Содержание

ВВЕДЕНИЕ

Транслятором называется программа, преобразующая программу с языка программирования, понятного человеку, в машинные коды (файлы типа *.com и *.exe для Windows и MSDos). Отладчик имеет обратную задачу – перевод машинных кодов на язык команд, понятный человеку и последовательное выполнение этих команд с возможностью наблюдения изменения данных[1].

Так как транслятор с языка Ассемблер предполагает запись исходной программы в наиболее близкой к машинным кодам форме. Трансляция программы включает в себя проверку синтаксической правильности исходного текста, т.е. подразумевается лексический и синтаксический анализ, а также представление исходной программы в виде кода, который мог бы, после загрузки в память, непосредственно использоваться микропроцессором. Команды здесь представляют собой мнемокоды, а операндами являются регистры и оперативная память (или ссылки (метки) на какую-то её область). То для разработки транслятора для языка Ассемблер, от программиста требуется знание организации всей системы компьютера: архитектуры, памяти, адресации и т.д.

Следовательно, главной задачей данной курсовой работы является приобретение практических навыков в разработке транслятора языка Ассемблер IBM PC (система команд совместимая с процессором Intel – 8086/8088).

1. Постановка задачи

Необходимо разработать транслятор, реализующий обработку заданного подмножества команд микропроцессора i8088 семейства IBM PC. Форматы кодирования команд должны соответствовать форматам команд микропроцессора i8086/i8088. Транслятор должен выявлять ошибки в исходной программе, распределять память и переводить в машинный язык команды и константы, формировать объектный код программы и протокол трансляции.

Входными данными является текст программы, размещённый в файле с расширением '.asm'. Выходные данные должны находиться в бинарном файле '.bin' — объектный код программы и в текстовых файлах '.lst' — листинг, '.map' — карта.

Текст исходной транслируемой программы должен иметь следующий формат:

NAMESEG SEGMNET
ORG выражение
…….
область команд
…….
NAMEDATA DB
…….
область данных
INT 20H
NAMESEG ENDS
END

Программа должна обрабатывать следующие наборы команд:

1. MOV Р, Р
2. MOV Р, НО
3. MOV ОП (би), Р
4. MOV Р, ОП (би)
5. XCHG Р, Р
6. XCHG ОП (би), Р
7. XCHG Р, ОП (би)
8. IMUL Р
9. LOOP метка
10. INT 20H

Где Р — регистр, НО — непосредственно операнд, ОП — оперативная память, би — базово-индексный режим адресации

Также предусмотреть обработку следующих директив: SEGMENT, ENDS, END, ORG, OFFSET, DB, DW.

Протокол трансляции представляется файлом-листингом, в котором содержится информация о результатах трансляции: номер сроки, адрес команды, шестнадцатеричный и мнемонический коды команд, а также информация об ошибках при их наличии.

Объектный код программы представляет собой файл, в котором содержатся шестнадцатеричные константы в следующем формате:

H <запись-заголовок> T <запись-тело> … E <запись-конец>

где H, T и E — признаки соответствующих записей

1. *<Запись-заголовок>* должна быть самой первой записью в объектном файле и иметь следующий формат:

<Длина имени>	<Имя сегмента>	<Длина сегмента>

<Длина имени> - целое число (байт), указывающее длину поля <Имя сегмента>

<Имя сегмента> - строковая константа размером <Длина имени> байт, указывающая имя сегмента кода

<Длина сегмента> - целое число (слово), указывающее длину сегмента кода

Не допускается наличие нескольких H-записей в одном файле.

2. *<Запись-тело>* имеет следующий формат

<Смещение кода>	<Длина кода>	<Код>

<Смещение кода> - 16-битное смещение <Кода> относительно начала сегмента

<Длина кода> - целое число (слово), указывающее длину поля <Код> данной записи

<Код> - массив размером в <Длину кода> байт, содержащий непосредственно объектный код

3. <Запись-конец> имеет следующий формат

<Точка входа>

<Точка входа> - адрес первой выполняемой команды

Так же транслятор должен обрабатывать следующие ошибки:

1. Неправильность задания команд, констант или операндов.
2. Переполнение таблиц символов.
3. Неизвестная операция.

4. Отсутствие входного файла.
5. Исходный файл пуст.
6. Переменная или метка определена дважды.
7. Отсутствует переменная или метка на которую ссылаются.

2. Определение форматов команд

Форматы кодирования команд соответствуют форматам команд микропроцессора Intel 8086/8088. Длина команд данной курсовой работы составляет от 2 до 4 байт. Код команды занимает первый байт, способ адресации задаётся во втором байте, в остальных байтах указываются данные, либо адрес памяти, где они находятся.

