НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
Кафедра системного програмування і спеціалізованих комп'ютерних систем

ПОЯСНЮВАЛЬНА ЗАПИСКА

з дисципліни «Системне програмування»
на тему «Розробка компілятора програм мовою Асемблера»

Студента _____ курсу _____ групи
напряму підготовки
6.050102 «Комп'ютерна інженерія»

_____
(прізвище та ініціали)

Керівник _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка _____

Кількість балів: _____ Оцінка: ECTS _____

Члени комісії _____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

КИЇВ – 2014

ТЕХНІЧНЕ ЗАВДАННЯ

1. Вхідні дані транслятора - текстовий файл з довільною програмою на мові Асемблера, складеною в відповідності з обмеженнями, які задані в варіанті курсової роботи. Для підготовки програми на мові Асемблера використовується, наприклад, стандартний додаток OS Windows Блокнот.
2. На всі синтаксичні конструкції (ідентифікатори, константи, директиви, машинні команди, режими адресації і т.д.), які допускаються в TASM(MASM) і які виходять за рамки обмежень в варіанті курсової роботи повинно видаватись діагностичне повідомлення про синтаксичну помилку.
3. В результаті роботи транслятора повинен бути створений текстовий файл лістінга (розширення .lst). Формат файлу лістінга повинен співпадати з форматом файлу лістінга MASM або TASM. Діагностичні повідомлення формуються на українській мові. Таблиця символів в файлі лістінга може бути в довільному форматі.
4. Транслятор повинен аналізувати командний рядок, в якому задаються імена початкового файлу та файлу лістінга. Всі діагностичні повідомлення, які формуються в файлі лістінга додатково повинні виводитись на екран монітора. Крім того, на екран виводиться загальна кількість помилок, виявлених в початковій програмі.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

Варіант 1.

Ідентифікатори

Містять великі і малі букви латинского алфавіту та цифри. Починаються з букви. Великі та малі букви не відрізняються. Довжина ідентифікаторів не більше 8 символів.

Константи

Шістнадцятерічні, десяткові, двійкові та текстові константи

Директиви

END, SEGMENT - без операндів, ENDS, ASSUME

DB,DW,DD з одним операндом - константою (строкові константи тільки для DB)

Розрядність даних та адрес

16 - розрядні дані та зміщення в сегменті, в випадку 32-розрядних даних та 32 - розрядних зміщень генеруються відповідні префікси зміни розрядності

Адресація операндів пам'яті

Індексна адресація (Val1[si],Val1[bx],Val1[eax],Val1[edi] і т.п.)

Заміна сегментів

Префікси заміни сегментів можуть задаватись явно, а при необхідності автоматично генеруються транслятором

Машинні команди

Cli
Inc **mem**
Dec **reg**
Add **mem,imm**
Cmp **reg,mem**
Xor **mem,reg**
Mov **reg,imm**
Or **reg,reg**
Jb
jmp (внутрішньосегментна відносна адресація)

Де **reg –** 8,16 або 32-розрядні РЗП
mem – адреса операнда в пам’яті
imm - 8,16 або 32-розрядні безпосередні дані (константи)

КАЛЕНДАРНИЙ ПЛАН-ГРАФІК ВИКОНАННЯ КР

№ пп	Етапи виконання	Дата
1	Створення на базі варіанту завдання тестових програм мовою Асемблера та узгодження їх з викладачем.	
2	Розробка лексичного аналізатора.	
3	Розробка програми 1-го перегляду.	
4	Розробка програми 2-го перегляду.	
5	Оформлення результатів курсової роботи.	
6	Захист курсової роботи.	

ОСНОВНІ ПРИНЦИПИ РОБОТИ РЕАЛІЗОВАНОГО ТРАНСЛЯТОРА

Вихідний файл зчитується у масив стрічок після чого кожна стрічка перетворюється у клас який містить інформацію про тип і розмір(у байтах) стрічки, прапорець правильності стрічки, машинний код. Для формування цієї інформації стрічка проходить через 3 етапи аналізу: лексичний, синтаксичний, граматичний.

На етапі лексичного аналізу стрічка розбивається на масив лексем. Кожна лексема – клас який містить два поля: лексему і тип лексеми. Отриманий масив лексем поступає на наступний етап – синтаксичний аналіз.

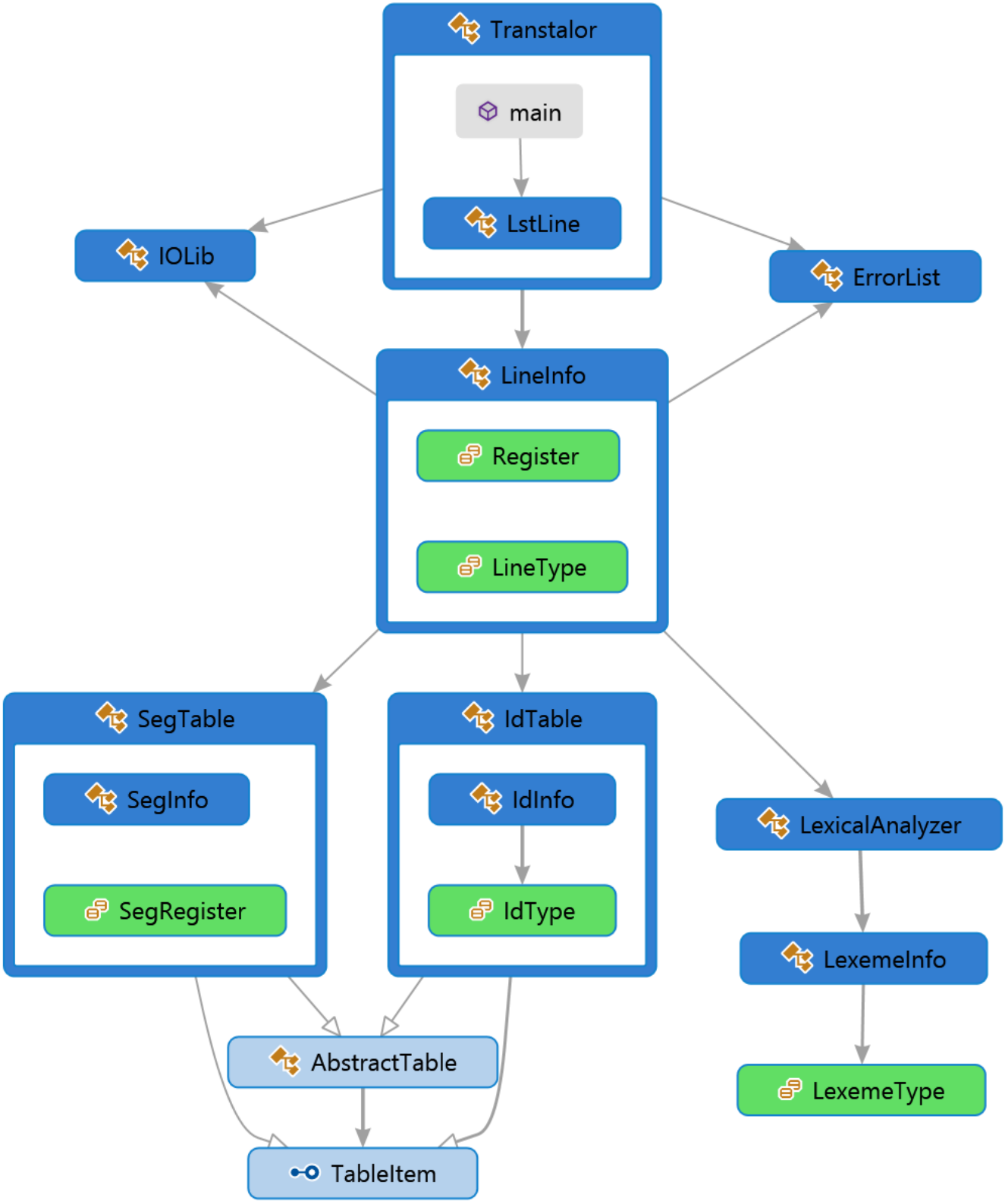
На цьому етапі будується шаблон вхідної стрічки для подальшого визначення типу стрічки. Також на цьому етапі можна відкинути всі недопустимі синтаксичні конструкції, якщо створеного шаблону стрічки немає у масиві допустимих зразків, то стрічка вважається помилковою.

Наступний етап - граматичний аналіз і формування машинного коду. На даному етапі проводиться аналіз на сумісність типів операндів, існування ідентифікаторів, заповнюються таблиці ідентифікаторів і сегментів, формується машинний код команди. Якщо в стрічці були помилки машинний код не генерується і стрічка додається до списку помилкових стрічок. Стрічки з командами переходу(JMP/JB) додаються у спеціальний список для другого проходу.

Після того як перший прохід по файлу закінчено виконується додатковий прохід по списку стрічок що потребують повторного перегляду. На останньому етапі масив класів виводиться у файл лістингу з відповідним форматуванням. Також у файл лістингу виводиться таблиця сегментів і ідентифікаторів.

На наступній сторінці зображена поверхова схема залежностей класів. Всі етапи аналізу стрічки виконуються у класі LineInfo, клас LexicalAnalyzer реалізований окремо тому що так потребувало ТЗ курсового проекту.

СХЕМА ЗАЛЕЖНОСТЕЙ



ОПИС РОЗРОБЛЕНИХ МОДУЛІВ ТА ПІДПРОГРАМ

Транслятор був розроблений на мові програмування Java, тому всі модулі реалізовані у вигляді окремих класів. Усі типи та методи супроводжуються короткими коментарями - Javadoc. Список усіх класів:

Клас	Короткий опис
LexemeType	*Перелік.* Типи лексем
LexemeInfo	Контейнер для збереження інформації про одну лексему
LexicalAnalyzer	Лексичний аналізатор
LineInfo	Контейнер для збереження інформації про вихідну стрічку мовою асемблера
LineInfo.LineType	*Перелік.* Типи вихідних стрічок мовою асемблера
LineInfo.Register	*Перелік.* Регістри загального призначення
TableItem	*Інтерфейс.* Елемент таблиці
AbstractTable	*Абстрактний клас.* Спільна «база» для таблиць
IdTable	Таблиця ідентифікаторів. Розширює *AbstractTable*
IdTable.IdType	*Перелік.* Тип елементу таблиці ідентифікаторів.
IdTable.IdInfo	Елемент таблиці ідентифікаторів, реалізує *TableItem*
SegTable	Таблиця сегментів. Розширює *AbstractTable*
SegTable.SegInfo	Елемент таблиці сегментів, реалізує *TableItem*
SegTable.SegRegister	*Перелік.* Сегментні регістри
Translator	Головний клас транслятора
Translator.LstLine	Контейнер для збереження інформації про стрічку лістингу
IOLib	Клас для роботи з вводом та виводом даних
ErrorList	Список помилок при трансляції

Проект розбитий на 8 файлів вихідного коду. Нижче ви можете бачити опис та призначення кожного з файлів.

