환
                .collect(Collectors.toCollection(ArrayList::new)); // ArrayList<Token>으로

        int locctr = 0;

        for (Token token : _tokens) {
            if (token.getOperator().isEmpty()) {
```

```
                boolean isLabelEmpty = token.getLabel().isEmpty();
                boolean isOperandEmpty = token.getOperands().isEmpty();
                if (!isLabelEmpty || !isOperandEmpty)
                    throw new RuntimeException("missing operator\n\n" + token.toString()
                continue;
            }
            String operator = token.getOperator().get();

            Optional<InstructionInfo> optInst = instTable.search(operator);
            boolean isOperatorInstruction = optInst.isPresent();
            if (isOperatorInstruction) {
                locctr = handlePass1InstructionStep(token, locctr, _symbolTable, _litera
                // System.out.println(token.toString()); /** 디버깅 용도 */
            } else {
                locctr = handlePass1DirectiveStep(token, locctr, _symbolTable, _literalTa
                // System.out.println(token.toString()); /** 디버깅 용도 */
            }
        }
        _endAddress = Optional.of(locctr);
        _programLength = Optional.of(locctr - _startAddress.get());


    }

    int handlePass1InstructionStep(Token token, int locctr, SymbolTable _symbolTable, Lit
        Optional<String> label = token.getLabel();

        if(label.isPresent()) {
            _symbolTable.putLabel(label.get(), locctr);
        }

        ArrayList<String> operands = token.getOperands();

        if (operands.get(0).startsWith("=")) {
            _literalTable.putLiteral(operands.get(0));
        }

        int size = instInfo.getSize(token);
        token.setSize(size);
        token.setAddress(locctr);
        return (locctr + size);
    }

    int handlePass1DirectiveStep(Token token, int locctr, SymbolTable _symbolTable, Liter

        // System.out.println("operator: " + operator);
        Optional<String> label = token.getLabel();
        switch (operator) {
            case "START":
                _startAddress = Optional.of(Integer.parseInt(token.getOperands().get(0),
                locctr = _startAddress.get();
                _sectionLabel = token.getLabel();
                _symbolTable.setSectionName(_sectionLabel);
                _symbolTable.putLabel(_sectionLabel.get(), locctr);
                break;
            case "CSECT":
                _sectionLabel = token.getLabel();
                _symbolTable.setSectionName(_sectionLabel);
                locctr = 0;
                _startAddress = Optional.of(locctr);
```

```
                _symbolTable.putLabel(_sectionLabel.get(), locctr);
                break;
            case "END":
                //남은 리터럴 주소 할당 해줘야함
                locctr = _literalTable.setLiteralAddressAll(locctr);

                break;
            case "BYTE":
                if(label.isPresent()) {
                    _symbolTable.putLabel(label.get(), locctr);
                }
                token.setSize(1);
                token.setAddress(locctr);
                locctr += 1;
                break;
            case "WORD":
                if(label.isPresent()) {
                    _symbolTable.putLabel(label.get(), locctr);
                }
                token.setSize(3);
                token.setAddress(locctr);
                locctr += 3;
                break;
            case "RESB":
                if(token.getOperands().get(0).startsWith("=")) {
                    _literalTable.putLiteral(token.getOperands().get(0));
                }
                if(token.getLabel().isPresent()) {
                    _symbolTable.putLabel(token.getLabel().get(), locctr);
                }
                locctr += Integer.parseInt(token.getOperands().get(0));
                break;
            case "RESW":
                if(token.getOperands().get(0).startsWith("=")) {
                    _literalTable.putLiteral(token.getOperands().get(0));
                }
                if(token.getLabel().isPresent()) {
                    _symbolTable.putLabel(token.getLabel().get(), locctr);
                }
                locctr += 3 * Integer.parseInt(token.getOperands().get(0));
                break;
            case "EQU":
                _symbolTable.putLabel(token.getLabel().get(), locctr, token.getOperands()
                break;
            case "EXTDEF":
                break;
            case "EXTREF":
                for (String refer : token.getOperands()) {
                    _symbolTable.putRefer(refer);
                }
                break;
            case "LTORG":
                locctr = _literalTable.setLiteralAddressAll(locctr);
                break;
            default:
                break;
        }


