숭실대학교 Soongsil University

시스템프로그래밍 2021 보고서

보고서 제출서약서

나는 숭실대학교 컴퓨터학부의 일원으로 명예를 지키면서 생활하고 있습니다.

나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.

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

과목	시스템프로그래밍 2021
과제명	프로젝트1b
담당교수	최 재 영 교 수
제출인	컴퓨터학부 20192698 심원준(출석번호 124)
제출일	2021년 5월 17일

차 례

1장. 프로젝트 목적/동기

이 프로젝트는 SIC/XE머신의 어셈블러를 구현하여 입력된 SIC/XE코드를 Object Program Code로 바꾸는 것을 JAVA언어로 구현함을 목적으로한다.

2장. 설계/구현 아이디어

지난번 프로젝트1에서 C로 제작한 어셈블러를 자바 언어로 바꾸어 제작하고자 하였다. 이를 위해서 Assembler, tokenTable, instTable,LabelTable의 클래스를 활용하여 각각 어셈블러의 메인 함수들, 토큰, 명령어 appendix, Symbol table/litertable을 만들고자 하였다. Assembler.java에서 token객체 배열을 만든 뒤, 이를 tokentable클래스 객체 생성자에 넣어주고, symbol/literaltable또한 각각의 컨트롤섹션 별로 분리된 tokentable클래스 객체에 넣어주어 컨트롤섹션별로 구분할 수 있도록 해주었다. 또한 TokenTable클래스에 ExtREF, EXTDEF를 저장할 수 있는 <String> ArrayList를 각각 넣어주어 pass2과정에서 활용할 수 있도록 해주고자 하였다. Pass1에서 input의 한줄한줄을 putToken을 이용하여 파싱 한 뒤에, 이를 다시 임시 Token객체에 받도록 하여 locctr값을 location에 넣어주고, set함수를 이용하여 다시 넣어주는 방식으로 locctr를 각각 저장해 주었다.

Pass2에서는 pass1의 결과를 바탕으로 하여 setFlag함수를 활용하여 flag를 설정해주고, location address 값을 계산하여 objectcode를 계산, String의 형태로 저장해주어 C언어보다 간단한 계산이 가능하였다.

3장 수행결과(구현 화면 포함)

```
COPY    0
FIRST   0
CLOOP   3
ENDFIL  17
RETADR  2A
LENGTH  2D
BUFFER  33
BUFEND  1033
MAXLEN  1000
RDREC   0
RLOOP   9
EXIT    20
INPUT   27
MAXLEN  28
WRREC   0
WLOOP   6
```

