EMB0000243c3415

**시스템프로그래밍 2020 보고서**

**보고서 제출서약서**

**나는 숭실대학교 소프트웨어학부의 일원으로 명예를 지키면서 생활하고 있습니다.**

**나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.**

**1. 나는 자력으로 보고서를 작성하였습니다.**

**1.1. 나는 동료의 보고서를 베끼지 않았습니다.**

**1.2. 나는 비공식적으로 얻은 해답/해설을 기초로 보고서를 작성하지 않았습니다.**

**2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다. (나는 특히**

**인터넷에서 다운로드한 내용을 보고서에 거의 그대로 복사하여 사용하지 않았습니다.)**

**3. 나는 보고서를 제출하기 전에 동료에게 보여주지 않았습니다.**

**4. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.**

|  |  |
| --- | --- |
| **과목** | **시스템프로그래밍 2020** |
| **과제명** | **프로젝트 1b** |
| **담당교수** | **최 재 영 교 수** |
| **제출인** | ***소프트웨어학부 20150286 최유준* (출석번호 310번)** |
| **제출일** | **2020년 5월 13일** |

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

1. **동기/목적**
2. **설계/구현 아이디어**

**2.1 설계 아이디어**

**2.2 구현 아이디어**

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

**1장 동기/목적**

이번 시스템 프로그래밍의 프로젝트는 한번 작성해 보았던 어셈블리 프로그램을 JAVA에서 다시 완성하는 것이었습니다. 내용은 똑같지만 JAVA에서 클래스와 생성자 등을 이용하여 다른 환경에서 프로젝트를 함으로써 이해도를 높이는 것이 목적입니다. JAVA에서 프로젝트를 진행하면서 C와 다른 점을 비교하고 어셈블리 프로그램에 대한 이해도가 더욱 높아지게 되었습니다. 또한 ArrayList를 이용해서 section을 분리하는 과정을 더욱 정확하게 표현할 수 있습니다.

**2장 설계/구현 아이디어**

**2.1 설계 아이디어**

프로그램의 실행 순서는 다음과 같습니다.

1. assembler 생성자를 통해 inst.data read
2. loadInputFile 함수 호출
3. pass1 함수 호출
   1. putToken 함수 호출하여 Token에 값 입력
   2. address 계산 후 Token.location에 저장
   3. section별 symboltable, literaltable 저장
4. SymbolTable, LiteralTable 출력
5. pass2 함수 호출
   1. 각 section마다 makeObjectCode 함수 호출
   2. 각 토큰 별 ObjectCode를 만든 후 저장
   3. getObjectCode를 통해 codeList에 objectcode 저장
6. ObjectCode 출력

**\*모듈별 설계 아이디어**

**(1) loadInputFile**

input file을 읽어서 line별로 lineList에 저장한다.

**(2) pass1**

TokenList SymbolTable

symbol location

TokenTable1

TokenTable2

TokenTable3..

LiteralTable

literal location

instTable

TokenList에 new TokenTable을 생성하고 그 안에서 TokenTable과 new SymbolTable, new LiteralTable, instTable을 link한다.

1. **putToken**

line을 받아서 new Token에 넣고 그 값을 tokenList에 add한다..

1. **parsing**

line을 받아서 의미를 분석한 후 Token의 각 변수에 저장한다.

1. **printSymbolTable, printLiteralTable**

file에 각 section별 SymbolTable(LiteralTable)들을 symbolList(iteralList)와 locationList의 형태로 출력한다.

1. **pass2**

각 TokenTable별로 ObjectCode를 만들어준 후 codeList에 add한다

1. **makeObjectCode**

index를 입력받아 해당 tokenList의 값을 get하여 objectcode를 만든다

1. **setFlag**

flag와 value를 입력받아 해당 flag의 값을 value로 만든다.

1. **getObjectCode**

index를 입력받아 해당 index의 objectCode를 return한다.

1. **printObjectCode**

출력 형태에 맞게 codeList를 출력한다.

1. **modifySymbol, modifyLiteral**

symbol(literal)값과 변경할 주소값을 입력하여 해당 symbol(literal)의 location을 변경한다.

1. **search**

입력된 symbol(literal)의 location을 return한다.

1. **hash\_get**

operator를 입력하여 Instruction일 경우 그 format을 return한다.

**2.2 구현 아이디어**

1. loadInputFile

file을 open한 후 file의 마지막까지 readLine한 후 lineList에 add한다.

1. pass1

TokenList에 new TokenTable을 add하면서 new SymbolTable, new LiteralTable, instTable을 link한다. section별로 lineList에서 한 줄씩 읽어와 putToken한다. 각 Token의 operator를 분석하여 location 값을 저장한다.

1. parsing

line을 입력받아서 split 함수로 line을 label, operator, operand, comment로 분리한다. 각 요소를 변수에 저장하고 operand는 operand의 개수만큼 Stringp[]에 저장한다.

1. printSymbolTable, printLiteralTable

각 section별로 symbolList(literalList)의 size만큼 반복하여 해당 symtab(literaltab)의 symbolList(literalList)와 locationList를 출력한다.

1. pass2

각 section별로 makeObjectCode를 실행한 후 codeList에 add한다.

1. makeObjectCode

hash\_get함수를 통해 해당 index의 Token이 Instruction인지 판별한다. Instruction일 경우 각 Token의 format을 판별한다. 각 Token의 operand를 분석하여 #의 경우 immediate, @의 경우 indirect 등 setFlag해준다. 또한 같은 section의 symbol인지, 외부참조인지 등을 판별하여 address를 만든 후 opcode, nixbpe와 비트연산을 통해 objectcode를 완성한다.

1. setFlag

처음에 nixbpe를 parsing할 때 0으로 초기화하였다. 이 함수에서 value가 1일 경우 nixbpe에 flag를 더해주고 value가 0일 경우 빼준다.

1. printObjectCode

각 section별로 operator와 operand를 분석하여 H, D, R을 file에 출력한다. 그 후 txtlen 변수와 byteSize를 통해 각 T의 길이를 계산하여 txtlen에 저장한 후 T를 출력한다. 해당 token의 operand가 외부 참조 label을 사용하였을 경우 Modify를 출력한다. 이 과정을 TokenList의 size만큼 반복한다.

1. modifySymbol, modifyLiteral

iterator를 이용하여 hasNext를 조건으로 while문 반복한다. 입력한 symbol(literal)이 symbolList(literalList)와 equal할 경우 location을 newLocation으로 수정한다.