        return locctr;
```

```
    }

    /**
     * pass2 작업을 수행한다. pass1에서 초기화한 토큰 테이블, 심볼 테이블 및 리터럴 테이블을 통해 오브젝
     *
     * @return 해당 control section에 해당하는 오브젝트 코드 객체
     * @throws RuntimeException 소스 코드 컴파일 오류
     */
    public ObjectCode buildObjectCode() throws RuntimeException {
        ObjectCode objCode = new ObjectCode();
        objCode.setProgramLength(_programLength.get()); // Set the length of the program
        int locctr = _startAddress.get(); // Initialize LOCCTR to the start address of th

        for (Token token : _tokens) {
            if (token.getOperator().isEmpty()) {
                continue;
            }

            int pc = locctr + token.getSize();  // Set PC to the address after the curren
            String operator = token.getOperator().get();
            Optional<InstructionInfo> optInst = _instTable.search(operator);

            if (optInst.isPresent()) {
                handlePass2InstructionStep(pc, token, locctr, _symbolTable, _literalTable
            } else {
                handlePass2DirectiveStep(pc, token, locctr, operator, objCode);
                // If the directive changes the LOCCTR, handle it in `handlePass2Directiv
            }
            locctr = pc;
        }

        return objCode;
    }

    private void handlePass2InstructionStep(int pc, Token token, int locctr, SymbolTable

        int opcode = instInfo.getOpcode();
        InstructionInfo.Format format = instInfo.getFormat();
        boolean isExtended = token.isE();
        boolean isPCRelative = token.isP();


        if(token.getOperator().get().equals("RSUB")) {
            objCode.addText(locctr, "4F0000");
            return;
        }

        String operand = token.getOperands().isEmpty() ? "" : token.getOperands().get(0)
        int targetAddress = 0;
        int n = token.isN() ? 1 : 0;
        int i = token.isI() ? 1 : 0;
        int x = token.isX() ? 1 : 0;
        int p = token.isP() ? 1 : 0;
        int b = 0; // Base relative is not used here
        int e = isExtended ? 1 : 0;

        switch (format) {
            case TWO:
                handleFormat2Instruction(token, opcode, objCode, locctr, instInfo);
                break;
```

```
            case THREE_OR_FOUR:
                targetAddress = calculateTargetAddress(pc, operand, locctr, _symbolTable,
                String objectCode = encodeInstruction(opcode, targetAddress, n, i, x, b,
                objCode.addText(locctr, objectCode);
                if (isExtended) {
                    objCode.addModification(operand, locctr + 1, 5);  // Assume modificat
                }
                break;
        }

        // pc counter op와 opcode targetAddress n i x b p e
        // System.out.println("pc = " + pc + " operator" + token.getOperator() + " opcode
    }

    private int calculateTargetAddress(int pc, String operand, int locctr, SymbolTable _s
        int targetAddress = 0;
        if (isExtended) {
            return targetAddress; // 포맷 4는 주소 계산을 따로 하지 않습니다.
        }

        // 피연산자로부터 타겟 주소 또는 변위를 계산
        if (operand.startsWith("=")) {
            targetAddress = _literalTable.searchLiteral(operand).get().getAddress().get()
        } else if (operand.startsWith("#")) {
            targetAddress = Integer.parseInt(operand.substring(1));
        } else if (operand.startsWith("@")) {
            String symbol = operand.substring(1);
            targetAddress = _symbolTable.getAddress(symbol).get();
        } else {
            Optional<Integer> SymbolIsPresent = _symbolTable.getAddress(operand);
            if (SymbolIsPresent.isPresent()) {
                targetAddress = SymbolIsPresent.get();
            } else {
                targetAddress = 0;
            }