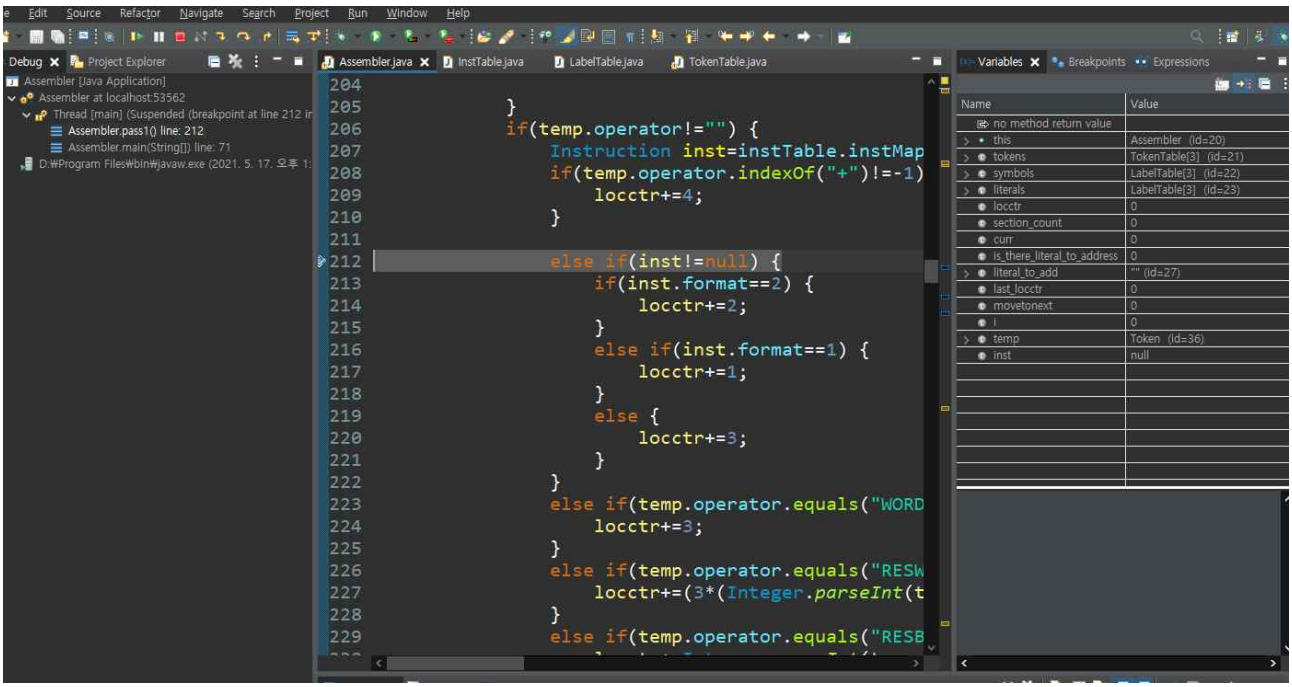
EOF 30
05 1B

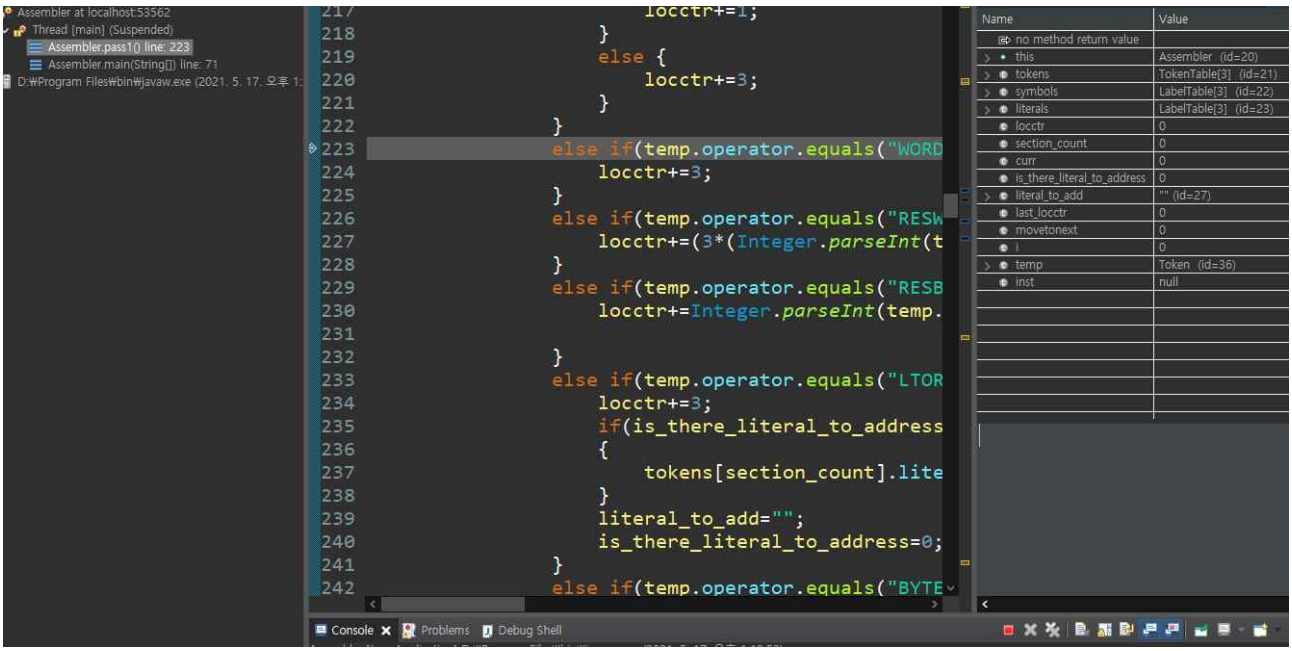
pass1뒤 생성한 symboltable과 literal table(확인을 위해 이 경우.txt로 하였으나 소스코드에는 .txt를 제거하였음)

4장 결론 및 보충할 점

objectcode를 계산하는 과정에서 symboltable에서 주소값을 불러와야 하는데, 이 과정에서 함수를 호출하는 데에서 index오류가 발생하였다. 이 오류를 해결하기 위해서 return값등을 조정해보았지만 해결되지 않았고, pass2가 모두 완료되지 못한 채로 프로그램이 종료되어 objectcode를 알맞게 출력하지 못하는 문제가 발생하였다. 문제의 원인은 파악되나, 해결책을 시간내에 찾지 못하여 필요한 출력의 반(pass1이후의 symboltable과 literal table 출력)밖에 이루어내지 못하였다.

5장 디버깅


Edit Source Refactor Navigate Search Project Run Window Help
Debug Project Explorer
Assembler [Java Application]
Assembler at localhost:53562
Thread [main] (Suspended (breakpoint at line 212 in
Assembler.pass1() line: 212
Assembler.main(String[]) line: 71
D:₩Program Files₩bin₩javaw.exe (2021. 5. 17. 오후 1:
Assembler.java
InstTable.java
LabelTable.java
TokenTable.java
204
205 }
206 if(temp.operator!="") {
207 Instruction inst=instTable.instMap
208 if(temp.operator.indexOf("+")!=-1)
209 locctr+=4;
210 }
211
212 else if(inst!=null) {
213 if(inst.format==2) {
214 locctr+=2;
215 }
216 else if(inst.format==1) {
217 locctr+=1;
218 }
219 else {
220 locctr+=3;
221 }
222 }
223 else if(temp.operator.equals("WORD
224 locctr+=3;
225 }
226 else if(temp.operator.equals("RESW
227 locctr+=(3*(Integer.parseInt(t
228 }
229 else if(temp.operator.equals("RESB
Variables
Breakpoints
Expressions
Name
Value
no method return value
this
Assembler (id=20)
tokens
TokenTable[3] (id=21)
symbols
LabelTable[3] (id=22)
literals
LabelTable[3] (id=23)
locctr
0
section_count
0
curr
0
is_there_literal_to_address
0
literal_to_add
"" (id=27)
last_locctr
0
movetonext
0
i
0
temp
Token (id=36)
inst
null