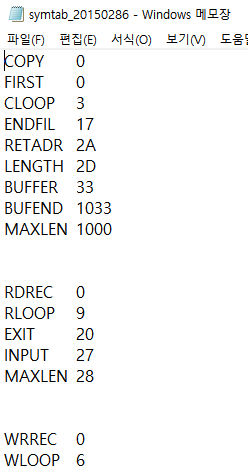
1. search

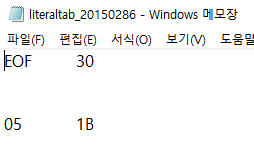
iterator를 이용하여 hasNext를 조건으로 while문 반복한다. 입력한 symbol(literal)이 symbolList(literalList)와 equal할 경우 location을 return한다.

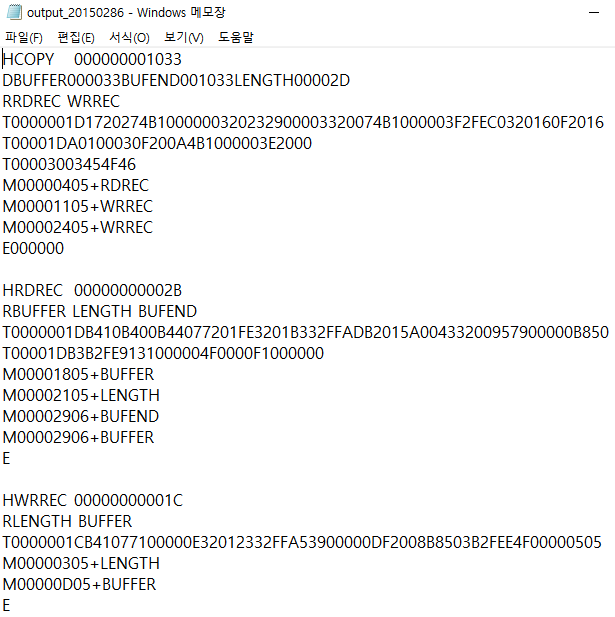
1. hash\_get

line을 입력받으면 해당 line이 4형식일 경우 +를 제외한 string으로 바꾼다. line이 Instruction일 경우 해당 Instruction의 format을 return한다. Instruction이 아니면 -1을 return한다.

**3장 수행결과**





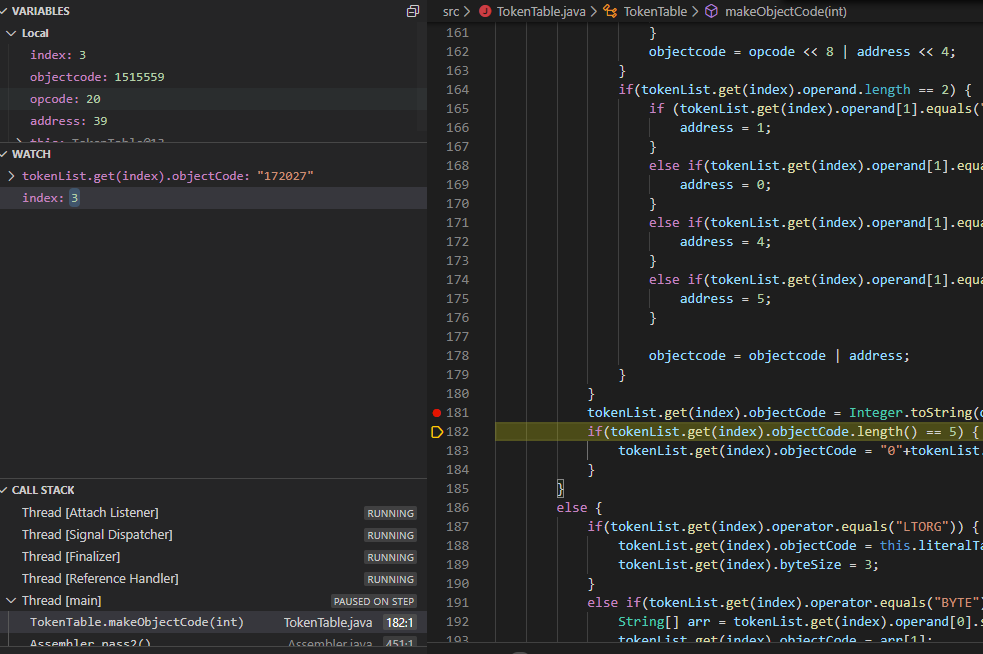


file output

**4장 결론 및 보충할 점**

이미 C로 한번 작성해본 프로젝트이기 때문에 간단할 줄 알았지만 생각만큼 쉬운 과정은 아니었습니다. 자바에 익숙하지 않고 사용해본지 많은 시간이 지났기 때문에 새롭게 공부를 하면서 프로젝트를 진행하였습니다. 그러나 진행해본 결과 C와 비교해서 훨씬 편리하고 어셈블리 프로그램 작성에 유용했습니다. 이번 프로젝트 1b에서는 하드코딩이 아닌 최대한 범용할 수 있게 작성하는 것에 초점을 맞추었지만 아직까지는 완벽하게 하드코딩을 벗어나지 못했습니다. 그러나 C로 작성할 때와 비교하여 발전한 프로그램을 작성할 수 있었습니다.

* **DEBUGING**



eclips로 프로젝트를 진행하였으나 debugging이 되지 않았고 문제를 해결하려 하였으나 어떠한 방법으로도 해결되지 않아 부득이하게 debugging은 vscode로 진행하였습니다.

**5장 소스코드**

**Assembler.java**

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import java.io.FileWriter;

/\*\*

 \* Assembler :

 \* 이 프로그램은 SIC/XE 머신을 위한 Assembler 프로그램의 메인 루틴이다.

 \* 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 내용을 바탕으로 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 출력시키지 말 것. (채점상의 이유. 주석에 포함된 한글은 상관 없음)

 \*

 \*

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

 \*/

public class Assembler {

    /\*\* instruction 명세를 저장한 공간 \*/

    InstTable instTable;

    /\*\* 읽어들인 input 파일의 내용을 한 줄 씩 저장하는 공간. \*/

    ArrayList<String> lineList;

    /\*\* 프로그램의 section별로 symbol table을 저장하는 공간\*/

    ArrayList<SymbolTable> symtabList;

    /\*\* 프로그램의 section별로 literal table을 저장하는 공간\*/

    ArrayList<LiteralTable> 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.