LexicalAnalyzer.java

Лексичний аналізатор. У цьому файлі реалізовані наступні класи: LexemeType, LexemeInfo там LexicalAnalyzer. LexemeType – перелік в якому перевизначений метод toString() для зручного виводу типу лексеми. LexemeInfo – контейнер для збереження інформації про одну лексему. Клас має два поля: тип лексеми і її значення та перевизначений метод toString(). Клас LexicalAnalyzer містить такі методи:

- **static LexemeInfo[] getLexemeInfo(String line)**
  Повертає інформацію про вхідну стрічку у вигляді масиву лексем.
  Parameters:
  line – Стрічка для аналізу
  Returns:
  Масив лексем
- **static String getStringToPrint(LexemeInfo[] info)**
  Перетворює масив лексем в зручну для друку стрічку.
  Parameters:
  info - Масив лексем
  Returns:
  Стрічка для друку

- **static long getConstValue(String item)**
  Перетворює стрічку у число. Для аналізу констант.
  Parameters:
  item – Число у вигляді стрічки (Bin, Dec, Hex)
  Returns:
  Перетворене число

- **static long getConstSize(long value)**
  Повертає розмір константи у байтах.(1, 2 або 4 байти) У разі переповнення повертає -1.
  Parameters:
  value – число для аналізу
  Returns:
  Розмір числа у байтах

AbstractTable.java, SegTable.java, IdTable.java

Таблиці сегментів та ідентифікаторів. Клас AbstractTable та інтерфейс TableItem виступають спільними частими для обох таблиць. Класи IdTable та SegTable - розширюють абстрактний клас, класи IdInfo та SegInfo – реалізують TableItem.

AbstractTable

- **boolean isExist(String name)**
  Перевіряє чи елемент з заданим ім’ям вже існує у таблиці.
  Parameters:
  name - Ім’я елементу
  Returns:
  Результат перевірки

- **void add(TableItem item)**
  Додає елемент до таблиці. Якщо елемент уже існує – нічого не робить.
  Parameters:
  item – Новий елемент

- **TableItem get(java.lang.String name)**
  Повертає елемент з заданим ім’ям, якщо такого елемента не існує – повертає null.
  Parameters:
  name - Ім’я елементу
  Returns:
  Елемент з заданим ім’ям

Як видно з методів вище, у таблицю можна лише додавати елементи, перевіряти їх існування у таблиці та «брати» елемент з заданим ім’ям. Це спільні функції для обох таблиць, тому було прийнято рішення винести їх у спільний абстрактний клас. Клас IdTable перевизначає метод void add(TableItem item), тепер якщо елемент вже існує у таблиці стрічка маркується як помилкова. В класі SegTable реалізований метод void assume(String assume) який виконує функцію занесення сегментів у відповідні регістри. Особливості реалізації класів IdTable та SegTable опускаються, їх можна зрозуміти при детальному вивчені коментарів файлу IdTable.java та SegTable.java.

LineInfo.java

Цей файл є найбільшим. У ньому реалізовано один клас – LineInfo. Клас має публічними лише конструктори, метод toString(), isCorrect() та поля, всі інші методи – приватні. Клас виконує функцію синтаксичного та граматичного аналізу, генерує машинний код стрічки. Також під час генерації машинного коду ідентифікатори та сегменти додаються до відповідних таблиць. Конструктор першого проходу приймає як параметр – стрічку, другого проходу – об'єкт типу LineInfo. Методи та поля:

Модифікатор доступу та тип	Поля та опис
private int	**address** Зміщення стрічки
private boolean	**isCorrect** Чи являється стрічка помилковою?
final String	**opCode** Машинний код стрічки
final int	**sizeInBytes** Розмір стрічки у байтах
private String	**template** Шаблон стрічки
final LineInfo.LineType	**type** Тип стрічки
private static final String[]	**validTemplates** Масив допустимих шаблонів
final String	**value** Вхідна стрічка

Модифікатор доступу та тип	Метод та опис
private String	**getInstructionCode**(**LexemeInfo**[] lexemes) Генерує машинний код для інструкції
private String	**getLexemeTemplate**(**LexemeInfo** lexeme) Перетворює лексему у шаблон
private String	**getLineTemplate**(String line) Перетворює стрічку у шаблон
private String	**getOpCode**(String input) Генерує машинний код для стрічки
private String	**getSegPrefix**(String reg) Повертає префікс заміни сегмента.
private LineInfo.LineType	**getTemplateType**(String template) Повертає тип шаблону.
boolean	**isCorrect**() Гетер для поля isCorrect.
public String	**toString**() Повертає зручну для виводу стрічку

IOLib.java та ErrorList.java

У цих файлах реалізовано два допоміжних класи: IOLib та ErrorList.
IOLib містить у собі 4 публічні статичні методи:
String[] readAllLine(String filePath) – зчитує всі текстовий файл за шляхом filePath у масив стрічок.
void writeAllLines(String[] lines, PrintStream writer) – записує массив стрічок за допомогою переданого PrintStream.(у файл\на екран) Та два варіанти методу String toHex(long\String item, int length) – методи переводять число у 16-ковий вигляд з заданою довжиною length.
ErrorList являється статичним сховищем для номерів помилкових стрічок. Має публічне статичне поле currentLine – яке використовується як глобальний лічильник стрічок файлу вихідного коду. Має два статичні методи для додавання номеру помилкової стрічки у список void Add() та void Add(int i), а також статичний метод для перетворення списку у стрічку для друку String getStringToPrint().

Translator.java

Виконавчий клас транслятора. У метод makeLST() створюється лістинг, а також, при необхідності файл першого проходу, та файл лексичного аналізу. Метод showHelp() виводить на екран інструкцію користування программою. Клас LstLine є зручним контейнером для зберігання інформації про стрічку лістингу.

Для кращого розуміння того як саме працює транслятор нижче описано декілька прикладів поступового формування стрічки лістингу.

Приклад 1: **`“dbVar1 db 10010011b”`**

1. За допомогою методу **`LexicalAnalyzer.getLexemeInfo(String input)`** розбиваємо стрічку на масив лексем **`LexemInfo[]`**.
2. За допомогою приватного методу **`LineInfo.getLineTemplate(String line)`** створюємо шаблон стрічки.
   У нашому випадку шаблон буде: **`“ID DB CONST”`**
3. Перевіряємо наявність шаблону у базі допустимих шаблонів. (по суті – синтаксичний аналіз) У нашому випадку: **`true`**
4. Додаємо ідентифікатор до таблиці ідентифікаторів.
5. Транслюємо команду. Записуємо результат у лістинг.

Приклад 2: **`“mov al,0123h”`**

1. За допомогою методу **`LexicalAnalyzer.getLexemeInfo(String input)`** розбиваємо стрічку на масив лексем **`LexemInfo[]`**.
2. За допомогою приватного методу **`LineInfo.getLineTemplate(String line)`** створюємо шаблон стрічки.
   У нашому випадку шаблон буде: **`“MOV reg , CONST”`**
3. Перевіряємо наявність шаблону у базі допустимих шаблонів.
   У нашому випадку: **`true`**
4. Перевіряємо розмір операндів. У нашому випадку константа займає більше ніж 1 байт(розмір регістру AL), тому стрічка маркується як помилкова.
5. Записуємо результат у лістинг.

Приклад 3: **`“label:”`**

1. За допомогою методу **`LexicalAnalyzer.getLexemeInfo(String input)`** розбиваємо стрічку на масив лексем **`LexemInfo[]`**.
2. За допомогою приватного методу **`LineInfo.getLineTemplate(String line)`** створюємо шаблон стрічки.
   У нашому випадку шаблон буде: **`“ID :”`**
3. Перевіряємо наявність шаблону у базі допустимих шаблонів.
   У нашому випадку: **`true`**
4. Перевіряємо чи мітка існує. У нашому випадку: ні.
5. Додаємо до таблиці ідентифікаторів. Записує адресу у лістинг.

Приклад 4: **`“Data segment”`**

1. За допомогою методу **`LexicalAnalyzer.getLexemeInfo(String input)`** розбиваємо стрічку на масив лексем **`LexemInfo[]`**.
2. За допомогою приватного методу **`LineInfo.getLineTemplate(String line)`** створюємо шаблон стрічки.
   У нашому випадку шаблон буде: **`“ID SEGMENT”`**
3. Перевіряємо наявність шаблону у базі допустимих шаблонів.
   У нашому випадку: **`true`**
4. Якщо сегмент існує у таблиці сегментів то поточне зміщення береться із таблиці, якщо ні – поточне зміщення скидається на нульове значення.
5. Якщо сегмент новий – додаємо до таблиці сегментів. Записує адресу у лістинг.