Assembler at localhost:53562
Thread [main] (Suspended)
Assembler.pass1() line: 223
Assembler.main(String[]) line: 71
D:₩Program Files₩bin₩javaw.exe (2021. 5. 17. 오후 1:
217 locctr+=1;
218 }
219 else {
220 locctr+=3;
221 }
222 }
223 else if(temp.operator.equals("WORD
224 locctr+=3;
225 }
226 else if(temp.operator.equals("RESW
227 locctr+=(3*(Integer.parseInt(t
228 }
229 else if(temp.operator.equals("RESB
230 locctr+=Integer.parseInt(temp.
231
232 }
233 else if(temp.operator.equals("LTOR
234 locctr+=3;
235 if(is_there_literal_to_address
236 {
237 tokens[section_count].lite
238 }
239 literal_to_add="";
240 is_there_literal_to_address=0;
241 }
242 else if(temp.operator.equals("BYTE
Name
Value
no method return value
this
Assembler (id=20)
tokens
TokenTable[3] (id=21)
symbols
LabelTable[3] (id=22)
literals
LabelTable[3] (id=23)
locctr
0
section_count
0
curr
0
is_there_literal_to_address
0
literal_to_add
"" (id=27)
last_locctr
0
movetonext
0
i
0
temp
Token (id=36)
inst
null
Console
Problems
Debug Shell

6장 소스코드(+주석)

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.io.*;
/**
 * Assembler: 이 프로그램은 SIC/XE 머신을 위한 Assembler 프로그램의 메인루틴이다. 프로그램의 수행 작업은 다음과 같다.
 * 1) 처음 시작하면 Instruction 명세를 읽어들여서 assembler를 세팅한다.
 *
 * 2) 사용자가 작성한 input 파일을 읽어들인 후 저장한다
 *
 * 3) input 파일의 문장들을 단어별로 분할하고 의미를 파악해서 정리한다. (pass1)
 *
 * 4) 분석된 내용을바탕으로 컴퓨터가 사용할 수 있는 object code를 생성한다. (pass2)
 *
 *
 * 작성중의 유의사항:
 *
 * 1) 새로운 클래스, 새로운 변수, 새로운 함수 선언은 얼마든지 허용됨. 단, 기존의 변수와 함수들을 삭제하거나 완전히 대체하는 것은
 * 안된다.
 *
 * 2) 마찬가지로 작성된 코드를 삭제하지 않으면 필요에 따라 예외처리, 인터페이스 또는 상속 사용 또한 허용됨
 *
 * 3) 모든 void 타입의 리턴값은 유저의 필요에 따라 다른 리턴 타입으로 변경 가능.
 *
 * 4) 파일, 또는 콘솔창에 한글을 출력시키지말 것. (채점상의 이유. 주석에 포함된 한글은 상관 없음)
 *
 * + 제공하는 프로그램 구조의 개선방법을 제안하고 싶은 분들은 보고서의 결론 뒷부분에 첨부 바랍니다. 내용에 따라 가산점이 있을 수
 * 있습니다.
 */

public class Assembler {
	/** instruction 명세를 저장한 공간 */
	public InstTable instTable;
	/** 읽어들인 input 파일의 내용을 한 줄씩 저장하는 공간. */
	ArrayList<String> lineList;
```

```
	/** 프로그램의 section별로 symbol table을 저장하는 공간 */
	ArrayList<LabelTable> symtabList;
	/** 프로그램의 section별로 literal table을 저장하는 공간 */
	ArrayList<LabelTable> literaltabList;
	/** 프로그램의 section별로 프로그램을 저장하는 공간 */
	ArrayList<TokenTable> TokenList;
	/**
	 * Token, 또는 지시어에 따라 만들어진 오브젝트 코드들을 출력 형태로 저장하는 공간. 필요한 경우 String 대신 별도의 클래스를
	 * 선언하여 ArrayList를 교체해도 무방함.
	 */
	ArrayList<String> codeList;

	/**
	 * 클래스 초기화. instruction Table을 초기화와 동시에 세팅한다.
	 *
	 * @param instFile : instruction 명세를 작성한 파일 이름.
	 * @throws IOException
	 */
	public int sttadd;//스타팅주소값
	public Assembler(String instFile) throws IOException {
		instTable = new InstTable(instFile);
		lineList = new ArrayList<String>();
		symtabList = new ArrayList<LabelTable>();
		literaltabList = new ArrayList<LabelTable>();
		TokenList = new ArrayList<TokenTable>();
		codeList = new ArrayList<String>();
	}

	/**
	 * 어셈블러의 메인 루틴
	 * @throws IOException
	 */
	public static void main(String[] args) throws IOException {
		Assembler assembler = new Assembler("inst.data");

		assembler.loadInputFile("input.txt");
		assembler.pass1();


		assembler.printSymbolTable("symtab_20192698");

		assembler.printLiteralTable("literaltab_20192698");
		assembler.pass2();

		assembler.printObjectCode("output_20192698");

	}

	/**
	 * inputFile을 읽어들여서 lineList에 저장한다.
	 *
	 * @param inputFile : input 파일 이름.
	 * @throws IOException
	 */
	private void loadInputFile(String inputFile) throws IOException {
		File input=new File(inputFile);
		FileReader finput=new FileReader(input);
		BufferedReader bufReader=new BufferedReader(finput);
		String line="";
		int i=0;

		while((line=bufReader.readLine())!=null) {
			lineList.add(line);
			line="";
			i++;
		}