     \*/

    public Assembler(String instFile) {

        instTable = new InstTable(instFile);

        lineList = new ArrayList<String>();

        symtabList = new ArrayList<SymbolTable>();

        literaltabList = new ArrayList<LiteralTable>();

        TokenList = new ArrayList<TokenTable>();

        codeList = new ArrayList<String>();

    }

    /\*\*

     \* 어셈블러의 메인 루틴

     \*/

    public static void main(String[] args) {

        Assembler assembler = new Assembler("inst.data");

        assembler.loadInputFile("input.txt");

        assembler.pass1();

        assembler.printSymbolTable("symtab\_20150286.txt");

        assembler.printLiteralTable("literaltab\_20150286.txt");

        assembler.pass2();

        assembler.printObjectCode("output\_20150286.txt");

    }

    /\*\*

     \* inputFile을 읽어들여서 lineList에 저장한다.

     \* @param inputFile : input 파일 이름.

     \*/

    private void loadInputFile(String inputFile) {

        try {

            File filename = new File(inputFile);

            FileReader fr = new FileReader(filename);

            BufferedReader br = new BufferedReader(fr);

            String line = "";

            while ((line = br.readLine()) != null) {//한줄씩 읽기

                lineList.add(line);

            }

            br.close();

        } catch (FileNotFoundException e) {

            System.out.println(e);

        } catch( IOException e) {

            System.out.println(e);

        }

    }

    /\*\*

     \* pass1 과정을 수행한다.

     \*   1) 프로그램 소스를 스캔하여 토큰단위로 분리한 뒤 토큰테이블 생성

     \*   2) label을 symbolTable에 정리

     \*

     \*    주의사항 : SymbolTable과 TokenTable은 프로그램의 section별로 하나씩 선언되어야 한다.

     \*/

    private void pass1() {

        // TODO Auto-generated method stub

        int i = 0;

        int j;

        int start = 0;

        int addr = 0;

        String lit = null;

        TokenList.add(new TokenTable(new SymbolTable(), new LiteralTable(), instTable));//section 1

        while(i < lineList.size()) {//token parsing

            TokenList.get(0).putToken(lineList.get(i));

            TokenList.get(0).tokenList.get(i).location = addr;

            if(i > 1) {

                if(TokenList.get(0).tokenList.get(i).location - TokenList.get(0).tokenList.get(i-1).location >= 0) {//TokenTable의 length 구하기

                    TokenList.get(0).Listlength = TokenList.get(0).tokenList.get(i).location;

                }

            }

            //location find

            if(instTable.hash\_get(TokenList.get(0).tokenList.get(i).operator) > 0) {//Instruction

                if(instTable.hash\_get(TokenList.get(0).tokenList.get(i).operator) == 3) {

                    if(TokenList.get(0).tokenList.get(i).operator.charAt(0) == '+') {//format 4

                        addr += 4;

                    }

                    else {//format 3

                        addr += 3;

                    }

                }

                else if(instTable.hash\_get(TokenList.get(0).tokenList.get(i).operator) == 2) {

                    TokenList.get(0).tokenList.get(i).location += 2;//format2

                }

            }

            else {//not Instruction

                if(TokenList.get(0).tokenList.get(i).operator.equals("RESW")) {

                    int tmpadr = Integer.parseInt(TokenList.get(0).tokenList.get(i).operand[0]);

                    addr += (3\*tmpadr);

                }

                else if(TokenList.get(0).tokenList.get(i).operator.equals("RESB")) {

                    int tmpadr = Integer.parseInt(TokenList.get(0).tokenList.get(i).operand[0]);

                    addr += tmpadr;

                }

                else if(TokenList.get(0).tokenList.get(i).operator.equals("BYTE")) {

                    addr += 1;

                }

                else if(TokenList.get(0).tokenList.get(i).operator.equals("WORD")) {

                    addr += 3;

                }

                else if(TokenList.get(0).tokenList.get(i).operator.equals("EQU")) {//equation일 경우

                    if(!TokenList.get(0).tokenList.get(i).operand[0].equals("\*")) {

                        String[] arr = TokenList.get(0).tokenList.get(i).operand[0].split("-");//ex) BUFEND-BUFFER

                        int tmpadr1=0, tmpadr2=0;

                        for(int k = 0; k < TokenList.get(0).tokenList.size(); k++) {

                            if(TokenList.get(0).tokenList.get(k).label.equals(arr[0])) {

                                tmpadr1 = TokenList.get(0).tokenList.get(k).location;

                            }

                            if(TokenList.get(0).tokenList.get(k).label.equals(arr[1])) {

                                tmpadr2 = TokenList.get(0).tokenList.get(k).location;

                            }

                        }

                        TokenList.get(0).tokenList.get(i).location = tmpadr1-tmpadr2;

                    }

                }

                else if(TokenList.get(0).tokenList.get(i).operator.equals("LTORG")) {

                    addr += 3;

                }

            }

            if(TokenList.get(0).getToken(i).operator.equals("CSECT")){//다음 section이 현재 Table에 포함되었을 경우 remove해준다.

                TokenList.get(0).tokenList.remove(i);

                break;

            }

            i++;

        }

        for(j = 0; j < i; j++) { //literal, symbol input

            if(!TokenList.get(0).getToken(j).label.equals("")) {//label이 있으면 symboltable에 저장

                TokenList.get(0).symTab.putSymbol(TokenList.get(0).getToken(j).label, TokenList.get(0).getToken(j).location);

            }

            if(TokenList.get(0).getToken(j).operand == null) {//lit에 저장된 값 literaltable에 저장

                if((TokenList.get(0).getToken(j).operator.equals("LTORG")) || (TokenList.get(0).getToken(j).operator.equals("END"))) {

                    TokenList.get(0).literalTab.putLiteral(lit, TokenList.get(0).getToken(j).location);

                }

            }

            else {

                if (TokenList.get(0).getToken(j).operand[0].charAt(0) == '=' ) {//operand가 literal일 경우 lit에 저장

                    String[] arr1 = TokenList.get(0).getToken(j).operand[0].split("'");

                    lit = arr1[1];

                }

            }

        }

        symtabList.add(TokenList.get(0).symTab);//symtabList에 현재 section 저장

        literaltabList.add(TokenList.get(0).literalTab);

        j = i;

        start = i;

        addr = 0;

        i = 0;

        TokenList.add(new TokenTable(new SymbolTable(), new LiteralTable(), instTable));//section 2 이하 반복

        while(j < lineList.size()) {

            if(lineList.get(j).charAt(0) == '.') {

                j++;

                continue;

            }

            TokenList.get(1).putToken(lineList.get(j));