        }

        if (isPCRelative) {
            int disp = targetAddress - pc;
            if (disp >= -2048 && disp <= 2047) {
                return disp & 0xFFF;
            } else {
                throw new RuntimeException("PC-relative addressing out of range for symbo
            }
        } else {
            return targetAddress;
        }
    }

    private String encodeInstruction(int opcode, int targetAddress, int n, int i, int x,

        if (e == 1) {
            String ret = "";
            String objectCode = String.format("%02X", opcode);
            ret += objectCode.charAt(0);

            // objectCOde 뒤에 남은 숫자와 n, i 를 조합
            int temp = Integer.parseInt(objectCode.substring(1), 16);
            temp += n * 2;
```

```
            temp += i;
            objectCode = String.format("%01X", temp);
            ret += objectCode;

            // x, b, p, e 를 조합
            temp = x * 8;
            temp += b * 4;
            temp += p * 2;
            temp += e;
            objectCode = String.format("%01X", temp);

            ret += objectCode;

            // targetAddress를 16진수로 변환
            objectCode = String.format("%05X", targetAddress);
            ret += objectCode;

            return ret;
        }

        //opcode 16진수로 변환
        String ret = "";
        String objectCode = String.format("%02X", opcode);
        ret += objectCode.charAt(0);

        // objectCOde 뒤에 남은 숫자와 n, i 를 조합
        int temp = Integer.parseInt(objectCode.substring(1), 16);
        temp += n * 2;
        temp += i;
        objectCode = String.format("%01X", temp);
        ret += objectCode;

        // x, b, p, e 를 조합
        temp = x * 8;
        temp += b * 4;
        temp += p * 2;
        temp += e;
        objectCode = String.format("%01X", temp);
        ret += objectCode;

        // targetAddress를 16진수로 변환
        objectCode = String.format("%03X", targetAddress);
        ret += objectCode;

        return ret;
    }



    private void handleFormat2Instruction(Token token, int opcode, ObjectCode objCode, in
        int r1 = InstructionInfo.stringToRegister(token.getOperands().get(0));
        int r2 = token.getOperands().size() > 1 ? InstructionInfo.stringToRegister(token
        String objectCode = String.format("%02X%1X%1X", opcode, r1, r2);
        objCode.addText(locctr, objectCode);
    }
```

```
    private void handlePass2DirectiveStep(int pc, Token token, int locctr, String operato
        switch (operator) {
            case "START":
                objCode.setStartAddress(locctr);
                objCode.setSectionName(_sectionLabel.get());

                break;
            case "CSECT":
                objCode.setSectionName(_sectionLabel.get());
                objCode.setStartAddress(0);
                objCode.setProgramLength(_endAddress.get());
                break;
            case "END":
                break;
            case "BYTE":
                String operand = token.getOperands().get(0);
                if (operand.startsWith("X")) {
                    String value = operand.substring(2, operand.length() - 1); // X'...'C
                    objCode.addText(locctr, value);
                    locctr += value.length() / 2; // 16진수이므로 2로 나누어야 함
                } else if (operand.startsWith("C")) {
                    String value = operand.substring(2, operand.length() - 1); // C'...'C
                    StringBuilder hexValue = new StringBuilder();
                    for (int i = 0; i < value.length(); i++) {
                        hexValue.append(String.format("%02X", (int) value.charAt(i))); //
                    }
                    objCode.addText(locctr, hexValue.toString());
                    locctr += value.length(); // 문자 수만큼 LOCCTR 증가
                }
                break;
            case "WORD":
                locctr += 3; // 3바이트
                break;
            case "RESB":
                break;
            case "RESW":
                break;
            case "EQU":
                ArrayList<String> operands = token.getOperands();
                String symbol = token.getLabel().get();

                break;
            case "EXTDEF":
                ArrayList<String> defList = token.getOperands();
                for (String def : defList) {
                    int address = _symbolTable.getAddress(def).get();
                    objCode.addDefineSymbol(def, _symbolTable.getAddress(def).get());
                }
                break;
            case "EXTREF":
                ArrayList<String> refList = token.getOperands();
                for (String ref : refList) {
                    objCode.addReferSymbol(ref);
                }
                break;
            case "LTORG":
                ArrayList<Literal> literals = _literalTable.getLiteralMap();
                for (Literal lit : literals) {
                    String literal = lit.getLiteral();
                    String value = literal.substring(3, literal.length() - 1); // 리터럴에
```