	}

	/**
```

```
        * pass1 과정을 수행한다.
        *
        * 1) 프로그램 소스를 스캔하여 토큰 단위로 분리한 뒤 토큰 테이블을 생성.
        *
        * 2) symbol, literal 들을 SymbolTable, LiteralTable에 정리.
        *
        * 주의사항: SymbolTable, LiteralTable, TokenTable은 프로그램의 section별로 하나씩 선언되어야 한다.
        *
        * @param inputFile : input 파일 이름.
        */
       private void pass1() {
            TokenTable[] tokens=new TokenTable[3];
            LabelTable[] symbols=new LabelTable[3];
            LabelTable[] literals=new LabelTable[3];
            for(int i=0;i<3;i++) {
                symbols[i]=new LabelTable();
                literals[i]=new LabelTable();
                tokens[i]=new TokenTable(symbols[i],literals[i],instTable);
            }
            int locctr=0;
            int section_count=0;
            sttadd=0;
            int curr=0;
            int is_there_literal_to_address=0;
            String literal_to_add=new String();
            int last_locctr=sttadd;
            int movetonext=0;
            for(int i=0;i<lineList.size();i++)
            {

tokens[section_count].putToken(lineList.get(i));
                Token temp=tokens[section_count].getToken(curr);
                temp.location=locctr;
                if(i==0) {

sttadd=Integer.parseInt(temp.operand[0],16);

temp.location=locctr;

tokens[section_count].tokenList.set(curr, temp);
                }
                if(temp.operator!="")
                {


if(temp.operator.equals("CSECT"))
                {

last_locctr=locctr;

locctr=0;//controlsection바뀌어서 초기화해줌.

temp.location=locctr;

tokens[section_count+1].tokenList.add(temp);

tokens[section_count].tokenList.remove(curr);

movetonext=1;

curr=0;

                }
                else if(temp.operator.equals("EXTREF"))
                {
                    for(int refs=0;refs<3;refs++)
                    {

        if(!temp.operand[refs].equals(""))

        {


tokens[section_count].reference.add(temp.operand[refs]);

        }
```

```
                                        }
                                }
                                else if(temp.operator.equals("EXTDEF"))
                                {
                                        for(int defs=0;defs<3;defs++)
                                        {
        if(!temp.operand[defs].equals(""))
        {

tokens[section_count].EXTDEF.add(temp.operand[defs]);

        }
                                        }
                                }

                        }
                        if(temp.operator==""&&temp.operand[0]==""&&temp.label==""&&temp.comment!="")
                        {//comment만 존재하는 것.

temp.location=locctr;

tokens[section_count].tokenList.set(curr, temp);
                        }
                        if(temp.label!="")
                        {

if(temp.operator.equals("EQU"))
                                {
                                        Token pretemp=tokens[section_count].getToken(curr-1);

if(pretemp.operator.equals("EQU"))
                                        {
        Token temp3=tokens[section_count].getToken(curr-2);

temp.location=Integer.parseInt(temp3.operand[0],10);

tokens[section_count].symTab.putName(temp.label,temp.location);

        tokens[section_count].tokenList.set(curr, temp);
                                        }
                                        else {

tokens[section_count].symTab.putName(temp.label, temp.location);

        tokens[section_count].tokenList.set(curr, temp);
                                        }
                                }
                                else
                                {

temp.location=locctr;

tokens[section_count].symTab.putName(temp.label,temp.location);

tokens[section_count].tokenList.set(curr, temp);
                                }

                        }
                        if(temp.operator!="") {
                                Instruction inst=instTable.instMap.get(temp.operator);

if(temp.operator.indexOf("+")!=-1) {

locctr+=4;
                                }
```

```
                                    e l s e
if(inst!=null) {

if(inst.format==2) {
        locctr+=2;
                                    }
                                    e l s e
if(inst.format==1) {
        locctr+=1;
                                    }
                                    else {
        locctr+=3;
                                    }
                                }
                                e l s e
if(temp.operator.equals("WORD")) {

locctr+=3;
                                }
                                e l s e
if(temp.operator.equals("RESW")) {

locctr+=(3*(Integer.parseInt(temp.operand[0],10)
));
                                }
                                e l s e
if(temp.operator.equals("RESB")) {

locctr+=Integer.parseInt(temp.operand[0],10);

                                }
                                e l s e
if(temp.operator.equals("LTORG")) {

locctr+=3;

if(is_there_literal_to_address>=1)
                                    {


tokens[section_count].literalTab.modifyName(lite
ral_to_add, locctr-3);
                                    }
literal_to_add="";

is_there_literal_to_address=0;
                                }
                                e l s e
if(temp.operator.equals("BYTE")) {

locctr+=1;
                                }
                            }

if(temp.operand[0].indexOf("=")==0) {//literal
                                String[]
temporary;

temporary=temp.operand[0].split("₩'");

if(tokens[section_count].literalTab.locationList.siz
e()>0)
                                    {

if(tokens[section_count].literalTab.search(tempor
ary[1])==-1)
                                        {

        literal_to_add=temporary[1];


tokens[section_count].literalTab.putName(tempor
ary[1],555 );//임시값

        is_there_literal_to_address=1;


                                        }

                                    }
                                    else {

literal_to_add=temporary[1];

tokens[section_count].literalTab.putName(tempor
ary[1],555 );//임시값

is_there_literal_to_address=1;
```

```
                        }
                    }
                    if(movetonext==1||
temp.operator.equals("END"))
                    {//지난섹션의 값 저장.

if(is_there_literal_to_address>=1) {

if(locctr==0) {


literals[section_count].modifyName(literal_to_add
, last_locctr);

        is_there_literal_to_address-=1;

        literal_to_add="";
                                    }
                                    else {


literals[section_count].modifyName(literal_to_add
,locctr);
                                    }

                                }
                                movetonext=0;

section_count+=1;

symtabList.add(tokens[section_count-1].symTab)
;

literaltabList.add(tokens[section_count-1].literal
Tab);

TokenList.add(tokens[section_count-1]);
                }
                    else {

tokens[section_count].tokenList.set(curr, temp);
                    }
                    curr++;//section내에서
의 순서.