Приклад 5 **`“ Add cs:dbVar1[si], 00010001b”`**

1. За допомогою методу **`LexicalAnalyzer.getLexemeInfo(String input)`** розбиваємо стрічку на масив лексем **`LexemInfo[]`**.
2. За допомогою приватного методу **`LineInfo.getLineTemplate(String line)`** створюємо шаблон стрічки.
   У нашому випадку шаблон буде: **`“ADD rS : ID [ ADDR ] , CONST”`**
3. Перевіряємо наявність шаблону у базі допустимих шаблонів.
   У нашому випадку: **`true`**
4. Перевіряємо наявність ідентифікатора у таблиці. Перевіряємо розмір операндів. У нашому випадку обидві перевірки успішні.
5. Транслюємо команду. Записуємо результат у лістинг

ДОДАТОК 1
Текст модулів транслятора
LexicalAnalyzer.java

```java
package trasm;

import java.util.ArrayList;
import java.util.regex.Pattern;

/**
 * Типы лексем
 */
enum LexemeType {

    INSTRUCTION,
    /*1*/ DIRECTIVE,
    /*2*/ REGISTER_GENERAL,
    /*3*/ REGISTER_SEGMENT,
    /*4*/ DATA_TYPE,
    /*5*/ CONST_BIN,
    /*6*/ CONST_DEC,
    /*7*/ CONST_HEX,
    /*8*/ CONST_STRING,
    /*9*/ ONE_SYMBOL,
    /*10*/ USER_IDENTIFIER,
    /*11*/ ERROR_LEXEME;

    /**
     * Возвращает подробное описание типа лексемы
     *
     * @return Строка для печати
     */
    @Override
    public String toString() {
        final String[] typesDescription = {
            /*0*/"Ідентифікатор мнемокоду машинної інструкції",
            /*1*/ "Ідентифікатор директиви",
            /*2*/ "Ідентифікатор регістра загального призначення",
            /*3*/ "Ідентифікатор сегментного регістра",
            /*4*/ "Ідентифікатор дерективи даних",
            /*5*/ "Двійкова константа",
            /*6*/ "Десяткова константа",
            /*7*/ "Шістнадцяткова константа",
            /*8*/ "Текстова константа",
            /*9*/ "Односимвольна",
            /*10*/ "Ідентифікатор користувача або не визначений",
            /*11*/ "Недопустима лексема"
        };

        return typesDescription[this.ordinal()];
    }
}
```

```
/**
 * Контейнер для хранения информации про одну лексему
 */
class LexemeInfo {

    /**
     * Значение лексемы
     */
    final String value;
    /**
     * Тип лексемы
     */
    final LexemeType type;

    public LexemeInfo(String value, LexemeType type) {
        this.value = value;
        this.type = type;
    }

    /**
     * Преобразование лексемы в удобный для печати вид
     *
     * @return Строка для печати
     */
    @Override
    public String toString() {
        return String.format("%1$-8s %2$-8d %3$s", value, value.length(),
type.toString());
    }
}

/**
 * Лексический анализатор
 */
class LexicalAnalyzer {

    /**
     * Возвращает информацию про все лексемы в строке
     *
     * @param line Строчка для анализа
     * @return Массив лексем
     */
    static LexemeInfo[] getLexemeInfo(String line) {
        ArrayList<LexemeInfo> infoList = new ArrayList();

        final Pattern[] patterns = {
            /*0*/Pattern.compile("^(cli|inc|dec|add|cmp|xor|mov|or|jb|jmp)$",
Pattern.CASE_INSENSITIVE),
            /*1*/ Pattern.compile("^(segment|ends|end|assume)$",
Pattern.CASE_INSENSITIVE),
            /*2*/
Pattern.compile("^(al|cl|dl|bl|ah|ch|dh|bh|ax|cx|dx|bx|sp|bp|si|di|eax|ecx|ed
x|ebx|esp|ebp|esi|edi)$", Pattern.CASE_INSENSITIVE),
            /*3*/ Pattern.compile("^(es|cs|ss|ds|fs|gs)$",
Pattern.CASE_INSENSITIVE),
            /*4*/ Pattern.compile("^(db|dw|dd)$", Pattern.CASE_INSENSITIVE),
```

```
            /*5*/ Pattern.compile("^([01]+b)$", Pattern.CASE_INSENSITIVE),
            /*6*/ Pattern.compile("^(\\d+d?)$", Pattern.CASE_INSENSITIVE),
            /*7*/ Pattern.compile("^(\\d+[A-F0-9]*h)$",
Pattern.CASE_INSENSITIVE),
            /*8*/ Pattern.compile("^('[^']*')$", Pattern.CASE_INSENSITIVE),
            /*9*/ Pattern.compile("^[.,+\\-*/:\\[\\]]$"),
            /*10*/ Pattern.compile("^([a-z][a-z0-9]{0,7})$",
Pattern.CASE_INSENSITIVE),
            /*11*/ Pattern.compile(".*")
        };

        line = line.trim();
        line = line.replaceAll(";.*", "");
        line = line.replaceAll("([,:\\[\\]])", " $1 ");

        String[] lexemes = line.split("[\\s\\t]+");

        for (String lexeme : lexemes) {
            for (LexemeType lexemeType : LexemeType.values()) {
                if(patterns[lexemeType.ordinal()].matcher(lexeme).matches())
{
                    infoList.add(new LexemeInfo(lexeme, lexemeType));
                    break;
                }
            }
        }

        return infoList.toArray(new LexemeInfo[infoList.size()]);
    }

    /**
     * Преобразовывает массив лексем в удобный для печати вид
     *
     * @param info Массив лексем
     * @return Строка для печати
     */
    static String getStringToPrint(LexemeInfo[] info) {
        StringBuilder outStr;
        outStr = new StringBuilder(String.format("%1$-8s %2$-8s %3$-8s
%4$s\n", "№ п/п", "Лексема", "Довжина", "Тип лексеми"));

        for (int i = 0; i < info.length; i++) {
            LexemeInfo lexemeInfo = info[i];
            outStr = outStr.append(String.format("%1$-8d %2$s\n", i + 1,
lexemeInfo.toString()));
        }
        return outStr.toString();
    }
```

```
    /**
     * Преобразовывает строку в число
     *
     * @param item Строка для преобразования (Bin, Dec, Hex)
     * @return Значение строки
     */
    static long getConstValue(String item) {
        if (item.toUpperCase().contains("H")) {
            return Long.parseLong(item.substring(0, item.length() - 1), 16);
        }
        if (item.toUpperCase().contains("B")) {
            return Long.parseLong(item.substring(0, item.length() - 1), 2);
        }
        if (item.toUpperCase().contains("D")) {
            return Long.parseLong(item.substring(0, item.length() - 1));
        }

        return Long.parseLong(item);
    }

    /**
     * Возвращает размер константы: 1,2 или 4 байта. В случае переполнения
     * возвращает -1;
     *
     * @param value Входное значение
     * @return Размер в байтах
     */
    static int getConstSize(long value) {
        if (value > Integer.MAX_VALUE || value < Integer.MIN_VALUE) {
            return -1;
        }
        return value < 256 ? 1 : (value < 256 * 256 ? 2 : 4);
    }

    /**
     * Возвращает размер константы: 1,2 или 4 байта. В случае переполнения
     * возвращает -1;
     *
     * @param value Строка для преобразования в число
     * @return Размер в байтах
     */
    static int getConstSize(String value) {
        return getConstSize((int) getConstValue(value));
    }
}
```

AbstractTable.java

```java
package trasm;

import java.util.ArrayList;

/**
 * ИнтИерфейс элемента таблицы сегментов/идентификаторов
 */
interface TableItem {

    String getName();

    @Override
    String toString();
}

/**
 * Абстрактный класс для таблицы сегментов/идентификаторов
 */
abstract class AbstractTable {

    /**
     * Список для хранения элементов таблицы
     */
    protected final ArrayList<TableItem> list = new ArrayList<>();

    /**
     * Проверяет или элемент с заданным именем существует
     *
     * @param name Имя элемента
     * @return Ответ на главный вопрос
     */
    boolean isExist(String name) {
        for (TableItem tableItem : list) {
            if (name.equalsIgnoreCase(tableItem.getName())) {
                return true;
            }
        }

        return false;
    }

    /**
     * Добаляет элемент в таблицу, если он уже существует - ничего не
происходит
     *
     * @param item Новый элемент
     */
    void add(TableItem item) {
        if (!isExist(item.getName())) {
            list.add(item);
        }
    }
```

```
    /**
     * Возвращает элемент с заданным именем. В случае если элемент не
существует
     * - null.
     *
     * @param name имя элемента
     * @return Элемент с заданным именем
     */
    TableItem get(String name) {
        for (TableItem tableItem : list) {
            if (name.equalsIgnoreCase(tableItem.getName())) {
                return tableItem;
            }
        }

        return null;
    }
}
```

ErrorList.java

```
package trasm;

import java.util.ArrayList;

/**
 * Список ошибок при создании листинга.
 */
//p.s. Очень быдлокод...
class ErrorList {

    /**
     * Список с номерами строк
     */
    private static final ArrayList<Integer> errorLineList = new
ArrayList<>();
    /**
     * Глобальная переменная хранящая текущий номер строки
     */
    static int currentLine = 1;

    /**
     * Добавить ошибку. (номер строки = currentLine)
     */
    static void AddError() {
        errorLineList.add(currentLine);
    }

    /**
     * Добавить ошибку с заданным номером строки
     *
     * @param line Номер строки
     */
    static void AddError(int line) {
        errorLineList.add(line);
    }
```

```
    /**
     * Строка для печати с количеством ошибок и номерами строчек с ошибками
     *
     * @return Строка для печати
     */
    static String getStringToPrint() {
        StringBuilder outStr;
        outStr = new StringBuilder("Помилки: ");

        outStr =
outStr.append(errorLineList.size()).append((errorLineList.isEmpty()) ? "\n" :
"\nРядки з помилками: ");

        for (Integer errorLine : errorLineList) {
            outStr = outStr.append(errorLine).append(" ");
        }

        return outStr.toString();
    }
}
```

IOLib.java

```
package trasm;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.Scanner;

/**
 * Класс для работы с вводом и выводом данных
 */
class IOLib {

    /**
     * Считывает текстовый файл в массив строк
     *
     * @param filePath Путь к текстовому файлу
     * @return Массив строк содержащий в себе каждую строку из входного файла
     * @throws FileNotFoundException
     * @throws IOException
     */
    static public String[] readAllLines(String filePath) throws
FileNotFoundException, IOException {
        ArrayList<String> outList = new ArrayList<>();

        try (Scanner scanner = new Scanner(new File(filePath))) {
            while (scanner.hasNextLine()) {
                outList.add(scanner.nextLine());
            }
        }
        return outList.toArray(new String[outList.size()]);    }
```

```
    /**
     * Записывает массив строк в поток
     *
     * @param lines Массив строк для записи
     * @param writer Врайтер
     * @throws FileNotFoundException
     * @throws IOException
     */
    static public void writeAllLines(String[] lines, PrintStream writer)
throws FileNotFoundException, IOException {
         String about = "Курсова робота студента КПІ ФПМ групи КВ-23
Чугаєвського Максима Варіант 1\n";
        SimpleDateFormat sdfDate = new SimpleDateFormat("dd/mm/yyyy
HH:mm:ss");
        Date now = new Date();
        about += "Згенеровано: " + sdfDate.format(now);

        writer.println(about);

        for (String line : lines) {
            writer.println(line);
        }
        if (!writer.equals(System.out)) {
            writer.close();
        }
    }

    /**
     * Преобразовывает число в hex вид с заданной шириной
     *
     * @param i Число для преобразования
     * @param lenght Длина выходной строки
     * @return Число в hex формате
     */
    static String toHex(long i, int lenght) {
        StringBuilder outStr = new
StringBuilder(Long.toHexString(i).toUpperCase());

        for (int j = outStr.length(); j < lenght; j++) {
            outStr.insert(0, "0");
        }

        return outStr.length() != lenght ? outStr.substring(outStr.length() -
lenght, outStr.length()) : outStr.toString();
    }

    /**
     * Преобразовывает число в hex вид с заданной шириной
     *
     * @param value Строка для преобразования
     * @param lenght Длина выходной строки
     * @return Число в hex формате
     */
    static String toHex(String value, int lenght) {
        return toHex(LexicalAnalyzer.getConstValue(value), lenght);
    }
}
```