            TokenList.get(1).tokenList.get(i).location = addr;

            if(i > 1) {

                if(TokenList.get(1).tokenList.get(i).location - TokenList.get(1).tokenList.get(i-1).location >= 0) {

                    TokenList.get(1).Listlength = TokenList.get(1).tokenList.get(i).location;

                }

            }

            if(instTable.hash\_get(TokenList.get(1).tokenList.get(i).operator) > 0) {

                if(instTable.hash\_get(TokenList.get(1).tokenList.get(i).operator) == 3) {

                    if(TokenList.get(1).tokenList.get(i).operator.charAt(0) == '+') {

                        addr += 4;

                    }

                    else {

                        addr += 3;

                    }

                }

                else if(instTable.hash\_get(TokenList.get(1).tokenList.get(i).operator) == 2) {

                    addr += 2;

                }

            }

            else {

                if(TokenList.get(1).tokenList.get(i).operator.equals("RESW")) {

                    int tmpadr = Integer.parseInt(TokenList.get(1).tokenList.get(i).operand[0]);

                    addr += (3\*tmpadr);

                }

                else if(TokenList.get(1).tokenList.get(i).operator.equals("RESB")) {

                    int tmpadr = Integer.parseInt(TokenList.get(1).tokenList.get(i).operand[0]);

                    addr += tmpadr;

                }

                else if(TokenList.get(1).tokenList.get(i).operator.equals("BYTE")) {

                    addr += 1;

                }

                else if(TokenList.get(1).tokenList.get(i).operator.equals("WORD")) {

                    addr += 3;

                }

                else if(TokenList.get(1).tokenList.get(i).operator.equals("EQU")) {

                    if(!TokenList.get(1).tokenList.get(i).operand[0].equals("\*")) {

                        String[] arr = TokenList.get(1).tokenList.get(i).operand[0].split("-");

                        int tmpadr1=0, tmpadr2=0;

                        for(int k = 0; k < TokenList.get(1).tokenList.size(); k++) {

                            if(TokenList.get(1).tokenList.get(k).label.equals(arr[0])) {

                                tmpadr1 = TokenList.get(1).tokenList.get(k).location;

                            }

                            if(TokenList.get(1).tokenList.get(k).label.equals(arr[1])) {

                                tmpadr2 = TokenList.get(1).tokenList.get(k).location;

                            }

                        }

                        addr = tmpadr1-tmpadr2;

                    }

                }

                else if(TokenList.get(1).tokenList.get(i).operator.equals("LTORG")) {

                    addr += 3;

                }

            }

            if(TokenList.get(1).getToken(i).label.equals("WRREC")) {

                TokenList.get(1).tokenList.remove(i);

                break;

            }

            j++;

            i++;

        }

        start = j;

        for(j = 0; j < i; j++) {

            if(!TokenList.get(1).getToken(j).label.equals("")) {

                TokenList.get(1).symTab.putSymbol(TokenList.get(1).getToken(j).label, TokenList.get(1).getToken(j).location);

            }

            if(TokenList.get(1).getToken(j).operand == null) {

                if(TokenList.get(1).getToken(j).operator.equals("LTORG") || (TokenList.get(1).getToken(j).operator.equals("END"))) {

                    continue;

                }

            }

            else {

                if(!TokenList.get(1).getToken(j).operator.equals("RSUB")) {

                    if(TokenList.get(1).getToken(j).operand[0].charAt(0) == '=' ) {

                        String[] arr1 = TokenList.get(1).getToken(j).operand[0].split("'");

                        TokenList.get(1).literalTab.putLiteral(arr1[1], TokenList.get(1).getToken(j).location);

                    }

                }

            }

        }

        symtabList.add(TokenList.get(1).symTab);

        literaltabList.add(TokenList.get(1).literalTab);

        j = start;

        addr = 0;

        i = 0;

        TokenList.add(new TokenTable(new SymbolTable(), new LiteralTable(), instTable));//section3 이하 반복

        while(j < lineList.size()) {

            if(lineList.get(j).charAt(0) == '.') {

                j++;

                continue;

            }

            TokenList.get(2).putToken(lineList.get(j));

            TokenList.get(2).tokenList.get(i).location = addr;

            if(i > 1) {

                if(TokenList.get(2).tokenList.get(i).location - TokenList.get(2).tokenList.get(i-1).location >= 0) {

                    TokenList.get(2).Listlength = TokenList.get(2).tokenList.get(i).location;

                }

            }

            if(instTable.hash\_get(TokenList.get(2).tokenList.get(i).operator) > 0) {

                if(instTable.hash\_get(TokenList.get(2).tokenList.get(i).operator) == 3) {

                    if(TokenList.get(2).tokenList.get(i).operator.charAt(0) == '+') {

                        addr += 4;

                    }

                    else {

                        addr += 3;

                    }

                }

                else if(instTable.hash\_get(TokenList.get(2).tokenList.get(i).operator) == 2) {

                    addr += 2;

                }

            }

            else {

                if(TokenList.get(2).tokenList.get(i).operator.equals("RESW")) {

                    int tmpadr = Integer.parseInt(TokenList.get(2).tokenList.get(i).operand[0]);

                    addr += (3\*tmpadr);

                }

                else if(TokenList.get(2).tokenList.get(i).operator.equals("RESB")) {

                    int tmpadr = Integer.parseInt(TokenList.get(2).tokenList.get(i).operand[0]);

                    addr += tmpadr;

                }

                else if(TokenList.get(2).tokenList.get(i).operator.equals("BYTE")) {

                    addr += 1;

                }

                else if(TokenList.get(2).tokenList.get(i).operator.equals("WORD")) {

                    addr += 3;

                }

                else if(TokenList.get(2).tokenList.get(i).operator.equals("EQU")) {

                    if(!TokenList.get(2).tokenList.get(i).operand[0].equals("\*")) {

                        String[] arr = TokenList.get(2).tokenList.get(i).operand[0].split("-");

                        int tmpadr1=0, tmpadr2=0;

                        for(int k = 0; k < TokenList.get(2).tokenList.size(); k++) {

                            if(TokenList.get(2).tokenList.get(k).label.equals(arr[0])) {

                                tmpadr1 = TokenList.get(2).tokenList.get(k).location;

                            }

                            if(TokenList.get(2).tokenList.get(k).label.equals(arr[1])) {

                                tmpadr2 = TokenList.get(2).tokenList.get(k).location;

                            }

                        }

                        addr = tmpadr1-tmpadr2;