      }//for문끝
}

      /**
       * 작성된 SymbolTable들을 출력형태에
맞게 출력한다.
       *
       * @param fileName : 저장되는 파일 이
름
       * @throws IOException
       */
      private void printSymbolTable(String
fileName) throws IOException {
                if(fileName=="") {
                    for(int i=0;i<3;i++) {
                        TokenTable
temp=TokenList.get(i);
                        LabelTable
temperar=temp.symTab;
                        for(int
j=0;j<temperar.label.size();j++) {

System.out.print(temperar.label.get(j));

System.out.print("₩t");
                            String
temmm=Integer.toHexString(temperar.search(tem
perar.label.get(j)));

System.out.println(temmm.toUpperCase());
                        }
                    }
                }
                else {
                    File symfile=new
File(fileName);
                    FileWriter ftowr=new
FileWriter(symfile);
                    BufferedWriter
bw=new BufferedWriter(ftowr);
                    for(int i=0;i<3;i++) {
                        TokenTable
temp=TokenList.get(i);
                        LabelTable
temperar=temp.symTab;
                        for(int
```

```
j=0;j<temperar.label.size();j++) {

bw.write(temperar.label.get(j));

bw.write("₩t");
                                        String
temmm=Integer.toHexString(temperar.search(tem
perar.label.get(j)));

bw.write(temmm.toUpperCase());

bw.newLine();
                                    }
                            }
                            bw.close();
                    }
            }

            /**
             * 작성된 LiteralTable들을 출력형태에 맞
게 출력한다.
             *
             * @param fileName : 저장되는 파일 이
름
             * @throws IOException
             */
            private void printLiteralTable(String
fileName) throws IOException {
                    if(fileName=="") {
                            for(int i=0;i<3;i++) {
                                    TokenTable
temp=TokenList.get(i);
                                    LabelTable
temperar=temp.literalTab;
                                    for(int
j=0;j<temperar.label.size();j++) {

System.out.print(temperar.label.get(j));

System.out.print("₩t");
                                        String
temmm=Integer.toHexString(temperar.search(tem
perar.label.get(j)));

System.out.println(temmm.toUpperCase());
                                    }
                            }
                    }
                    else {
                            File litfile=new
File(fileName);
                            FileWriter ftowr=new
FileWriter(litfile);
                            BufferedWriter
bw=new BufferedWriter(ftowr);
                            for(int i=0;i<3;i++) {
                                    TokenTable
temp=TokenList.get(i);
                                    LabelTable
temperar=temp.literalTab;
                                    for(int
j=0;j<temperar.label.size();j++) {

bw.write(temperar.label.get(j));

bw.write("₩t");
                                        String
temmm=Integer.toHexString(temperar.search(tem
perar.label.get(j)));

bw.write(temmm.toUpperCase());

bw.newLine();
                                    }
                            }
                            bw.close();
                    }


            }

            /**
             * pass2 과정을 수행한다.
             *
              * 1) 분석된 내용을 바탕으로 object
code를 생성하여 codeList에 저장.
             */
            private void pass2() {

                    for(int i=0;i<3;i++)//flag 설정
                    {
                            TokenTable
```

```
temp=TokenList.get(i);
                        for(int j=0;j<temp.tokenList.size();j++)
                        {
temp.makeObjectCode(j);
                        }
                }

        }

        /**
         * 작성된 codeList를 출력형태에 맞게 출력한다.
         *
         * @param fileName : 저장되는 파일 이름
         * @throws IOException
         */
        private void printObjectCode(String fileName) throws IOException {
                if(fileName.equals(""))
                {
                        for(int i=0;i<3;i++)
                        {
                                TokenTable temp=TokenList.get(i);
                                for(int j=0;j<temp.tokenList.size();j++)
                                {
                                        Token temperary=temp.getToken(j);

if(!temperary.objectCode.equals(""))
                                        {

System.out.println(temperary.objectCode);
                                        }
                                }

                        }
                }
                else
                {
                        File fw=new File(fileName);
                        FileWriter fr=new FileWriter(fw);
                        BufferedWriter bf=new BufferedWriter(fr);
                        for(int i=0;i<3;i++)
                        {
                                TokenTable temp=TokenList.get(i);
                                for(int j=0;j<temp.tokenList.size();j++)
                                {
                                        Token temperary=temp.getToken(j);

if(!temperary.objectCode.equals(""))
                                        {

                bf.write(temperary.objectCode);

                                        }
                                }
                                bf.newLine();

                        }
                        bf.close();
                }

        }

}


import java.util.HashMap;
import java.io.*;
/**
 * 모든 instruction의 정보를 관리하는 클래스. instruction data들을 저장한다 또한 instruction 관련 연산,
 * 예를 들면 목록을 구축하는 함수, 관련 정보를 제공하는 함수 등을 제공 한다.
 */
public class InstTable {
```

```
	/**
	 * inst.data 파일을 불러와 저장하는 공간. 명령어의 이름을 집어넣으면 해당하는 Instruction의 정보들을 리턴할 수 있다.
	 */
	HashMap<String, Instruction> instMap;

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

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

	/**
	 * 입력받은 이름의 파일을 열고 해당 내용을 파싱하여 instMap에 저장한다.
	 * @throws FileNotFoundException
	 */
	public Instruction inst[]=new Instruction[256];//instruction객체 배열
	public void openFile(String fileName) throws IOException {
		File file =new File(fileName);
		FileReader ftor=new FileReader(file);
		BufferedReader bufReader=new BufferedReader(ftor);
		String line="";
		int i=0;

while((line=bufReader.readLine())!=null) {
			inst[i] = new Instruction(line);

instMap.put(inst[i].instruction,inst[i]);
			line="";
			i++;
		}
		bufReader.close();

	}

	// get, set, search 등의 함수는 자유 구현

}

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


	String instruction;
	int opcode;
	int format;
	int ops;