IdTable.java

```
package trasm;

/**
 * Таблица идентификаторов
 */
class IdTable extends AbstractTable {

    /**
     * Типы идентификаторов
     */
    static enum IdType {

        DB(1), DW(2), DD(4), LABEL(0);

        private int size;

        private IdType(int i) {
            size = i;
        }

        /**
         * Возвращает тип идентификатора в байтах(для DB,DW,DD)
         *
         * @return Тип идентификатора
         */
        int getSize() {
            return size;
        }
    }

    /**
     * Добавляет в таблицу новый элемент. Если элемент существует -
записывает
     * строчку как ошибочную.
     *
     * @param item Новый элемент
     */
    void add(TableItem item) {
        if (isExist(item.getName())) {
            ErrorList.AddError();
        } else {
            super.add(item);
        }
    }

    private static IdTable instance = null;

    private IdTable() {
    }
```

```
/**
 * Возвращает единственный экземпляр таблицы идентификаторов
 *
 * @return Таблица идентификаторов
 */
static IdTable getInstance() {
    if (instance == null) {
        instance = new IdTable();
    }
    return instance;
}

/**
 * Элемент таблицы идентификаторов
 */
static class IdInfo implements TableItem {

    private final String name;
    private final String segment;
    private final int address;
    private final IdType type;

    /**
     * Конструктор элемента таблицы идентификаторов
     *
     * @param name имя нового элемента
     * @param type Тип нового элемента
     */
    public IdInfo(String name, IdType type) {
        this.name = name;
        this.segment = SegTable.getInstance().getCurrentSegment();
        this.address = SegTable.getInstance().getCurrentAddress();
        this.type = type;
    }

    /**
     * Возвращает тип идентификатора
     *
     * @return Тип идентификатора
     */
    public IdType getType() {
        return type;
    }

    /**
     * Возвращает смещение элемента
     *
     * @return Смещение элемента
     */
    public int getAddress() {
        return address;
    }
```

```
        /**
         * Возвращает сегмент в котором объявлен элемент
         *
         * @return Сегмент элемента
         */
        public String getSegment() {
            return segment;
        }

        /**
         * Возвращает имя элемента
         *
         * @return имя элемента
         */
        @Override
        public String getName() {
            return name;
        }

        /**
         * Преобразовывает элемент в удобный для чтения вид
         *
         * @return Строка для печати
         */
        @Override
        public String toString() {
            return String.format("%1$-8s %2$-8s %3$s:%4$s\n", name,
type.toString(), segment, IOLib.toHex(address, 4));
        }

    }

    /**
     * Возвращает таблицу идентификторов в удобном для чтения виде
     *
     * @return Строка для печати
     */
    @Override
    public String toString() {

        StringBuilder outStr;
        outStr = new StringBuilder("Ім'я     Тип      Адреса\n");

        for (TableItem tableItem : list) {
            outStr = outStr.append(tableItem.toString());
        }

        return outStr.toString();
    }
}
```

SegTable.java

```
package trasm;

/**
 * Таблица сегментов
 */
class SegTable extends AbstractTable {

    /**
     * Пустой сегмент
     */
    static final String NULL_SEG_NAME = "NOTHING";
    /**
     * Текущий сегмент
     */
    private static String currentSegment = SegTable.NULL_SEG_NAME;
    /**
     * Текущее смещение
     */
    private static int currentAddress = 0;
    /**
     * Состояние Assume-а
     */
    private SegInfo[] assumeSegs = {new SegInfo(), new SegInfo(), new
SegInfo(),
        new SegInfo(), new SegInfo(), new SegInfo()}; //что бы не
инициализировать
    private static SegTable instance = null;

    private SegTable() {

    }

    /**
     * Возвращает единственный экземпляр таблицы сегментов
     *
     * @return Таблица сегментов
     */
    static SegTable getInstance() {
        if (instance == null) {
            instance = new SegTable();
        }
        return instance;
    }

    /**
     * Сегментные регистры
     */
    static enum SegRegister {

        ES, CS, SS, DS, FS, GS
    }
```

```java
/**
 * Элемент таблицы сегментов
 */
static class SegInfo implements TableItem {

    /**
     * Имя сегмента
     */
    private final String name;
    /**
     * Размер сегмента
     */
    private int size;

    public SegInfo() {
        name = NULL_SEG_NAME;
        size = 0;
    }

    public SegInfo(String name) {
        this.name = name;
        this.size = 0;
    }

    /**
     * Устанавливает размер сегмента
     *
     * @param size Новый размер сегмента
     */
    public void setSize(int size) {
        this.size = size;
    }

    /**
     * Возвращает размер сегмента
     *
     * @return Размер сегмента
     */
    public int getSize() {
        return size;
    }

    /**
     * Возвращает имя сегмента
     *
     * @return имя сегмента
     */
    @Override
    public String getName() {
        return name;
    }
```

```
    /**
     * Возвращает описание сегмента в удобном для печати виде
     *
     * @return Строка для печати
     */
    @Override
    public String toString() {
    return String.format("%1$-8s %2$-4s\n", name, IOLib.toHex(size,4));
    }
}

/**
 * Возвращает текущий сегмент
 *
 * @return Текущий сегмент
 */
String getCurrentSegment() {
    return currentSegment;
}

/**
 * Задает текущий сегмент
 *
 * @param newSegment Новый текущий сегмент
 */
void setCurrentSegment(String newSegment) {
    currentSegment = newSegment;
}

/**
 * Возвращает текущее смещение
 *
 * @return текущее смещение
 */
int getCurrentAddress() {
    return currentAddress;
}

/**
 * Задает текущее смещение
 *
 * @param newAddress Новое текущее смещение
 */
void setCurrentAddress(int newAddress) {
    currentAddress = newAddress;
}
```

```
    /**
     * Устанавливает новый размер сегмента
     *
     * @param segment имя сегмента
     * @param size Новый размер сегмента
     */
    void setSize(String segment, int size) {
        if (isExist(segment)) {
            for (int i = 0; i < list.size(); i++) {
                if (list.get(i).getName().equalsIgnoreCase(segment)) {
                    SegInfo info = (SegInfo) list.get(i);
                    info.setSize(size);
                    list.set(i, info);
                    break;
                }
            }
        }
    }
    /**
     * Assume
     *
     * @param assume Строчка для Assume-а
     */
    void assume(String assume) {
        LexemeInfo[] lexemes =
LexicalAnalyzer.getLexemeInfo(assume.toUpperCase());
        int index = 0;

        String segName = NULL_SEG_NAME;
        for (LexemeInfo lexemeInfo : lexemes) {
            if (lexemeInfo.type == LexemeType.REGISTER_SEGMENT) {
                index = SegRegister.valueOf(lexemeInfo.value).ordinal();
            } else if (lexemeInfo.type == LexemeType.USER_IDENTIFIER) {
                segName = lexemeInfo.value.toUpperCase();
            } else if (lexemeInfo.value.equals(",")) {
                assumeSegs[index] = new SegInfo(segName);
            }
        }

        assumeSegs[index] = new SegInfo(segName);
    }
    /**
     * Возвращает удобное для печати состояние сегментных регистров
     *
     * @return Состояние сегментных регистров
     */
    String assumeToString() {
        StringBuilder outStr;
        outStr = new StringBuilder(String.format("Сегмент  Регистр\n"));

        for (SegRegister segReg : SegRegister.values()) {
            outStr = outStr.append(String.format("%1$-8s %2$s\n",
assumeSegs[segReg.ordinal()].name, segReg.toString()));
        }

        return outStr.toString();
    }
```

```
    /**
     * Возвращает сегмент который сейчас "лежит" заданном регистре
     *
     * @param segment Сегментный регистр
     * @return Сегмент лежащий в этом регистре
     */
    SegRegister getSegmentReg(String segment) {
        for (SegRegister segReg : SegRegister.values()) {
            SegInfo segInfo = assumeSegs[segReg.ordinal()];
            if (segInfo.name.equalsIgnoreCase(segment)) {
                return segReg;
            }
        }
        return null;
    }

    /**
     * Возвращает таблицу сегментов в удобном для чтения виде
     *
     * @return Строка для печати
     */
    @Override
    public String toString() {
        StringBuilder outStr;
        outStr = new StringBuilder("Сегмент  Розмір\n");

        for (TableItem tableItem : list) {
            outStr = outStr.append(tableItem.toString());
        }

        return outStr.toString();
    }
}
```