                    }

                }

                else if(TokenList.get(2).tokenList.get(i).operator.equals("LTORG")) {

                    addr += 3;

                }

            }

            if(TokenList.get(2).getToken(i).operator.equals("END")) {

                TokenList.get(2).Listlength += 1;

                break;

            }

            j++;

            i++;

        }

        for(j = 0; j <i+1; j++) {

            if(!TokenList.get(2).getToken(j).label.equals("")) {

                TokenList.get(2).symTab.putSymbol(TokenList.get(2).getToken(j).label, TokenList.get(2).getToken(j).location);

            }

            if(TokenList.get(2).getToken(j).operand == null) {

                if(TokenList.get(2).getToken(j).operator.equals("LTORG")) {

                    TokenList.get(2).literalTab.putLiteral(lit, TokenList.get(2).getToken(j).location);

                }

            }

            else {

                if(!TokenList.get(2).getToken(j).operator.equals("RSUB")) {

                    if (TokenList.get(2).getToken(j).operand[0].charAt(0) == '=' ) {

                        String[] arr1 = TokenList.get(2).getToken(j).operand[0].split("'");

                        lit = arr1[1];

                    }

                }

                if(TokenList.get(2).getToken(j).operator.equals("END")) {

                    TokenList.get(2).literalTab.putLiteral(lit, TokenList.get(2).getToken(j).location);

                }

            }

        }

        symtabList.add(TokenList.get(2).symTab);

        literaltabList.add(TokenList.get(2).literalTab);

    }

    /\*\*

     \* 작성된 SymbolTable들을 출력형태에 맞게 출력한다.

     \* @param fileName : 저장되는 파일 이름

     \*/

    private void printSymbolTable(String fileName) {

        File filename = new File(fileName);

        FileWriter fw = null;

        try{

            fw = new FileWriter(filename, true);

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

                fw.write(TokenList.get(0).symTab.symbolList.get(i) +"\t"+ Integer.toHexString(TokenList.get(0).symTab.locationList.get(i)).toUpperCase()+"\n");//16진수 대문자로 출력 toHexString, toUpperCase 이용

            }

            fw.write("\n\n");

            for(int i = 0; i < symtabList.get(1).symbolList.size(); i++) {

                fw.write(TokenList.get(1).symTab.symbolList.get(i) +"\t"+ Integer.toHexString(TokenList.get(1).symTab.locationList.get(i)).toUpperCase()+"\n");

            }

            fw.write("\n\n");

            for(int i = 0; i < symtabList.get(2).symbolList.size(); i++) {

                fw.write(TokenList.get(2).symTab.symbolList.get(i) +"\t"+ Integer.toHexString(TokenList.get(2).symTab.locationList.get(i)).toUpperCase()+"\n");

            }

            fw.close();

        } catch (FileNotFoundException e) {

            System.out.println(e);

        } catch( IOException e) {

            System.out.println(e);

        }

        //TokenList.get(0).symTab.symbolList(i)

    }

    /\*\*

     \* 작성된 LiteralTable들을 출력형태에 맞게 출력한다.

     \* @param fileName : 저장되는 파일 이름

     \*/

    private void printLiteralTable(String fileName) {

        // TODO Auto-generated method stub

        File filename = new File(fileName);

        FileWriter fw = null;

        try{

            fw = new FileWriter(filename, true);

            for(int i = 0; i < literaltabList.get(0).literalList.size(); i++){

                fw.write(TokenList.get(0).literalTab.literalList.get(i) +"\t"+ Integer.toHexString(TokenList.get(0).literalTab.locationList.get(i)).toUpperCase()+"\n");

            }

            fw.write("\n\n");

            for(int i = 0; i < literaltabList.get(2).literalList.size(); i++) {

                fw.write(TokenList.get(2).literalTab.literalList.get(i) +"\t"+ Integer.toHexString(TokenList.get(2).literalTab.locationList.get(i)).toUpperCase()+"\n");

            }

            fw.close();

        } catch (FileNotFoundException e) {

            System.out.println(e);

        } catch( IOException e) {

            System.out.println(e);

        }

    }

    /\*\*

     \* pass2 과정을 수행한다.

     \*   1) 분석된 내용을 바탕으로 object code를 생성하여 codeList에 저장.

     \*/

    private void pass2() {

        // TODO Auto-generated method stub

        for(int i = 0; i < TokenList.get(0).tokenList.size(); i++) {//section 1

            TokenList.get(0).makeObjectCode(i);

            codeList.add(TokenList.get(0).getObjectCode(i));

        }

        for(int i = 0; i < TokenList.get(1).tokenList.size(); i++) {//section2

            TokenList.get(1).makeObjectCode(i);

            codeList.add(TokenList.get(1).getObjectCode(i));

        }

        for(int i = 0; i < TokenList.get(2).tokenList.size(); i++) {

            TokenList.get(2).makeObjectCode(i);//section3

            codeList.add(TokenList.get(2).getObjectCode(i));

        }

    }

    /\*\*

     \* 작성된 codeList를 출력형태에 맞게 출력한다.

     \* @param fileName : 저장되는 파일 이름

     \*/

    private void printObjectCode(String fileName) {

        // TODO Auto-generated method stub

        File filename = new File(fileName);

        FileWriter fw = null;

        int i;

        int start = 0;

        try{

            fw = new FileWriter(filename, true);

            for(int k = 0; k < TokenList.size(); k ++) {

                for(i = 0; i < TokenList.get(k).tokenList.size(); i++){

                    if(codeList.get(start+i) == null) {

                        if(TokenList.get(k).tokenList.get(i).operator.equals("START") || TokenList.get(k).tokenList.get(i).operator.equals("CSECT")) {

                            String str = "H"+TokenList.get(k).tokenList.get(i).label+"\t000000";

                            fw.write(str+""+String.format("%06X", TokenList.get(k).Listlength, 16));

                        }

                        else if(TokenList.get(k).tokenList.get(i).operator.equals("EXTDEF")) {

                            fw.write("D");

                            for(int j = 0; j < TokenList.get(k).tokenList.get(i).operand.length; j ++) {

                                String str = TokenList.get(k).tokenList.get(i).operand[j];

                                fw.write(str+String.format("%06X", TokenList.get(k).symTab.search(str)));

                            }

                        }

                        else if(TokenList.get(k).tokenList.get(i).operator.equals("EXTREF")) {

                            fw.write("R");

                            for(int j = 0; j < TokenList.get(k).tokenList.get(i).operand.length; j ++) {

                                fw.write(TokenList.get(k).tokenList.get(i).operand[j]+" ");