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

	/**
	 * 일반 문자열을 파싱하여 instruction 정보를 파악하고 저장한다.
	 *
	 * @param line : instruction 명세파일로부터 한줄씩 가져온 문자열
	 */
	public void parsing(String line) {
		String[]splitstring=line.split(" ");
		instruction=splitstring[0];
```

```
format=Integer.parseInt(splitstring[1],10);

opcode=Integer.parseInt(splitstring[2],16);

ops=Integer.parseInt(splitstring[3],10);
	}

	// 그 외 함수 자유 구현

}

import java.util.ArrayList;

/**
 * symbol, literal과 관련된 데이터와 연산을 소유한다. section 별로 하나씩 인스턴스를 할당한다.
 */
public class LabelTable {
	ArrayList<String> label;
	ArrayList<Integer> locationList;
	// external 선언 및 처리방법을 구현한다.
	public LabelTable() {
		label=new ArrayList<String>();
		locationList=new ArrayList<Integer>();
	}
		/**


	 * 새로운 symbol과 literal을 table에 추가한다.
	 *
	 * @param label	: 새로 추가되는 symbol 혹은 literal의 lable
	 * @param location : 해당 symbol 혹은 literal이 가지는 주소값 주의 : 만약 중복된 symbol, literal이
	 *		putName을 통해서 입력된다면 이는 프로그램 코드에 문제가 있음을 나타낸다. 매칭되는 주소값의 변경은
	 *		modifylable()을 통해서 이루어져야 한다.
	 */
	public void putName(String label, int location) {
		this.label.add(label);
		this.locationList.add(location);

	}

	/**
	 * 기존에 존재하는 symbol, literal 값에 대해서 가리키는 주소값을 변경한다.
	 *
	 * @param lable	: 변경을 원하는 symbol, literal의 label
	 * @param newLocation : 새로 바꾸고자 하는 주소값
	 */
	public void modifyName(String lable, int newLocation) {

this.locationList.set(this.label.indexOf(lable),newLocation);
	}

	/**
	 * 인자로 전달된 symbol, literal이 어떤 주소를 지칭하는지 알려준다.
	 *
	 * @param label : 검색을 원하는 symbol 혹은 literal의 label
	 * @return address: 가지고 있는 주소값. 해당 symbol, literal이 없을 경우 -1 리턴
	 */
	public int search(String label) {
		int address = -1;

address=this.locationList.get(this.label.indexOf(label));
		return address;
	}

}

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.io.*;
/**
 * 사용자가 작성한 프로그램 코드를 단어별로 분할
```

```
한 후, 의미를 분석하고, 최종 코드로 변환하는 과정을 총괄하는 클래스이다.
 *
 * pass2에서 object code로 변환하는 과정은 혼자 해결할 수 없고 symbolTable과 instTable의 정보가 필요하므로
 * 이를 링크시킨다. section 마다 인스턴스가 하나씩 할당된다.
 *
 */
public class TokenTable {
	public static final int MAX_OPERAND = 3;

	/* bit 조작의 가독성을 위한 선언 */
	public static final int nFlag = 32;
	public static final int iFlag = 16;
	public static final int xFlag = 8;
	public static final int bFlag = 4;
	public static final int pFlag = 2;
	public static final int eFlag = 1;

	/* Token을 다룰 때 필요한 테이블들을 링크시킨다. */
	LabelTable symTab;
	LabelTable literalTab;
	InstTable instTab;

	/** 각 line을 의미별로 분할하고 분석하는 공간. */
	ArrayList<Token> tokenList;
	ArrayList<String>reference;//reference를 별도로 저장하고자함.
	ArrayList<String>EXTDEF;
	/**
	 * 초기화하면서 symTable과 instTable을 링크시킨다.
	 *
	 * @param symTab     : 해당 section과 연결되어있는 symbol table
	 * @param literaTab : 해당 section과 연결되어있는 literal table
	 * @param instTab    : instruction 명세가 정의된 instTable
	 */
	public TokenTable(LabelTable symTab, LabelTable literalTab, InstTable instTab) {
		this.symTab=symTab;
		this.literalTab=literalTab;
		this.instTab=instTab;
		tokenList = new ArrayList<Token>();
		reference = new ArrayList<String>();
		EXTDEF = new ArrayList<String>();
	}

	/**
	 * 일반 문자열을 받아서 Token단위로 분리시켜 tokenList에 추가한다.
	 *
	 * @param line : 분리되지 않은 일반 문자열
	 */
	public void putToken(String line) {
		tokenList.add(new Token(line));
	}

	/**
	 * tokenList에서 index에 해당하는 Token을 리턴한다.
	 *
	 * @param index
	 * @return : index번호에 해당하는 코드를 분석한 Token 클래스
	 */
	public Token getToken(int index) {
		return tokenList.get(index);
	}

	/**
	 * Pass2 과정에서 사용한다. instruction table, symbol table 등을 참조하여 objectcode를 생성하고, 이를
	 * 저장한다.
	 *
	 * @param index
	 */
	public void makeObjectCode(int index) {
```

```
HashMap<String,String>registers=new HashMap<String,String>();
                registers.put("A", "0");
                registers.put("X", "1");
                registers.put("S", "4");
                registers.put("T", "5");
                registers.put("", "0");


                Token temperary=getToken(index);

if(temperary.operator.indexOf("+")!=-1) {

temperary.setFlag(TokenTable.eFlag, 1);
                }
                else
                {

temperary.setFlag(TokenTable.eFlag, 0);
                }

if(temperary.operand[0].indexOf("@")!=-1)
                {

temperary.setFlag(TokenTable.nFlag, 1);
                }
                else
                {

temperary.setFlag(TokenTable.nFlag, 0);
                }

if(temperary.operand[0].indexOf("#")!=-1)
                {

temperary.setFlag(TokenTable.iFlag, 1);