LineInfo.java

```
package trasm;

import java.nio.charset.StandardCharsets;
import trasm.IdTable.IdInfo;
import trasm.IdTable.IdType;
import trasm.SegTable.SegInfo;
import trasm.SegTable.SegRegister;

/**
 * Контейнер для хранения информации про исходную строчку
 */
class LineInfo {

    /**
     * Типы строчек
     */
    static enum LineType {

        BEGIN_SEGMENT,
        END_SEGMENT,
```

```
        DATA_DECLARATION,
        LABEL,
        ASSUME,
        INSTRUCTIONS,
        JUMP,
        END,
        ERROR_LINE,
        EMPTY
    }
    /**
     * Регистры общего назначения. (необходимы для генерации кода операции)
     */
    private enum Register {

        AL(0, 1), CL(1, 1), DL(2, 1), BL(3, 1), AH(4, 1), CH(5, 1), DH(6, 1),
BH(7, 1),
        AX(0, 2), CX(1, 2), DX(2, 2), BX(3, 2), SP(4, 2), BP(5, 2), SI(6, 2),
DI(7, 2),
        EAX(0, 4), ECX(1, 4), EDX(2, 4), EBX(3, 4), ESP(4, 4), EBP(5, 4),
ESI(6, 4), EDI(7, 4);

        private final int num, size;

        public int getSize() {
            return size;
        }

        public int getNum() {
            return num;
        }

        public static String getModRM(int base, Register index, boolean
isAddress) {
            int addr = (isAddress ? 0x80 : 0xC0) + 0x08 * base;

            if (index.name().contains("E")) {
                addr += index.getNum();
            }

            switch (index) {
                case SI:
                    addr += 4;
                    break;
                case DI:
                    addr += 5;
                    break;
                case BP:
                    addr += 6;
                    break;
                case BX:
                    addr += 7;
                    break;
                case ESP:
                    return IOLib.toHex(addr, 2) + " 24 ";
            }

            return IOLib.toHex(addr, 2) + " ";    }
```

```
        public static String getModRM(Register base, Register index, boolean
isAddress) {
            return Register.getModRM(base.getNum(), index, isAddress);
        }

        private Register(int num, int size) {
            this.num = num;
            this.size = size;
        }

    }

    /**
     * Исходное значение строки
     */
    final String value;
    /**
     * Размер строки в байтах (кода операции)
     */
    final int sizeInBytes;
    /**
     * Тип строки
     */
    final LineType type;
    /**
     * Код операции. Машинная трансляция строки
     */
    final String opCode;
    /**
     * Содержит ли строчка ошибку?
     */
    private boolean isCorrect;
    /**
     * Шаблон строчки (для внутринних нужд)
     */
    private String template;
    /**
     * Смещение строчки (для внутринних нужд)
     */
    private int address;

    //<editor-fold defaultstate="collapsed" desc="validTemplates">
    /**
     * Допустимые шаблоны строк
     */
    private static final String[] validTemplates = {
        /*0*/"ID SEGMENT",
        /*1*/ "ID ENDS",
        /*2*/ "ASSUME rS : ID",
        /*3*/ "END ID",
        /*4*/ "END",
        /*5*/ "ID :",
        /*6*/ "ID DB CONST",
        /*7*/ "ID DW CONST",
        /*8*/ "ID DD CONST",
        /*9*/ "ID DB C_STR",
        /*10*/ "CLI",
```

```
            /*11*/ "INC ID [ ADDR ]",
            /*12*/ "INC rS : ID [ ADDR ]",
            /*13*/ "DEC reg",
            /*14*/ "ADD ID [ ADDR ] , CONST",
            /*15*/ "ADD rS : ID [ ADDR ] , CONST",
            /*16*/ "CMP reg , ID [ ADDR ]",
            /*17*/ "CMP reg , rS : ID [ ADDR ]",
            /*18*/ "XOR ID [ ADDR ] , reg",
            /*19*/ "XOR rS : ID [ ADDR ] , reg",
            /*20*/ "MOV reg , CONST",
            /*21*/ "OR reg , reg",
            /*22*/ "JB ID",
            /*23*/ "JMP ID",
            /*24*/ ""
    };
//</editor-fold>

    boolean isCorrect() {
        return isCorrect;
    }

    /**
     * Преобразование строчки в удобный для печати вид
     *
     * @return Строка для печати
     */
    @Override
    public String toString() {
        return String.format("%1$-20s %2$s", opCode, value);
    }

    /**
     * Конструктор для второго прохода. Специально для команда JMP и JB.
     *
     * @param info Результат превого прохода
     */
    public LineInfo(LineInfo info) {
        value = info.value;
        sizeInBytes = info.sizeInBytes;
        type = info.type;
        address = info.address;
        isCorrect = info.isCorrect;
        template = info.template;
        opCode = getOpCode(value);
    }

    /**
     * Конструктор для первого прохода.
     *
     * @param line Исходная строчка
     */
    public LineInfo(String line) {
        template = getLineTemplate(line);
        template = template.replaceAll("(\\[ r16a \\])|(\\[ r32 \\])", "[
ADDR ]").replaceAll("r8|r16a|r16|r32", "reg");
        if (template.contains("DB C_STR") == false) {
            template = template.replace("C_STR", "CONST");}
```

```
        if (template.contains("ASSUME")) {
            template = template.replaceAll(" , rS : ID", "");
        }

        this.value = line;
        this.address = SegTable.getInstance().getCurrentAddress();
        this.type = getTemplateType(template);
        this.isCorrect = type != LineType.ERROR_LINE;
        this.opCode = getOpCode(line);
        this.sizeInBytes = opCode.replaceAll("[|\\s]", "").length() / 2;
    }

    /**
     * Преобразование лексемы к шаблону
     *
     * @param lexeme Лексема
     * @return Шаблон
     */
    private String getLexemeTemplate(LexemeInfo lexeme) {
        String outStr = "So empty...";
        switch (lexeme.type) {
            case INSTRUCTION:
                outStr = lexeme.value.toUpperCase();
                break;
            case DIRECTIVE:
                outStr = lexeme.value.toUpperCase();
                break;
            case REGISTER_GENERAL:
                String reg = lexeme.value.toUpperCase();
                if (reg.contains("L") || reg.contains("H")) {
                    outStr = "r8";
                } else if (reg.contains("E")) {
                    outStr = "r32";
                } else if (reg.contains("B") || reg.contains("I")) {
                    outStr = "r16a";
                } else {
                    outStr = "r16";
                }
                break;
            case REGISTER_SEGMENT:
                outStr = "rS";
                break;
            case DATA_TYPE:
                outStr = lexeme.value.toUpperCase();
                break;
            case CONST_BIN:
            case CONST_DEC:
            case CONST_HEX:
                outStr = "CONST";
                break;
            case CONST_STRING:
                outStr = "C_STR";
                break;
            case ONE_SYMBOL:
                outStr = lexeme.value;
                break;
            case USER_IDENTIFIER:
```

```
                outStr = "ID";
                break;
            case ERROR_LEXEME:
                if (lexeme.value.isEmpty()) {
                    return "";
                }
                outStr = "<Error lexeme, sad... So sad... =( " + lexeme.value
+ ">";
                break;
            default:
                throw new AssertionError(lexeme.type.name());
        }

        return outStr;
    }

    /**
     * Преобразование строчки к шаблону
     *
     * @param line ?сходная строчка
     * @return Шаблон
     */
    private String getLineTemplate(String line) {
        LexemeInfo[] lexemeInfo = LexicalAnalyzer.getLexemeInfo(line);
        StringBuilder template = new
StringBuilder(getLexemeTemplate(lexemeInfo[0]));

        for (int i = 1; i < lexemeInfo.length; i++) {
            template = template.append("
").append(getLexemeTemplate(lexemeInfo[i]));
        }

        return template.toString();
    }

    /**
     * Возвращает тип шаблона
     *
     * @param template Шаблон
     * @return Тип шаблона
     */
    private LineType getTemplateType(String template) {

        if (!template.equals("ID :") && template.startsWith("ID :")) {
            template = template.substring(template.indexOf(":") + 1).trim();
        }

        int index = -1;
        for (int i = 0; i < validTemplates.length; i++) {
            if (validTemplates[i].equals(template)) {
                index = i;
                break;
            }
        }

        if (index == -1) {
            return LineType.ERROR_LINE;  }
```

```
        switch (index) {
            case 0:
                return LineType.BEGIN_SEGMENT;
            case 1:
                return LineType.END_SEGMENT;
            case 2:
                return LineType.ASSUME;
            case 3:
            case 4:
                return LineType.END;
            case 5:
                return LineType.LABEL;
            case 22:
            case 23:
                return LineType.JUMP;
            case 24:
                return LineType.EMPTY;
            default:
                if (index < 10) {
                    return LineType.DATA_DECLARATION;
                }
                return LineType.INSTRUCTIONS;
        }
    }

    /**
     * Генерация кода операции для строчки
     *
     * @param input исходная строчка
     * @return Код операции
     */
    private String getOpCode(String input) {

        SegTable segTable = SegTable.getInstance();
        IdTable idTable = IdTable.getInstance();

        while (!template.equals("ID :") && template.startsWith("ID :")) {
            String label = input.substring(0, input.indexOf(":")).trim();
            input = input.substring(input.indexOf(":") + 1).trim();
            idTable.add(new IdInfo(label, IdType.LABEL));
            template = getLineTemplate(input);
        }

        LexemeInfo[] lexemes = LexicalAnalyzer.getLexemeInfo(input);

        switch (type) {
            case BEGIN_SEGMENT:
                if
(!segTable.getCurrentSegment().equals(SegTable.NULL_SEG_NAME)) {
                    isCorrect = false;
                } else {
                    segTable.setCurrentSegment(lexemes[0].value);
                    if (!segTable.isExist(lexemes[0].value)) {
                        segTable.add(new SegInfo(lexemes[0].value));
                    }
```

```
                    segTable.setCurrentAddress(((SegInfo)
segTable.get(lexemes[0].value)).getSize());
                }
                return "";
            case END_SEGMENT:
                if
(!segTable.getCurrentSegment().equalsIgnoreCase(lexemes[0].value)) {
                    isCorrect = false;
                } else {
                    segTable.setSize(segTable.getCurrentSegment(), address);
                }
                segTable.setCurrentSegment(SegTable.NULL_SEG_NAME);
                return "";
            case DATA_DECLARATION:
                if (lexemes[2].type == LexemeType.CONST_STRING) {
                    if (!lexemes[1].value.equalsIgnoreCase("DB")) {
                        isCorrect = false;
                        return "";
                    }
                    String constStr = lexemes[2].value.substring(1,
lexemes[2].value.length() - 1);
                    String constHex = "";
                    for (byte b : constStr.getBytes(StandardCharsets.UTF_8))
{
                        constHex += Integer.toHexString(b & 0xFF) + " ";
                    }

                    idTable.add(new IdInfo(lexemes[0].value, IdType.DB));

                    return constHex.trim().toUpperCase();
                }
                int immSize = LexicalAnalyzer.getConstSize(lexemes[2].value);

                IdType idType =
IdType.valueOf(lexemes[1].value.toUpperCase());

                idTable.add(new IdInfo(lexemes[0].value, idType));

                if (idType.getSize() < immSize || immSize == -1) {
                    isCorrect = false;
                    return "";
                }

                return IOLib.toHex(lexemes[2].value, immSize * 2);
            case LABEL:
                idTable.add(new IdInfo(lexemes[0].value, IdType.LABEL));
                return "";
            case ASSUME:
                segTable.assume(input);
                return "";
            case INSTRUCTIONS:
            case JUMP:
                return getInstructionCode(lexemes);
            case ERROR_LINE:
                isCorrect = false;
            default:
                return "";      }     }
```

```
    /**
     * Генерация кода операции для операций.
     *
     * @param lexemes Массив лексем
     * @return Код операции
     */
    private String getInstructionCode(LexemeInfo[] lexemes) {

        SegTable segTable = SegTable.getInstance();
        IdTable idTable = IdTable.getInstance();

        final String addrPrefix = "66| ";
        final String regPrefix = "67| ";
        boolean isAddrPref, isRegPref, isSegPref;
        String segPrefix = "";
        int immSize = 0;
        IdInfo idInfo = null;
        Register reg = null;
        SegRegister idSeg = null;
        switch (lexemes[0].value.toUpperCase()) {
            case "INC": {
                //FE /0 — INC r/m8
                //FF /0 — INC r/m16
                //FF /0 — INC r/m32
                isSegPref = lexemes[1].type == LexemeType.REGISTER_SEGMENT;
                // INC(0) ID(1) [(2) REG(3) ](4)
                // INC(0) S_REG(1) :(2) ID(3) [(4) REG(5) ](5)

                if (isSegPref) {
                    segPrefix = getSegPrefix(lexemes[1].value);
                }

                idInfo = (IdInfo) (idTable.get(lexemes[isSegPref ? 3 :
1].value));

                if (idInfo == null) {
                    isCorrect = false;
                    return "";
                }

                idSeg = segTable.getSegmentReg(idInfo.getSegment());
                if (idSeg != SegTable.SegRegister.DS && !isSegPref) {
                    segPrefix = getSegPrefix(idSeg.name());
                }

                isAddrPref = idInfo.getType() == IdType.DD;
                reg = Register.valueOf(lexemes[isSegPref ? 5 :
3].value.toUpperCase());
                isRegPref = reg.getSize() == 4;

                return (isAddrPref ? addrPrefix : "") + segPrefix +
(isRegPref ? regPrefix : "")
                        + "FE " + Register.getModRM(0, reg, true) +
IOLib.toHex(idInfo.getAddress(), (isRegPref ? 8 : 4)); }
```

```
            case "DEC": {
                //FE /1 — DEC r/m8
                //48+rw — DEC r16
                //48+rd — DEC r32
                // DEC(0) REG(1)

                reg = Register.valueOf(lexemes[1].value.toUpperCase());
                isRegPref = reg.getSize() == 4;

                return (isRegPref ? regPrefix : "") + (reg.getSize() == 1 ?
"FE "
                        + IOLib.toHex(0xC8 + reg.getNum(), 2) :
IOLib.toHex(0x48 + reg.getNum(), 2));
            }
            case "ADD": {
                //80 /0 ib — ADD r/m8,imm8
                //81 /0 iw — ADD r/m16,imm16
                //81 /0 id — ADD r/m32,imm32
                //83 /0 ib — ADD r/m16,imm8
                //83 /0 ib — ADD r/m32,imm8

                isSegPref = lexemes[1].type == LexemeType.REGISTER_SEGMENT;
                // ADD(0) ID(1) [(2) REG(3) ](4) ,(5) CONST(6)
                // ADD(0) S_REG(1) :(2) ID(3) [(4) REG(5) ](6) ,(7) CONST(8)

                if (isSegPref) {
                    segPrefix = getSegPrefix(lexemes[1].value);
                }

                idInfo = (IdInfo) (idTable.get(lexemes[isSegPref ? 3 :
1].value));
                if (idInfo == null || idInfo.getType().getSize() < immSize ||
immSize == -1) {
                    isCorrect = false;
                    return "";
                }

                idSeg = segTable.getSegmentReg(idInfo.getSegment());
                if (idSeg != SegTable.SegRegister.DS && !isSegPref) {
                    segPrefix = getSegPrefix(idSeg.name());
                }
                long imm = LexicalAnalyzer.getConstValue(lexemes[isSegPref ?
8 : 6].value);
                immSize = LexicalAnalyzer.getConstSize(imm);

                isAddrPref = idInfo.getType() == IdType.DD;
                reg = Register.valueOf(lexemes[isSegPref ? 5 :
3].value.toUpperCase());
                isRegPref = reg.getSize() == 4;

                return (isAddrPref ? addrPrefix : "") + segPrefix +
(isRegPref ? regPrefix : "")
                        + (immSize == 1 && idInfo.getType().getSize() != 1 ?
"83 " : idInfo.getType().getSize() == 1 ? "80 " : "81 ")
                        + Register.getModRM(0, reg, true) +
IOLib.toHex(idInfo.getAddress(), (isRegPref ? 8 : 4))
```

```
                        + " " + IOLib.toHex(imm, immSize != 1 ?
idInfo.getType().getSize() * 2 : 2);

            }
            case "CMP": {
                //3A /r — CMP r8,r/m8
                //3B /r — CMP r16,r/m16
                //3B /r — CMP r32,r/m32

                isSegPref = lexemes[3].type == LexemeType.REGISTER_SEGMENT;
                // CMP(0) FIRST_REG(1) ,(2) ID(3) [(4) REG(5) ](6)
                // CMP(0) FIRST_REG(1) ,(2) S_REG(3) :(4) ID(5) [(6) REG(7)
](8)

                if (isSegPref) {
                    segPrefix = getSegPrefix(lexemes[3].value);
                }

                idInfo = (IdInfo) (idTable.get(lexemes[isSegPref ? 5 :
3].value));
                Register firstReg =
Register.valueOf(lexemes[1].value.toUpperCase());
                if (idInfo == null || idInfo.getType().getSize() !=
firstReg.getSize()) {
                    isCorrect = false;
                    return "";
                }

                idSeg = segTable.getSegmentReg(idInfo.getSegment());
                if (idSeg != SegTable.SegRegister.DS && !isSegPref) {
                    segPrefix = getSegPrefix(idSeg.name());
                }

                isAddrPref = idInfo.getType() == IdType.DD;
                reg = Register.valueOf(lexemes[isSegPref ? 7 :
5].value.toUpperCase());
                isRegPref = reg.getSize() == 4;

                return (isAddrPref ? addrPrefix : "") + segPrefix +
(isRegPref ? regPrefix : "")
                        + (idInfo.getType() == IdType.DB ? "3A " : "3B ") +
Register.getModRM(firstReg, reg, true)
                        + IOLib.toHex(idInfo.getAddress(), (isRegPref ? 8 :
4));
            }
            case "XOR": {
                //30 /r — XOR r/m8,r8
                //31 /r — XOR r/m16,r16
                //31 /r — XOR r/m32,r32

                isSegPref = lexemes[1].type == LexemeType.REGISTER_SEGMENT;
                // XOR(0) ID(1) [(2) REG(3) ](4) ,(5) SECOND_REG(6)
                // XOR(0) S_REG(1) :(2) ID(3) [(4) REG(5) ](6) ,(7)
SECOND_REG(8)

                if (isSegPref) {
                    segPrefix = getSegPrefix(lexemes[1].value);
```

```
                }

                idInfo = (IdInfo) (idTable.get(lexemes[isSegPref ? 3 :
1].value));
                Register secondReg = Register.valueOf(lexemes[isSegPref ? 8 :
6].value.toUpperCase());
                if (idInfo == null || idInfo.getType().getSize() !=
secondReg.getSize()) {
                    isCorrect = false;
                    return "";
                }

                idSeg = segTable.getSegmentReg(idInfo.getSegment());
                if (idSeg != SegTable.SegRegister.DS && !isSegPref) {
                    segPrefix = getSegPrefix(idSeg.name());
                }

                isAddrPref = idInfo.getType() == IdType.DD;
                reg = Register.valueOf(lexemes[isSegPref ? 5 :
3].value.toUpperCase());
                isRegPref = reg.getSize() == 4;

                return (isAddrPref ? addrPrefix : "") + segPrefix +
(isRegPref ? regPrefix : "")
                        + (idInfo.getType() == IdType.DB ? "30 " : "31 ") +
Register.getModRM(secondReg, reg, true)
                        + IOLib.toHex(idInfo.getAddress(), (isRegPref ? 8 :
4));
            }
            case "MOV": {
                //B0+rb — MOV r8,imm8
                //B8+rw — MOV r16,imm16
                //B8+rd — MOV r32,imm32
                // MOV(0) REG(1) ,(2) imm(3)

                immSize = LexicalAnalyzer.getConstSize(lexemes[3].value);
                reg = Register.valueOf(lexemes[1].value.toUpperCase());
                isAddrPref = reg.getSize() == 4;

                if (reg.getSize() < immSize) {
                    isCorrect = false;
                    return "";
                }

                return (isAddrPref ? addrPrefix : "")
                        + (reg.getSize() == 1 ? IOLib.toHex(0xB0 +
reg.getNum(), 2) : IOLib.toHex(0xB8 + reg.getNum(), 2))
                        + " " + IOLib.toHex(lexemes[3].value, reg.getSize() *
2);
            }
            case "OR": {
                //0A /r — OR r8,r/m8
                //0B /r — OR r16,r/m16
                //0B /r — OR r32,r/m32
                // OR(0) REG(1) ,(2) SECOND_REG(3)

                reg = Register.valueOf(lexemes[1].value.toUpperCase());
```

```
                Register secondReg =
Register.valueOf(lexemes[3].value.toUpperCase());
                isAddrPref = reg.getSize() == 4;

                if (reg.getSize() != secondReg.getSize()) {
                    isCorrect = false;
                    return "";
                }

                return (isAddrPref ? addrPrefix : "") + (reg.getSize() == 1 ?
IOLib.toHex(0x0A, 2) : IOLib.toHex(0x0B, 2))
                        + " " + Register.getModRM(reg, secondReg, false);
            }
            case "JB": {
                //72 cb – JB rel8
                //0F 82 cw/cd – JB rel16/32
                //JB(0) ID(1)

                idInfo = (IdInfo) (idTable.get(lexemes[1].value));

                if (idInfo == null) {
                    if (Transtalor.isSecondPass) {
                        isCorrect = false;
                        return "";
                    }
                    return "90 90 90 90";
                }
                int jumpWidth = idInfo.getAddress() - (address + 2);
                if (jumpWidth > -128 && jumpWidth < 0) {
                    return "72 " + IOLib.toHex(jumpWidth, 2);
                }
                return jumpWidth < 127 && jumpWidth > -128 ? "72 " +
IOLib.toHex(jumpWidth, 2) + " 90 90" : "0F 82 " + IOLib.toHex(jumpWidth - 2,
4);
            }
            case "JMP": {
                //EB cb – JMP rel8
                //E9 cw – JMP rel16
                //JMP(0) ID(1)

                idInfo = (IdInfo) (idTable.get(lexemes[1].value));

                if (idInfo == null) {
                    if (Transtalor.isSecondPass) {
                        isCorrect = false;
                        return "";
                    }
                    return "90 90 90";
                }
                int jumpWidth = idInfo.getAddress() - (address + 2);
                if (jumpWidth > -128 && jumpWidth < 0) {
                    return "EB " + IOLib.toHex(jumpWidth, 2);
                }
                return jumpWidth < 127 && jumpWidth > -128 ? "EB " +
IOLib.toHex(jumpWidth, 2) + " 90" : "E9 " + IOLib.toHex(jumpWidth - 1, 4);
            }
            default: //CLI                return "FA";        }    }
```

```
    /**
     * Возвращает машинное представление префикса замены сегмента
     *
     * @param reg Сегментный регистр
     * @return Префикс замены сегмента
     */
    private String getSegPrefix(String reg) {
        switch (reg.toUpperCase()) {
            case "ES":
                return "26: ";
            case "CS":
                return "2E: ";
            case "SS":
                return "36: ";
            case "DS":
                return "3E: ";
            case "FS":
                return "64: ";
            case "GS":
                return "65: ";
            default:
                return "ERROR";
        }
    }
}
```