                            }

                        }

                        fw.write("\n");

                    }

                    else {

                        break;

                    }

                }

                int startadr = 0;

                int txtlen = 0;

                int j = i;

                int txt = 0;

                while(txtlen <= 30) {

                    if(txtlen + TokenList.get(k).tokenList.get(j).byteSize <= 30){

                        if(codeList.get(start+j).equals("05")) {

                            txtlen += 1;

                            break;

                        }

                        txtlen += TokenList.get(k).tokenList.get(j).byteSize;

                    }

                    else {

                        break;

                    }

                    j++;

                }

                fw.write("T"+String.format("%06X%X", startadr, txtlen));

                for(; i < TokenList.get(k).tokenList.size(); i++) {

                    fw.write(codeList.get(start+i));

                    txt += TokenList.get(k).tokenList.get(i).byteSize;

                    if(txt == txtlen) {

                        break;

                    }

                }

                i = j;

                startadr = txtlen;

                txtlen = 0;

                txt = 0;

                while(txtlen <= 30) {

                    if(k == 2) {

                        break;

                    }

                    if(txtlen + TokenList.get(k).tokenList.get(j).byteSize <= 30){

                        if(codeList.get(start+j+1) == null) {

                            break;

                        }

                        txtlen += TokenList.get(k).tokenList.get(j).byteSize;

                    }

                    else {

                        break;

                    }

                    j++;

                }

                if(k != 2){

                    fw.write("\nT"+String.format("%06X%X", startadr, txtlen));

                }

                for(; i < TokenList.get(k).tokenList.size(); i++) {

                    fw.write(codeList.get(start+i));

                    txt += TokenList.get(k).tokenList.get(i).byteSize;

                    if(i == j) {

                        break;

                    }

                }

                for(; i < TokenList.get(k).tokenList.size(); i++) {

                    if(TokenList.get(k).tokenList.get(i).operator.equals("LTORG")) {

                        char c1, c2, c3;

                        int i1, i2, i3;

                        c1 = TokenList.get(k).literalTab.literalList.get(0).charAt(0);

                        c2 = TokenList.get(k).literalTab.literalList.get(0).charAt(1);

                        c3 = TokenList.get(k).literalTab.literalList.get(0).charAt(2);

                        i1 = (int)c1; i2 = (int)c2; i3 = (int)c3;

                        fw.write("\nT"+String.format("%06X", TokenList.get(k).literalTab.locationList.get(0))+String.format("%02X", TokenList.get(k).tokenList.get(i).byteSize)+String.format("%X%X%X", i1, i2, i3));

                    }

                }

                for(int cnt = 3; cnt < TokenList.get(k).tokenList.size(); cnt++) {

                    if(TokenList.get(k).tokenList.get(cnt).operand == null) {

                        cnt++;

                        continue;

                    }

                    if(k == 0) {

                        if(TokenList.get(k).tokenList.get(cnt).operand[0].contains("RDREC") || TokenList.get(k).tokenList.get(cnt).operand[0].contains("WRREC")) {

                            fw.write("\nM"+String.format("%06X", TokenList.get(k).tokenList.get(cnt).location+1)+"05"+"+"+TokenList.get(k).tokenList.get(cnt).operand[0]);

                        }

                    }

                    else if (k == 1) {

                        if(TokenList.get(k).tokenList.get(cnt).operand[0].contains("BUFEND") || TokenList.get(k).tokenList.get(cnt).operand[0].contains("LENGTH") || TokenList.get(k).tokenList.get(cnt).operand[0].contains("BUFFER")) {

                            if(TokenList.get(k).tokenList.get(cnt).operand[0].equals("BUFEND-BUFFER")) {

                                fw.write("\nM"+String.format("%06X", TokenList.get(k).tokenList.get(cnt).location+1)+"06"+"+BUFEND");

                                fw.write("\nM"+String.format("%06X", TokenList.get(k).tokenList.get(cnt).location+1)+"06"+"+BUFFER");

                            }

                            else {

                                fw.write("\nM"+String.format("%06X", TokenList.get(k).tokenList.get(cnt).location+1)+"05"+"+"+TokenList.get(k).tokenList.get(cnt).operand[0]);

                            }

                        }

                    }

                    else if(k == 2) {

                        if(TokenList.get(k).tokenList.get(cnt).operand[0].contains("BUFFER") || TokenList.get(k).tokenList.get(cnt).operand[0].contains("LENGTH")) {

                            fw.write("\nM"+String.format("%06X", TokenList.get(k).tokenList.get(cnt).location+1)+"05"+"+"+TokenList.get(k).tokenList.get(cnt).operand[0]);

                        }

                    }

                }

                fw.write("\nE");

                if(k == 0) {

                    fw.write("000000");

                }

                start += i;

                fw.write("\n\n");

            }

        fw.close();

        } catch (FileNotFoundException e) {

            System.out.println(e);

        } catch( IOException e) {

            System.out.println(e);

        }

    }

}

**InstTable.java**

import java.util.HashMap;

import java.io.File;

import java.io.FileReader;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.FileNotFoundException;

/\*\*

 \* 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

     \*/

    public InstTable(String instFile) {

        instMap = new HashMap<String, Instruction>();

        openFile(instFile);

    }

    /\*\*

     \* 입력받은 이름의 파일을 열고 해당 내용을 파싱하여 instMap에 저장한다.

     \*/

    public void openFile(String fileName) {

        File filename = new File(fileName);

        FileReader fr = null;

        try {

            fr = new FileReader(filename);

            BufferedReader br = new BufferedReader(fr);

            String line = "";

            while ((line = br.readLine()) != null) {

                String[] arr = line.split("\t");//tab으로 구분하여 앞은 Instruction(key) 뒤는 format,opcode,number of operand

                Instruction inst = new Instruction(arr[1]);

                instMap.put(arr[0], inst);

            }

            br.close();

        } catch (FileNotFoundException e) {

            System.out.println(e);

        } catch( IOException e) {

            System.out.println(e);

        }

    }

    public int hash\_get(String line) {

        if(line.charAt(0) == '+') {//4형식일 경우

            String newline = line.substring(1);//+를 떼준다

            return instMap.get(newline).format;

        }

        else {

            if(instMap.containsKey(line) == true) {

                return (instMap.get(line).format);

            }

            else {

                return -1;

            }

        }

    }

}

/\*\*

 \* 명령어 하나하나의 구체적인 정보는 Instruction클래스에 담긴다.