                }
                else
                {

temperary.setFlag(TokenTable.iFlag, 0);
                }

if(temperary.operand[1].equals("X"))
                {

temperary.setFlag(TokenTable.xFlag, 1);
                }
                else
                {

temperary.setFlag(TokenTable.xFlag, 0);
                }

if(temperary.getFlag(TokenTable.nFlag)==0 && temperary.getFlag(TokenTable.iFlag)==0)
                {

temperary.setFlag(TokenTable.nFlag, 1);

temperary.setFlag(TokenTable.iFlag, 1);//simple addressing
                }
                int is_reference_=0;
                for(int re=0;re<reference.size();re++)
                {

if(temperary.operand[0].contains(reference.get(re))==true)//reference를 operand로 가지는 경우 p register 0
                        {

is_reference_=1;
                        }
                }
                if(is_reference_==0)
                {

temperary.setFlag(TokenTable.pFlag, 1);
                }
                //objectcode

if(temperary.getFlag(TokenTable.nFlag)==1 && temperary.getFlag(TokenTable.iFlag)==1 )
                {

temperary.objectCode+=(Integer.toHexString(instTab.instMap.get(temperary.operator).opcode+3));
```

```
                }
                else if(temperary.getFlag(TokenTable.nFlag)==1)
                {

temperary.objectCode+=(Integer.toHexString(instTab.instMap.get(temperary.operator).opcode+2));
                }
                else if(temperary.getFlag(TokenTable.iFlag)==1)
                {

temperary.objectCode+=(Integer.toHexString(instTab.instMap.get(temperary.operator).opcode+1));
                }
                int xbpe=0;

if(temperary.getFlag(TokenTable.xFlag)==1)
                {
                        xbpe+=8;
                }

if(temperary.getFlag(TokenTable.pFlag)==1)
                {
                        xbpe+=2;
                }

if(temperary.getFlag(TokenTable.eFlag)==1)
                {
                        xbpe+=1;
                }

temperary.objectCode+=Integer.toString(xbpe);

if(instTab.instMap.get(temperary.operator)!=null)
                {

if(instTab.instMap.get(temperary.operator).format==1)
                        {

temperary.objectCode=Integer.toHexString(instTab.instMap.get(temperary.operator).opcode);
                        }
                        else if(instTab.instMap.get(temperary.operator).format==2)
                        {

temperary.objectCode=Integer.toHexString(instTab.instMap.get(temperary.operator).opcode)+registers.get(temperary.operand[0])+registers.get(temperary.operand[1]);
                        }
                        else if(instTab.instMap.get(temperary.operator).format==3)
                        {

if(temperary.operand[0].contains("="))
                                {

if(index<tokenList.size()-1)
                                        {

                                Token cal=getToken(index+1);

                                String[] ttttl=temperary.operand[0].split("₩'");

                                int temlit=literalTab.search(ttttl[1]);

                                int tempob=temlit-cal.location;


temperary.objectCode+=Integer.toHexString(tempob);
                                        }

                                }
                                else if(temperary.operand[0].contains("#"))
                                {

temperary.objectCode+=temperary.operand[0].substring(1);
                                }
                                else if(temperary.operand[0].contains("@"))
                                {

if(index<tokenList.size()-1)
```

```
                                        {
        Token cal=getToken(index+1);
        int temlit=symTab.search(temperary.operand[0].substring(1));
        int tempob=temlit-cal.location;
if(Integer.toHexString(tempob).length()<3)
        {
if(Integer.toHexString(tempob).length()==1)
                {
temperary.objectCode+=("00"+Integer.toHexString(tempob));
                }
                else if(Integer.toHexString(tempob).length()==2)
                {
temperary.objectCode+=("0"+Integer.toHexString(tempob));
                }
                else if(Integer.toHexString(tempob).length()==0)
                {
temperary.objectCode+="000";
                }
        }
        else
        {
temperary.objectCode+=Integer.toHexString(tempob);
        }
                                                }
                                        }
                                        else
                                        {
                                                int temlit=symTab.search(temperary.operand[0]);
if(temlit!=-1)
                                                {
        if(index<tokenList.size()-1)
        {
                Token cal=getToken(index+1);
                if(temlit<cal.location)
                {
                        int tem=cal.location-temlit-1;
                        tem=4095-tem;
temperary.objectCode+=Integer.toHexString(tem);
                }
```

```
                else

                {


temperary.objectCode+ =Integer.toHexString(temlit-cal.location);

                }

        }
                                        }
                                        else
                                        {

        temperary.objectCode+ ="000";
                                        }
                                }


                        }
                        else
                        {

temperary.objectCode="";
                        }

                }
                else if(temperary.operator.contains("+ "))//4형식.
                {
                        String temoper=temperary.operator.substring(1);

if(temperary.operand[0].contains("="))
                        {

if(index<tokenList.size()-1)
                                {
                                        Token cal=getToken(index+ 1);

String[] ttttl=temperary.operand[0].split("₩'");
                                        int tempob=temlit-cal.location;

temperary.objectCode+ =Integer.toHexString(tempob);
                                }

                        }
                        else if(temperary.operand[0].contains("#"))
                        {

temperary.objectCode+ =temperary.operand[0].substring(1);
                        }
                        else if(temperary.operand[0].contains("@"))
                        {

if(index<tokenList.size()-1)
                                {
                                        Token cal=getToken(index+ 1);
                                        int temlit=symTab.search(temperary.operand[0].substring(1));
                                        int tempob=temlit-cal.location;

if(Integer.toHexString(tempob).length()<5)
                                        {


if(Integer.toHexString(tempob).length()==1)