Translator.java

```
package trasm;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import trasm.LineInfo.LineType;

/**
 * Главный класс транслятора
 */
class Transtalor {

    /**
     * Глобальный флаг первого/второго прохода
     */
    static boolean isSecondPass = false;

    /**
     * Контейнер для хренения информации про строчку листинга
     */
    private static class LstLine {

        int address, lineNum;
        LineInfo info;
```

```
        public LstLine(int address, int lineNum, LineInfo info) {
            this.address = address;
            this.lineNum = lineNum;
            this.info = info;
        }

        @Override
        public String toString() {
            if (info.type == LineType.ASSUME) {
                return "         " + info.toString();
            }
            return String.format("%1$3d", lineNum) + " " +
IOLib.toHex(address, 4) + "    " + info.toString();
        }
    }

    /**
     * Генерирует файл листинга
     *
     * @param asmFilePath Пусть к исходному файлу
     * @param lstFilePath Пусть для файла листинга
     * @param options Дополнительные опции генерации
     * @throws IOException
     */
    private static void makeLST(String asmFilePath, String lstFilePath,
String options) throws IOException {

        boolean firstPassOut = options.contains("f");
        boolean lexicalOut = options.contains("l");
        boolean assumeOut = options.contains("a");
        boolean consoleOut = options.contains("c");

        if (!asmFilePath.toLowerCase().contains(".asm") &&
!asmFilePath.contains(".")) {
            asmFilePath += ".asm";
        }

        if (!lstFilePath.toLowerCase().contains(".lst")) {
            lstFilePath += ".lst";
        }

        ArrayList<LstLine> allLines = new ArrayList<>();
        String[] fileLines = IOLib.readAllLines(asmFilePath);
        ArrayList<LstLine> jumps = new ArrayList<>();
        for (String source_line : fileLines) {
            LineInfo line = new LineInfo(source_line);
            allLines.add(new
LstLine(SegTable.getInstance().getCurrentAddress(), ErrorList.currentLine,
line));
            if (!line.isCorrect()) {
                ErrorList.AddError();
            }
            if (line.type == LineType.JUMP && line.isCorrect()) {
                jumps.add(new
LstLine(SegTable.getInstance().getCurrentAddress(), ErrorList.currentLine,
line));
```

```
            }


SegTable.getInstance().setCurrentAddress(SegTable.getInstance().getCurrentAdd
ress() + line.sizeInBytes);
            ErrorList.currentLine++;
        }

        if (firstPassOut) {
            ArrayList<String> listing = new ArrayList<>();
            String firstPass = lstFilePath.toLowerCase().replace(".lst",
".flst");
            for (LstLine lstLine : allLines) {
                if (lstLine.info.type == LineType.EMPTY) {
                    listing.add("");
                } else if (!lstLine.info.isCorrect()) {
                    listing.add("Синтаксична помилка! : " +
lstLine.toString());
                } else {
                    listing.add(lstLine.toString());
                }
            }
            IOLib.writeAllLines(listing.toArray(new String[listing.size()]),
new PrintStream(new File(firstPass)));
        }

        isSecondPass = true;
        for (LstLine lstLine : jumps) {
            LineInfo line = new LineInfo(lstLine.info);
            allLines.set(lstLine.lineNum - 1, new LstLine(lstLine.address,
lstLine.lineNum, line));
            if (!line.isCorrect()) {
                ErrorList.AddError(lstLine.lineNum);
            }
        }

        ArrayList<String> listing = new ArrayList<>();
        for (LstLine lstLine : allLines) {
            if (lstLine.info.type == LineType.EMPTY) {
                listing.add("");
            } else if (!lstLine.info.isCorrect()) {
                listing.add("Синтаксична помилка! : " + lstLine.toString());
            } else {

                listing.add(lstLine.toString());
                if (assumeOut && lstLine.info.type == LineType.ASSUME) {
                    listing.add(SegTable.getInstance().assumeToString());
                }
            }
        }

        listing.add("\n" + SegTable.getInstance().toString());
        listing.add(IdTable.getInstance().toString());
        listing.add(ErrorList.getStringToPrint());

        if (consoleOut) {
```

```
            IOLib.writeAllLines(listing.toArray(new String[listing.size()]),
System.out);
        }
        IOLib.writeAllLines(listing.toArray(new String[listing.size()]), new
PrintStream(new File(lstFilePath)));

        System.out.println("Вхідний файл: " + asmFilePath + "\nВихідний файл:
" + lstFilePath);

        if (firstPassOut) {
            String firstPass = lstFilePath.toLowerCase().replace(".lst",
".flst");
            System.out.println("Файл першого проходу: " + firstPass);
        }

        if (lexicalOut) {
            String lexemes = lstFilePath.toLowerCase().replace(".lst",
".lex");
            try (PrintStream writer = new PrintStream(new File(lexemes))) {
                for (String source_line : fileLines) {
                    if (source_line.replaceAll(";.*", "").trim().isEmpty()) {
                        continue;
                    }

writer.println(LexicalAnalyzer.getStringToPrint(LexicalAnalyzer.getLexemeInfo
(source_line)));
                }
            }
            System.out.println("Файл лексичного аналізу: " + lexemes);
        }
        System.out.println(ErrorList.getStringToPrint());
    }

    /**
     * Описание работы программы
     */
    private static void showHelp() {
        System.out.println("Використання: trasm [asmFile] [lstFile] [-
options]");
        System.out.println("Довідка: ");
        System.out.println("[asmFile] - шлях до файлу з початковим кодом
мовою ассемблер");
        System.out.println("[lstFile] - шлях до вихідного файлу лістингу");
        System.out.println("[-options] - додаткові опції виконання
программи:");
        System.out.println("    -f - генерація файлу першого проходу
[lstFile].flst");
        System.out.println("    -l - генерація файлу лексичного аналізу за
шляхом [lstFile].lex");
        System.out.println("    -a - виведення інформації(у файлі лістингу)
про Assume");
        System.out.println("    -c - виведення лістингу на екран");
        System.out.println("\nПриклад: trasm source out");
        System.out.println("trasm src.asm out.lst -c");
        System.out.println("trasm test.asm test -af");
    }
```