 \* instruction과 관련된 정보들을 저장하고 기초적인 연산을 수행한다.

 \*/

class Instruction {

     int format;

     String opcode;

     int numberOfOperand;

    /\*\* instruction이 몇 바이트 명령어인지 저장. 이후 편의성을 위함 \*/

    /\*\*

     \* 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) {

        //Instruction inst = new Instruction(line);

        String[] arr = line.split(",");//format,opcode,numberOfOperand 분리

        this.format = Integer.parseInt(arr[0]);

        this.opcode = arr[1];

        this.numberOfOperand = Integer.parseInt(arr[2]);

    }

}

**LiteralTable.java**

import java.util.ArrayList;

import java.util.Iterator;

/\*\*

 \* literal과 관련된 데이터와 연산을 소유한다.

 \* section 별로 하나씩 인스턴스를 할당한다.

 \*/

public class LiteralTable {

    ArrayList<String> literalList;

    ArrayList<Integer> locationList;

    // 기타 literal, external 선언 및 처리방법을 구현한다.

    /\*\*

     \* 새로운 Literal을 table에 추가한다.

     \* @param literal : 새로 추가되는 literal의 label

     \* @param location : 해당 literal이 가지는 주소값

     \* 주의 : 만약 중복된 literal이 putLiteral을 통해서 입력된다면 이는 프로그램 코드에 문제가 있음을 나타낸다.

     \* 매칭되는 주소값의 변경은 modifyLiteral()을 통해서 이루어져야 한다.

     \*/

    public void putLiteral(String literal, int location) {

        literalList = new ArrayList<>();

        locationList = new ArrayList<>();

        literalList.add(literal);

        locationList.add(location);

    }

    /\*\*

     \* 기존에 존재하는 literal 값에 대해서 가리키는 주소값을 변경한다.

     \* @param literal : 변경을 원하는 literal의 label

     \* @param newLocation : 새로 바꾸고자 하는 주소값

     \*/

    public void modifyLiteral(String literal, int newLocation) {

        Iterator<String> it = literalList.iterator();//반복자 선언

        int i = 0;

        while(it.hasNext()) {

            if (literal.equals(literalList.get(i)))

                break;

            i++;

        }

        locationList.set(i, newLocation);

    }

    /\*\*

     \* 인자로 전달된 literal이 어떤 주소를 지칭하는지 알려준다.

     \* @param literal : 검색을 원하는 literal의 label

     \* @return literal이 가지고 있는 주소값. 해당 literal이 없을 경우 -1 리턴

     \*/

    public int search(String literal) {

        int address = 0;

        Iterator<String> it = literalList.iterator();

        int i = 0;

        while(it.hasNext()) {

            if (literal.equals(literalList.get(i))) {

                address = locationList.get(i);

                return address;

            }

            i++;

        }

        return -1;

    }

}

**SymbolTable.java**

import java.util.ArrayList;

import java.util.Iterator;

/\*\*

 \* symbol과 관련된 데이터와 연산을 소유한다.

 \* section 별로 하나씩 인스턴스를 할당한다.

 \*/

public class SymbolTable {

    ArrayList<String> symbolList = new ArrayList<String>();

    ArrayList<Integer> locationList = new ArrayList<Integer>();

    // 기타 literal, external 선언 및 처리방법을 구현한다.

    /\*\*

     \* 새로운 Symbol을 table에 추가한다.

     \* @param symbol : 새로 추가되는 symbol의 label

     \* @param location : 해당 symbol이 가지는 주소값

     \* 주의 : 만약 중복된 symbol이 putSymbol을 통해서 입력된다면 이는 프로그램 코드에 문제가 있음을 나타낸다.

     \* 매칭되는 주소값의 변경은 modifySymbol()을 통해서 이루어져야 한다.

     \*/

    public void putSymbol(String symbol, int location) {

        symbolList.add(symbol);

        locationList.add(location);

    }

    /\*\*

     \* 기존에 존재하는 symbol 값에 대해서 가리키는 주소값을 변경한다.

     \* @param symbol : 변경을 원하는 symbol의 label

     \* @param newLocation : 새로 바꾸고자 하는 주소값

     \*/

    public void modifySymbol(String symbol, int newLocation) {

        Iterator<String> it = symbolList.iterator();

        int i = 0;

        while(it.hasNext()) {

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

                break;

            i++;

        }

        locationList.set(i, newLocation);

    }

    /\*\*

     \* 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 = 0;

        Iterator<String> it = symbolList.iterator();

        int i = 0;

        while(it.hasNext()) {

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

                address = locationList.get(i);

                return address;

            }

            i++;

        }

        return -1;

    }

}

**TokenTable.java**

import java.util.ArrayList;

import java.util.HashMap;

/\*\*

 \* 사용자가 작성한 프로그램 코드를 단어별로 분할 한 후, 의미를 분석하고, 최종 코드로 변환하는 과정을 총괄하는 클래스이다. <br>

 \* pass2에서 object code로 변환하는 과정은 혼자 해결할 수 없고 symbolTable과 instTable의 정보가 필요하므로 이를 링크시킨다.<br>

 \* 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. \*/

    SymbolTable symTab;

    LiteralTable literalTab;

    InstTable instTab;

    int Listlength;

    /\*\* 각 line을 의미별로 분할하고 분석하는 공간. \*/

    ArrayList<Token> tokenList;

    /\*\*

     \* 초기화하면서 symTable과 instTable을 링크시킨다.

     \* @param symTab : 해당 section과 연결되어있는 symbol table

     \* @param instTab : instruction 명세가 정의된 instTable

     \*/

    public TokenTable(SymbolTable symTab, LiteralTable literalTab, InstTable instTab) {

        this.symTab = symTab;

        this.literalTab = literalTab;

        this.instTab = instTab;

        this.tokenList = new ArrayList<Token>();

    }

    /\*\*

     \* 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 literal table 등을 참조하여 objectcode를 생성하고, 이를 저장한다.