        {


temperary.objectCode+ =("0000"+ Integer.toHexString(tempob));

        }

        else if(Integer.toHexString(tempob).length()==2)
```

```
        {

temperary.objectCode+ =("000"+ Integer.toHexStri
ng(tempob));

        }

        e l s e
if(Integer.toHexString(tempob).length()==0)

        {

temperary.objectCode+ ="00000";

        }

        e l s e
if(Integer.toHexString(tempob).length()==3)

        {

temperary.objectCode+ =("00"+ Integer.toHexStrin
g(tempob));

        }

        e l s e
if(Integer.toHexString(tempob).length()==4)

        {

temperary.objectCode+ =("0"+ Integer.toHexString
(tempob));

        }
                                    }
                                    else
                                    {

temperary.objectCode+ =Integer.toHexString(tem
pob);
                                    }

                                }
                            }
                            else
                            {
                                i n t
temlit=symTab.search(temperary.operand[0]);
                                if(temlit!=-1)
                                {

if(index<tokenList.size()-1)
                                    {
        Token cal=getToken(index+ 1);

        if(temlit<cal.location)

        {

                int tem=cal.location-temlit-1;

                tem=4095-tem;


temperary.objectCode+ =Integer.toHexString(tem)
;

        }

        else

        {


temperary.objectCode+ =Integer.toHexString(teml
it-cal.location);

        }
                                    }
                                }
                                else
                                {
```

```
temperary.objectCode+ ="00000";
                                        }

                                }


                        }
                        else
                        {

temperary.objectCode="";
                        }
                        tokenList.set(index, temperary);

                }

                /**
                 * index번호에 해당하는 object code를 리턴한다.
                 *
                 * @param index
                 * @return : object code
                 */
                public String getObjectCode(int index) {
                        r e t u r n tokenList.get(index).objectCode;
                }

}

/**
 * 각 라인별로 저장된 코드를 단어 단위로 분할한 후 의미를 해석하는 데에 사용되는 변수와 연산을 정의한다. 의미 해석이 끝나면 pass2에서
 * object code로 변형되었을 때의 바이트 코드 역시 저장한다.
 */
class Token {
        // 의미 분석 단계에서 사용되는 변수들
        int location;
        String label;
        String operator;
        String[] operand;
        String comment;
        char nixbpe;

        // object code 생성 단계에서 사용되는 변수들
        String objectCode;
        int byteSize;

        /**
         * 클래스를 초기화 하면서 바로 line의 의미 분석을 수행한다.
         *
         * @param line 문장단위로 저장된 프로그램 코드
         */
        public Token(String line) {
                label="";
                operator="";
                operand=new String[3];
                for(int i=0;i<3;i++) {
                        operand[i]="";
                }
                comment="";
                nixbpe=0;
                objectCode="";
                parsing(line);
        }

        /**
         * line의 실질적인 분석을 수행하는 함수. Token의 각 변수에 분석한 결과를 저장한다.
         *
         * @param line 문장단위로 저장된 프로그램 코드.
         */
        public void parsing(String line) {

String[]splitstring=line.split("₩t");
                        ArrayList<String>ss=new ArrayList<>(Arrays.asList(splitstring));
                char []tmp=line.toCharArray();
                if(line.indexOf("₩t")==0) {//no label

operator=splitstring[1];
                        ss.remove(0);
                        ss.remove(0);
                }
                  else  if(tmp[0]=='.') {//only
```

```
comment
                        comment=line;
                        ss.remove(0);
                        ss.clear();
                }
                else {
                        label=splitstring[0];
                        ss.remove(0);

operator=splitstring[1];
                        ss.remove(0);
                }
                 if(ss.size()>0)  {//operand또는
comment가 남아있으면
                        if(ss.size()==2) {
                                        String
temp=ss.remove(0);

String[]splitoperands=temp.split(",");
                                        for(int
i=0;i<splitoperands.length;i++) {

operand[i]=splitoperands[i];
                                }

comment=ss.remove(0);
                        }
                          else  if(ss.size()==1)
{//comment또는 operand 둘중 하나만 남아있을때
                                        String
temp=ss.remove(0);

if(temp.indexOf(" ")!=-1) {//띄어쓰기 있으면 코멘
트

comment=temp;
                                }
                                else {

String[]splitoperands=temp.split(",");

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

        operand[i]=splitoperands[i];
                                        }
                                }
                                                }
                                        }
                                }
                        }


        /**
         * n,i,x,b,p,e flag를 설정한다.
         *
         *
          * 사용 예 : setFlag(nFlag, 1) 또는
setFlag(TokenTable.nFlag, 1)
         *
         * @param flag  : 원하는 비트 위치
         * @param value : 집어넣고자 하는 값.
1또는 0으로 선언한다.
         */
        public void setFlag(int flag, int value) {
                nixbpe+=(flag*value);
        }

        /**
         * 원하는 flag들의 값을 얻어올 수 있다.
flag의 조합을 통해 동시에 여러개의 플래그를 얻는
것 역시 가능하다.
         *
          * 사용 예 : getFlag(nFlag) 또는
getFlag(nFlag|iFlag)
         *
         * @param flags : 값을 확인하고자 하는
비트 위치
         * @return : 비트위치에 들어가 있는 값.
플래그별로 각각 32, 16, 8, 4, 2, 1의 값을 리턴할
것임.
         */

        public int getFlag(int flags) {
                return nixbpe & flags;
        }
}
```