```
                    if (literal.charAt(1) == 'X') {
                        if (value.length() % 2 != 0) {
                            value = "0" + value; // 짝수 길이가 아닐 경우 앞에 0을 추가
                        }
                        objCode.addText(locctr, value);
                        locctr += value.length() / 2; // 길이를 2로 나눈 값이 바이트 수임
                    } else if (literal.charAt(1) == 'C') {
                        // 문자 리터럴 처리: 각 문자를 ASCII 코드의 16진수로 변환
                        StringBuilder hexValue = new StringBuilder();
                        for (int i = 0; i < value.length(); i++) {
                            hexValue.append(String.format("%02X", (int) value.charAt(i))
                        }
                        objCode.addText(locctr, hexValue.toString());
                        locctr += value.length(); // 문자 수만큼 LOCCTR 증가
                    }
                }
                break;
            default:
                break;
            }
        }

        /**
         * 심볼 테이블을 String으로 변환하여 반환한다. Assembler.java에서 심볼 테이블을 출력하는 데에 사용
         *
         * @return 문자열로 변경된 심볼 테이블
         */
        public String getSymbolString() {
            return _symbolTable.toString();
        }

        /**
         * 리터럴 테이블을 String으로 변환하여 반환한다. Assembler.java에서 리터럴 테이블을 출력하는 데에 사
         *
         * @return 문자열로 변경된 리터럴 테이블
         */
        public String getLiteralString() {
            return _literalTable.toString();
        }

        /** 기계어 목록 테이블 */
        private InstructionTable _instTable;

        /** 토큰 테이블 */
        private ArrayList<Token> _tokens;

        /** 심볼 테이블 */
        private SymbolTable _symbolTable;

        /** 리터럴 테이블 */
        private LiteralTable _literalTable;

        private Optional<String> _sectionLabel;
        private Optional<Integer> _startAddress;
        private Optional<Integer> _programLength;
        private Optional<Integer> _endAddress;

    }
```

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.stream.Collectors;

/**
 * SIC/XE 머신을 위한 Assembler 프로그램의 메인 루틴이다.
 *
 * 작성 중 유의 사항
 * 1) Assembler.java 파일의 기존 코드를 변경하지 말 것. 다른 소스 코드의 구조는 본인의 편의에 따라 변경
 * 2) 새로운 클래스, 새로운 필드, 새로운 메소드 선언은 허용됨. 단, 기존의 필드와 메소드를 삭제하거나 대체하
 * 3) 예외 처리, 인터페이스, 상속 사용 또한 허용됨.
 * 4) 파일, 또는 콘솔창에 한글을 출력하지 말 것. (채점 상의 이유)
 *
 * 제공하는 프로그램 구조의 개선점을 제안하고 싶은 학생은 보고서의 결론 뒷부분에 첨부 바람. 내용에 따라 가산
 */
public class Assembler {
    public static void main(String[] args) {
        try {
            Assembler assembler = new Assembler("inst_table.txt");
            ArrayList<String> input = assembler.readInputFromFile("input.txt");
            ArrayList<ArrayList<String>> dividedInput = assembler.divideInput(input);
            dividedInput.size();

            ArrayList<ControlSection> controlSections = (ArrayList<ControlSection>) divi
                    .map(x -> assembler.pass1(x))
                    .collect(Collectors.toList());


            String symbolsString = controlSections.stream()
                    .map(x -> x.getSymbolString())
                    .collect(Collectors.joining("\n\n"));
            String literalsString = controlSections.stream()
                    .map(x -> x.getLiteralString())
                    .collect(Collectors.joining("\n\n"));

            assembler.writeStringToFile("output_symtab.txt", symbolsString);
            assembler.writeStringToFile("output_littab.txt", literalsString);

            ArrayList<ObjectCode> objectCodes = (ArrayList<ObjectCode>) controlSections.s
                    .map(x -> assembler.pass2(x))
                    .collect(Collectors.toList());

            String objectCodesString = objectCodes.stream()
                    .map(x -> x.toString())
                    .collect(Collectors.joining("\n\n"));

            assembler.writeStringToFile("output_objectcode.txt", objectCodesString);
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```