     \* @param index

     \*/

    public void makeObjectCode(int index){

        //...

        int objectcode = 0;//임시 int형 objectcode

        int opcode = 0;//임시 int형 opcode

        int address = 0;

        if(instTab.hash\_get(tokenList.get(index).operator) > 0) {//instruction이면

            if(tokenList.get(index).operator.charAt(0) == '+') {

                String newline;

                newline = tokenList.get(index).operator.substring(1);

                opcode = Integer.parseInt(instTab.instMap.get(newline).opcode, 16);

            }

            else {

                opcode = Integer.parseInt(instTab.instMap.get(tokenList.get(index).operator).opcode, 16);

            }

            if(instTab.hash\_get(tokenList.get(index).operator) == 3) {//format 3,4

                tokenList.get(index).setFlag(nFlag, 1);//기본적으로 3형식으로 간주, byteSize=3, n=1, i=1, PC relative

                tokenList.get(index).setFlag(iFlag, 1);

                tokenList.get(index).setFlag(pFlag, 1);

                tokenList.get(index).byteSize = 3;

                if(tokenList.get(index).operator.charAt(0) == '+') {//format4 p, e는 0, byteSize = 4

                    tokenList.get(index).setFlag(pFlag, 0);

                    tokenList.get(index).setFlag(eFlag, 1);

                    tokenList.get(index).byteSize = 4;

                }

                if(tokenList.get(index).operator.equals("RSUB")) {//RSUB은 예외처리

                    tokenList.get(index).setFlag(pFlag, 0);

                    objectcode = opcode << 16 | tokenList.get(index).nixbpe << 12;

                    tokenList.get(index).objectCode = Integer.toString(objectcode, 16).toUpperCase();

                    return;

                }

                if(tokenList.get(index).operand[0].charAt(0) == '#') {//immediate

                    tokenList.get(index).setFlag(pFlag, 0);

                    tokenList.get(index).setFlag(nFlag, 0);

                }

                else if(tokenList.get(index).operand[0].charAt(0) == '@') {//indirect

                    tokenList.get(index).setFlag(iFlag, 0);

                }

                if(tokenList.get(index).operand.length == 2 && tokenList.get(index).operand[1].equals("X")) {//index

                    tokenList.get(index).setFlag(xFlag, 1);

                }

                //find address

                String str;

                if(tokenList.get(index).operand[0].charAt(0) == '#' || tokenList.get(index).operand[0].charAt(0) == '@') {//immediate or indirect의 경우 기호 빼고 operand 저장

                    str = tokenList.get(index).operand[0].substring(1);

                }

                else {

                    str = tokenList.get(index).operand[0];

                }

                int i;

                int check = -1;

                for(i = 0; i < this.symTab.symbolList.size(); i++) {

                    if(str.equals(this.symTab.symbolList.get(i))) {//같은 section symbol이면 check >= 0

                        check = i;

                        break;

                    }

                }

                if(check != -1){//같은 section symbol

                    address = this.symTab.locationList.get(i) - tokenList.get(index+1).location;

                    if(address < 0) {

                        address = 4096 + (this.symTab.locationList.get(i) - tokenList.get(index+1).location);

                    }

                }

                else {//같은 section symbol이 아님

                    if(str.charAt(0) >= '0' && str.charAt(0) <= '9') {//immediate #number

                        address = Integer.parseInt(str);

                    }

                    else {

                        if(tokenList.get(index).operand[0].charAt(0) == '=') {//literal

                            address = this.literalTab.locationList.get(0) - tokenList.get(index+1).location;

                        }

                        else {//another section symbol

                            address = 0;

                        }

                    }

                }

                if(tokenList.get(index).byteSize == 3) {//3형식의 경우 비트연산

                    objectcode = opcode << 16 | tokenList.get(index).nixbpe << 12 | address;

                }

                else if(tokenList.get(index).byteSize == 4) {//format4 비트연산

                    objectcode = opcode << 24 | tokenList.get(index).nixbpe << 20 | address;

                }

            }

            else {//2형식

                tokenList.get(index).byteSize = 2;

                if(tokenList.get(index).operand.length >= 1){

                    if(tokenList.get(index).operand[0].equals("X")) {//operand[0]일 경우...->비트연산으로 4비트 밀어준다.

                        address = 1;

                    }

                    else if(tokenList.get(index).operand[0].equals("A")) {

                        address = 0;

                    }

                    else if(tokenList.get(index).operand[0].equals("S")) {

                        address = 4;

                    }

                    else if(tokenList.get(index).operand[0].equals("T")) {

                        address = 5;

                    }

                    objectcode = opcode << 8 | address << 4;

                }

                if(tokenList.get(index).operand.length == 2) {

                    if (tokenList.get(index).operand[1].equals("X")) {//operand[1]의 경우...->바로 저장

                        address = 1;

                    }

                    else if(tokenList.get(index).operand[1].equals("A")) {

                        address = 0;

                    }

                    else if(tokenList.get(index).operand[1].equals("S")) {

                        address = 4;

                    }

                    else if(tokenList.get(index).operand[1].equals("T")) {

                        address = 5;

                    }

                    objectcode = objectcode | address;

                }

            }

            tokenList.get(index).objectCode = Integer.toString(objectcode, 16).toUpperCase();

            if(tokenList.get(index).objectCode.length() == 5) {//제일 앞자리가 0이어서 생략된 경우 0 추가

                tokenList.get(index).objectCode = "0"+tokenList.get(index).objectCode;

            }

        }

        else {//not Instruction

            if(tokenList.get(index).operator.equals("LTORG")) {

                tokenList.get(index).objectCode = this.literalTab.literalList.get(0);

                tokenList.get(index).byteSize = 3;

            }

            else if(tokenList.get(index).operator.equals("BYTE")){

                String[] arr = tokenList.get(index).operand[0].split("'");

                tokenList.get(index).objectCode = arr[1];

                tokenList.get(index).byteSize = 1;

            }

            else if(tokenList.get(index).operator.equals("WORD")) {

                tokenList.get(index).objectCode = "000000";

                tokenList.get(index).byteSize = 3;

            }

            else if(tokenList.get(index).operator.equals("END")) {

                tokenList.get(index).objectCode = this.literalTab.literalList.get(0);

                tokenList.get(index).byteSize = 1;

            }

        }

    }

    /\*\*

     \* index번호에 해당하는 object code를 리턴한다.

     \* @param index

     \* @return : object code

     \*/

    public String getObjectCode(int index) {

        return 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) {

        //initialize 추가

        this.nixbpe = 0;

        this.location = 0;

        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[] arr = line.split("\t", 4);

        this.label = arr[0];

        this.operator = arr[1];

        if(arr.length > 3)

            this.comment = arr[3];

        if(arr.length > 2) {

            String op[] = arr[2].split(",", 3);

            this.operand = new String[op.length];

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

                this.operand[i] = op[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) {

        if(value == 1) {

            this.nixbpe += (char)flag;

        }

        else if(value == 0) {

            this.nixbpe -= (char)flag;

        }

    }

    /\*\*

     \* 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;

    }

}