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

        if (args.length == 0 || args.length > 3) {
            showHelp();
            return;
        }

        try {
            if (args.length == 3) {
                if (!args[2].matches("^-c?l?a?f?$")) {
                    System.out.println("Помилкові опції");
                    showHelp();
                    return;
                }
            }
            makeLST(args[0], args[1], args.length == 2 ? "" : args[2]);
        } catch (FileNotFoundException ex) {
            System.out.println("Файл не знайдено.");
        } catch (IOException ex) {
            System.out.println("Помилка виводу.");
        }
    }
}
```

ДОДАТОК 2
Тестові приклади

1. Тестовий приклад без помилок.

```
Data1 segment
    dbVar1       db 10010011b
    dwVar2       dw 0ABCh
    ddVar3       dd 10101010
Data1 ends

Data2 segment
    STR4  db 'Hello!'
    ddVar5       dd 10101010
Data2 ends

assume ds:Data1, cs:Code, es:Data2

Code segment
begin:
    tmp db 55h
    Cli
    Inc STR4[bx]
    Dec al
    Dec ebx
    Add cs:dbVar1[si], 00010001b
    Cmp bx, dwVar2[eax]
    Xor tmp[ebp], cl
    Mov ah, 128
    Or esi, eax

    jb labelJB

labelUP:
    jmp labelDW

labelJB:
    jmp labelUP
    jb begin

labelDW:

Code ends
end begin
```

2. Тестовий приклад з типовими помилками.

```
Data1 segment
    dbVar1        db 0FFFh
    dwVar2        dw 0ABCh
    ddVar3        dd 10101010
Data1 ends

Data2 segment
    STR4   dd 'Hello!'
    ddVar5        dd 10101010
Data2 ends

assume ds:Data1, cs:Code, es:Data2

Code segment
begin:
    tmp db 55h
    Cli
    Inc test[bx]
    Dec al
    Dec ebx
    Add cs:dbVar1[si], al
    Cmp bx, dwVar2[ax]
    Xor tmp[ebp], cl
    Mov ah, 1025
    Or esi, eax

    jb labelJB

labelUP:
    jmp label

labelJB:
    jmp labelUP
    jb begin

labelDW:

Code ends
end begin
```

ДОДАТОК 3

Файли лістингу, згенеровані транслятором

1. Тестовий приклад без помилок.

```
Курсова робота студента КПІ ФПМ групи КВ-23 Чугаєвського Максима
Варіант 1
Згенеровано: 12/46/2014 00:47:03
 1 0000                         Data1 segment
 2 0000    93                        dbVar1     db 10010011b
 3 0001    0ABC                      dwVar2     dw 0ABCh
 4 0003    009A2112                  ddVar3     dd 10101010
 5 0007                         Data1 ends

 7 0000                         Data2 segment
 8 0000    48 65 6C 6C 6F 21         STR4  db 'Hello!'
 9 0006    009A2112                  ddVar5     dd 10101010
10 000A                         Data2 ends

                              assume ds:Data1, cs:Code, es:Data2

14 0000                         Code segment
15 0000                         begin:
16 0000    55                        tmp db 55h
17 0001    FA                        Cli
18 0002    26: FE 87 0000            Inc STR4[bx]
19 0007    FE C8                     Dec al
20 0009    67| 4B                    Dec ebx
21 000B    2E: 80 84 0000 11         Add cs:dbVar1[si], 00010001b
22 0011    67| 3B 98 00000001        Cmp bx, dwVar2[eax]
23 0018    2E: 67| 30 8D 00000000    Xor tmp[ebp], cl
24 0020    B4 80                     Mov ah, 128
25 0022    66| 0B F0                 Or esi, eax

27 0025    72 05 90 90               jb labelJB

29 0029                         labelUP:
30 0029    EB 05 90                  jmp labelDW

32 002C                         labelJB:
33 002C    EB FB                     jmp labelUP
34 002E    72 D0                     jb begin

36 0030                         labelDW:

38 0030                         Code ends
39 0030                         end begin
```

```
Сегмент  Розмір
Data1    0007
Data2    000A
Code     0030

Ім'я     Тип      Адреса
dbVar1   DB       Data1:0000
dwVar2   DW       Data1:0001
ddVar3   DD       Data1:0003
STR4     DB       Data2:0000
ddVar5   DD       Data2:0006
begin    LABEL    Code:0000
tmp      DB       Code:0000
labelUP  LABEL    Code:0029
labelJB  LABEL    Code:002C
labelDW  LABEL    Code:0030

Помилки: 0
```

2. Тестовий приклад з типовими помилками.