```
public Assembler(String instFile) throws FileNotFoundException, IOException {
    _instTable = new InstructionTable(instFile);
}

private ArrayList<ArrayList<String>> divideInput(ArrayList<String> input) {
    ArrayList<ArrayList<String>> divided = new ArrayList<ArrayList<String>>();
    String lastStr = input.get(input.size() - 1);

    ArrayList<String> tmpInput = new ArrayList<String>();
    for (String str : input) {
        if (str.contains("CSECT")) {
            if (!tmpInput.isEmpty()) {
                tmpInput.add(lastStr);
                divided.add(tmpInput);
                tmpInput = new ArrayList<String>();
                tmpInput.add(str);
            }
        } else {
            tmpInput.add(str);
        }
    }

    if (!tmpInput.isEmpty()) {
        divided.add(tmpInput);
    }

    return divided;
}

private ArrayList<String> readInputFromFile(String inputFileName) throws FileNotFound
    ArrayList<String> input = new ArrayList<String>();

    File file = new File(inputFileName);
    BufferedReader bufReader = new BufferedReader(new FileReader(file));

    String line = "";
    while ((line = bufReader.readLine()) != null)
        input.add(line);

    bufReader.close();

    return input;
}

private void writeStringToFile(String fileName, String content) throws IOException {
    File file = new File(fileName);

    BufferedWriter writer = new BufferedWriter(new FileWriter(file));
    writer.write(content);
    writer.close();
}

private ControlSection pass1(ArrayList<String> input) throws RuntimeException {
    return new ControlSection(_instTable, input);
}

private ObjectCode pass2(ControlSection controlSection) throws RuntimeException {
    return controlSection.buildObjectCode();
}
```

```
    private InstructionTable _instTable;
}
```

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.Optional;
import java.util.stream.Collectors;

public class LiteralTable {
    /**
     * 리터럴 테이블을 초기화한다.
     */
    public LiteralTable() {
        literalMap = new LinkedHashMap<String, Literal>();
    }

/**
 * 리터럴을 리터럴 테이블에 추가한다.
 *
 * @param literal 추가할 리터럴
 * @throws RuntimeException 비정상적인 리터럴 서식
 */
public void putLiteral(String literal) throws RuntimeException {
    //중복되는 리터럴이 없다면 추가함
    if (!literalMap.containsKey(literal)) {
        literalMap.put(literal, new Literal(literal));
    }


}

public void setLiteralAddress(String literal, int address) {
    literalMap.get(literal).setAddress(address);
}
public int setLiteralAddressAll(int address) {
    //모든 리터럴의 주소를 설정함
    //여기서 address는 현재 locctr값임
    //주소값을 할당하고 리터럴 크기만큼 locctr값을 증가시킨 후 모든 리터럴에 대해 반복함
    // 만약 이미 주소값이 할당되어 있다면 그냥 넘어감

    for (Literal lit : literalMap.values()) {
        if (!lit.getAddress().isPresent()) {
            lit.setAddress(address);
            address += lit.getSize();
        }
    }

    return address;
}

public Optional<Literal> searchLiteral(String literal) {
    return Optional.ofNullable(literalMap.get(literal));
}

// TODO: 추가로 필요한 method 구현하기.
```

```
    /**
     * 리터럴 테이블을 String으로 변환한다.
     */
    @Override
    public String toString() {
        return literalMap.values().stream()
                .map(Literal::toString)
                .collect(Collectors.joining("\n"));
    }

    public ArrayList<Literal> getLiteralMap() {
        return new ArrayList<>(literalMap.values());
    }

    /** 리터럴 맵. key: 리터럴 String, value: 리터럴 객체 */
    private HashMap<String, Literal> literalMap;
}

class Literal {
    /**
     * 리터럴 객체를 초기화한다.
     *
     * @param literal 리터럴 String
     */
    public Literal(String literal) {
        // TODO: 리터럴 객체 초기화.
        _literal = literal;
        _address = Optional.empty();
    }

    /**
     * 리터럴 String을 반환한다.
     *
     * @return 리터럴 String
     */
    public String getLiteral() {
        return _literal;
    }

    public int getSize() {
        //=x'00'과 같은 리터럴의 경우 1바이트를 차지함
        //=c'EOF'와 같은 리터럴의 경우 3바이트를 차지함

        if (_literal.startsWith("=X'")) {
            return 1;
        } else if (_literal.startsWith("=C'")) {
            return _literal.length() - 4;
        } else {
            return 0;
        }
    }

    /**
     * 리터럴의 주소를 반환한다. 주소가 지정되지 않은 경우, Optional.empty()를 반환한다.
     *
     * @return 리터럴의 주소
     */
    public Optional<Integer> getAddress() {
        return _address;
    }
```

```
// TODO: 추가로 선언한 field에 대한 getter 작성하기.

/**
 * 리터럴을 String으로 변환한다. 리터럴의 address에 관한 정보도 리턴값에 포함되어야 한다.
 */
@Override
public String toString() {
    return _address.map(addr -> String.format("%s 0x%04X", _literal, addr))
                   .orElse(String.format("%s not assigned", _literal));
}

public void setAddress(int address) {
    _address = Optional.of(address);
}

/** 리터럴 String */
private String _literal;
    /** 리터럴 주소. 주소가 지정되지 않은 경우 empty */
    private Optional<Integer> _address;

    // TODO: 추가로 필요한 field 선언하기.
}
```

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

public class InstructionTable {
    /**
     * 기계어 목록 파일을 읽어, 기계어 목록 테이블을 초기화한다.
     *
     * @param instFileName 기계어 목록이 적힌 파일
     * @throws FileNotFoundException 기계어 목록 파일 미존재
     * @throws IOException           파일 읽기 실패
     */


    public InstructionTable(String instFileName) throws FileNotFoundException, IOExceptio
        HashMap<String, InstructionInfo> instMap = new HashMap<String, InstructionInfo>(

        ArrayList<String> data = readFile(instFileName);
        data.forEach(x -> {
            String name = x.substring(0, x.indexOf('\t'));
            instMap.put(name, new InstructionInfo(x));
        });

        _instructionMap = instMap;
    }

    /**
     * 기계어 목록 테이블에서 특정 기계어를 검색한다.
```

```
     *
     * @param instructionName 검색할 기계어 명칭
     * @return 기계어 정보. 없을 경우 empty
     */
    public Optional<InstructionInfo> search(String instructionName) {
        // TODO: instructionMap에서 instructionName에 해당하는 명령어의 정보 반환하기.

        if (_instructionMap.containsKey(instructionName))
            return Optional.of(_instructionMap.get(instructionName));
        else
            return Optional.empty();
    }

    private ArrayList<String> readFile(String fileName) throws FileNotFoundException, IOE
        ArrayList<String> data = new ArrayList<String>();

        File file = new File(fileName);
        BufferedReader bufReader = new BufferedReader(new FileReader(file));

        String line = "";
        while ((line = bufReader.readLine()) != null)
            data.add(line);

        bufReader.close();

        return data;
    }

    /** 기계어 목록 테이블. key: 기계어 명칭, value: 기계어 정보 */
    private HashMap<String, InstructionInfo> _instructionMap;
}
```