```
Курсова робота студента КПІ ФПМ групи КВ-23 Чугаєвського Максима
Варіант 1
Згенеровано: 12/46/2014 00:49:23
  1 0000                          Data1 segment
Синтаксична помилка! :   2 0000                                 dbVar1
      db 0FFFh
  3 0000    0ABC                        dwVar2     dw 0ABCh
  4 0002    009A2112                    ddVar3     dd 10101010
  5 0006                          Data1 ends

  7 0000                          Data2 segment
Синтаксична помилка! :   8 0000                                 STR4  dd
'Hello!'
  9 0000    009A2112                    ddVar5     dd 10101010
 10 0004                          Data2 ends

                                assume ds:Data1, cs:Code, es:Data2

 14 0000                          Code segment
 15 0000                          begin:
 16 0000    55                          tmp db 55h
 17 0001    FA                          Cli
Синтаксична помилка! :  18 0002                                 Inc
test[bx]
 19 0002    FE C8                       Dec al
 20 0004    67| 4B                     Dec ebx
Синтаксична помилка! :  21 0006                                 Add
cs:dbVar1[si], al
Синтаксична помилка! :  22 0006                                 Cmp     bx,
dwVar2[ax]
 23 0006    2E: 67| 30 8D 00000000     Xor tmp[ebp], cl
Синтаксична помилка! :  24 000E                                 Mov     ah,
1025
 25 000E    66| 0B F0                   Or esi, eax

 27 0011    72 05 90 90                 jb labelJB

 29 0015                          labelUP:
Синтаксична помилка! :  30 0015                                 jmp label

 32 0018                          labelJB:
 33 0018    EB FB                       jmp labelUP
 34 001A    72 E4                       jb begin
```

```
 36 001C                          labelDW:

 38 001C                          Code ends
 39 001C                          end begin

Сегмент  Розмір
Data1    0006
Data2    0004
Code     001C

Ім'я     Тип      Адреса
dbVar1   DB       Data1:0000
dwVar2   DW       Data1:0000
ddVar3   DD       Data1:0002
ddVar5   DD       Data2:0000
begin    LABEL    Code:0000
tmp      DB       Code:0000
labelUP  LABEL    Code:0015
labelJB  LABEL    Code:0018
labelDW  LABEL    Code:001C

Помилки: 7
Рядки з помилками: 2 8 18 21 22 24 30
```

ДОДАТОК 4

Файли лістингу, згенеровані *TASM*

1. Тестовий приклад без помилок.

```
Turbo Assembler  Version 4.0        05/09/14 23:13:04     Page 1
ttest.ASM
      1                         .386
      2        0000             Data1 segment use16
      3        0000  93                    dbVar1     db 10010011b
      4        0001  0ABC                  dwVar2     dw 0ABCh
      5        0003  009A2112              ddVar3     dd 10101010
      6        0007             Data1 ends
      7
      8        0000             Data2 segment use16
      9        0000  48 65  6C 6C 6F 21          STR4  db 'Hello!'
     10        0006  009A2112              ddVar5     dd 10101010
     11        000A             Data2 ends
     12
     13                         assume    ds:Data1, cs:Code, es:Data2
     14
     15        0000             Code segment use16
     16        0000             begin:
     17        0000  55                    tmp db    55h
     18        0001  FA                    Cli
     19        0002  26: FE 87 0000r       Inc STR4[bx]
     20        0007  FE C8                 Dec al
     21        0009  66| 4B            Dec ebx
     22        000B  2E: 80 84 0000r 11          Add    cs:dbVar1[si],
00010001b
     23        0011  67| 3B 98 00000001r         Cmp bx, dwVar2[eax]
     24        0018  2E: 67| 30 8D          +          Xor    tmp[ebp],
cl
     25           00000000r
     26        0020  B4 80                 Mov ah, 128
     27        0022  66| 0B F0             Or esi, eax
     28
     29        0025  72 05  90 90          jb labelJB
     30
     31        0029             labelUP:
     32        0029  EB 05  90             jmp labelDW
     33
     34        002C             labelJB:
     35        002C  EB FB                 jmp labelUP
     36        002E  72 D0                 jb begin
     37
     38        0030             labelDW:
```

```
     39
     40        0030              Code ends
     41                          end begin

Turbo Assembler  Version 4.0        05/09/14 23:13:04     Page 2
Symbol Table


Symbol Name           Type   Value                  Cref    (defined
at #)

??DATE                Text   "05/09/14"
??FILENAME       Text   "ttest    "
??TIME                Text   "23:13:04"
??VERSION        Number 0400
@CPU             Text   0F0FH                  #1
@CURSEG               Text   CODE                   #2  #8  #15
@FILENAME        Text   TTEST
@WORDSIZE        Text   2                 #1  #2  #8  #15
BEGIN            Near   CODE:0000        #16     36  41
DBVAR1                Byte   DATA1:0000             #3  22
DDVAR3                Dword  DATA1:0003             #5
DDVAR5                Dword  DATA2:0006             #10
DWVAR2                Word   DATA1:0001             #4  23
LABELDW               Near   CODE:0030         32  #38
LABELJB               Near   CODE:002C         29  #34
LABELUP               Near   CODE:0029         #31     35
STR4             Byte   DATA2:0000             #9  19
TMP              Byte   CODE:0000         #17     24

Groups & Segments     Bit Size Align  Combine    Class    Cref
     (defined at #)

CODE             16  0030 Para   none          13  #15
DATA1            16  0007 Para   none          #2  13
DATA2            16  000A Para   none          #8  13
```

2. Тестовий приклад з типовими помилками.

```
Turbo Assembler  Version 4.0        05/09/14 22:57:36     Page 1
tTest2.ASM
      1                             .386
      2        0000                 Data1 segment use16
      3        0000  00                        dbVar1     db 0FFFh
**Error** tTest2.ASM(3)    Value out of range
      4        0001  0ABC                      dwVar2     dw 0ABCh
      5        0003  009A2112                  ddVar3     dd 10101010
      6        0007                 Data1 ends
      7
      8        0000                 Data2 segment use16
      9        0000  00000000                  STR4  dd 'Hello!'
**Error** tTest2.ASM(9)    Value out of range
     10        0004  009A2112                  ddVar5     dd 10101010
     11        0008                 Data2 ends
     12
     13                             assume    ds:Data1, cs:Code, es:Data2
     14
     15        0000                 Code segment use16
     16        0000                 begin:
     17        0000  55                        tmp db    55h
     18        0001  FA                        Cli
     19        0002  FF 80  0000               Inc test[bx]
**Error** tTest2.ASM(19) Undefined symbol: TEST
*Warning* tTest2.ASM(19) Argument needs     type override
     20        0006  FE C8                     Dec al
     21        0008  66| 4B             Dec ebx
     22        000A  2E: 00 84 0000r           Add cs:dbVar1[si], al
     23        000F  3B                        Cmp bx, dwVar2[ax]
**Error** tTest2.ASM(23) Illegal indexing mode
     24        0010  2E: 67| 30 8D           +           Xor    tmp[ebp],
cl
     25           00000000r
     26        0018  B4 00                     Mov ah, 1025
**Error** tTest2.ASM(25) Constant too large
     27        001A  66| 0B F0                Or esi, eax
     28
     29        001D  72 02  90 90              jb labelJB
     30
     31        0021                 labelUP:
     32        0021                       jmp label
*Warning* tTest2.ASM(31) Reserved word used as symbol: JMP
     33
     34        0021                 labelJB:
```

```
     35        0021  EB FE                   jmp labelUP
     36        0023  72 DB                   jb begin
     37
     38        0025              labelDW:
     39
     40        0025              Code ends
     41                          end begin

Turbo Assembler  Version 4.0        05/09/14 22:57:36     Page 2
Symbol Table

Symbol Name           Type   Value                  Cref    (defined
at #)

??DATE                Text   "05/09/14"
??FILENAME       Text   "tTest2   "
??TIME                Text   "22:57:36"
??VERSION        Number 0400
@CPU             Text   0F0FH                  #1
@CURSEG               Text   CODE                   #2  #8  #15
@FILENAME        Text   TTEST2
@WORDSIZE        Text   2                 #1  #2  #8  #15
BEGIN            Near   CODE:0000        #16     36  41
DBVAR1                Byte   DATA1:0000            #3  22
DDVAR3                Dword  DATA1:0003            #5
DDVAR5                Dword  DATA2:0004            #10
DWVAR2                Word   DATA1:0001            #4  23
JMP              Word   CODE:0021        #32
LABELDW               Near   CODE:0025        #38
LABELJB               Near   CODE:0021        29  #34
LABELUP               Near   CODE:0021        #31     35
STR4             Dword  DATA2:0000            #9
TMP              Byte   CODE:0000        #17     24

Groups & Segments     Bit Size Align  Combine    Class    Cref
      (defined at #)

CODE             16  0025 Para   none          13  #15
DATA1            16  0007 Para   none          #2  13
DATA2            16  0008 Para   none          #8  13
```

Error Summary

```
**Error** tTest2.ASM(3)    Value out of range
**Error** tTest2.ASM(9)    Value out of range
**Error** tTest2.ASM(19) Undefined symbol: TEST
*Warning* tTest2.ASM(19) Argument needs     type override
**Error** tTest2.ASM(23) Illegal indexing mode
**Error** tTest2.ASM(25) Constant too large
*Warning* tTest2.ASM(31) Reserved word used as symbol: JMP
```

Список використаної літератури

1. В. И. Юров. «Assembler» 2003г. – 636 с.
2. Герберт Шилдт. «Java руководство для начинающих» 2014г. – 619 с.
3. Методичні вказівки з написання транслятора програм на мові асемблера.