Структура адресного байта имеет вид, представленный на рисунке 2.1

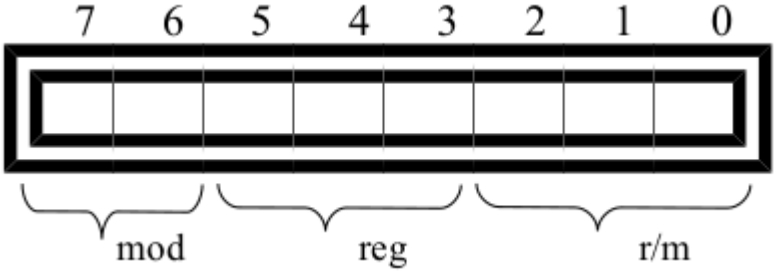

Рисунок 2.1 — Структура адресного байта

Таблица соответствий кодов и мнемонических имён регистров представлена в таблице 2.1

Таблица 2.1 — Соответствия кодов и мнемонических имён регистров

reg или r/m		000	001	010	011	100	101	110	111
Регистр	w = 0	AL	CL	DL	BL	AH	CH	DH	BH
	w = 1	AX	CX	DX	BX	SP	BP	SI	DI

Значения битов mod задают различные способы адресации и имеют следующую интерпретацию:

- mod = 00 — биты r/m задают абсолютный адрес, байт смещения отсуствует
- mod = 01 — биты r/m задают абсолютный адрес памяти и имеется один байт смещения
- mod = 10 — биты r/m задают абсолютный адрес памяти и меется два байта смещения
- mod = 11 — биты r/m и биты reg вместе с битом w определяют конкретные регистры

Кодирование типов адресаций приведено в таблице 2.2

Таблица 2.2 — Коды адресации

r/m	mod = 00	mod = 01 или mod = 10
000	[BX] + [SI]	[BX] + [SI] + смещение
001	[BX] + [DI]	[BX] + [DI] + смещение
010	[BP] + [SI]	[BP] + [SI] + смещение
011	[BP] + [DI]	[BP] + [DI] + смещение
100	[SI]	[SI] + смещение
101	[DI]	[DI] + смещение
110	прямая	[BP] + смещение
111	[BX]	[BX] + смещение

Так как в программе должна быть реализована только базово-индексная адресация, то из таблицы кодов адресации для данной курсовой работы характерны лишь 4 первые строки.

Таблица форматов команд приведена в таблице 2.3.

В таблице используются следующие сокращения:

- w — бит, определяющий длину операндов команды
- Data — байты с данными
- Disp — смещение в ОП
- d — бит, указывающий адресацию операндов
- КОП — код операции
- Рег8, Рег16 — однобайтный и двухбайтный регистры соответственно
- ОП — оперативная память
- НО — непосредственно операнд

Таблица 2.3 — Форматы команд

Команда:	**Оп1: (x)**	**Оп2: (y)**	**1й байт КОП:**			**2й байт адресный**			**3й байт**	**4й байт**
			КОП	**d**	**w**	**mod**	**reg**	**r/m**		
MOV	Per8	Per8	100010	1	0	11	xxx	yyy		
MOV	Per16	Per16	100010	1	1	11	xxx	yyy		
MOV	Per8	Per8	100010	0	0	11	yyy	xxx		
MOV	Per16	Per16	100010	0	1	11	yyy	xxx		
MOV	Per8	ОП	100010	1	0	10	xxx	11z	Disp16	
MOV	Per16	ОП	100010	1	1	10	xxx	11z	Disp16	
MOV	ОП	Per8	100010	0	0	10	yyy	11z	Disp16	
MOV	ОП	Per16	100010	0	1	10	yyy	11z	Disp16	
MOV	Per8	НО	110001	1	0	11	000	xxx	Data8	
MOV	Per16	НО	110001	1	1	11	000	xxx	LL Data16	HH
MOV	Per	НО	10110xxx			Data8				
MOV	Per	НО	10111xxx			LL Data16			HH	
XCHG	Per8	Per8	100001	1	0	11	xxx	yyy		
XCHG	Per16	Per16	100001	1	1	11	xxx	yyy		
XCHG	Per8	ОП	100001	1	0	10	xxx	11z	Disp16	
XCHG	ОП	Per8	100001	1	0	10	yyy	11z	Disp16	
XCHG	Per16	ОП	100001	1	1	10	xxx	11z	Disp16	
XCHG	ОП	Per16	100001	1	1	10	yyy	11z	Disp16	
IMUL	Per8	-	111101	1	0	11	101	xxx		
IMUL	Per16	-	111101	1	1	11	101	xxx		
LOOP	ссылка	-	11100010			Disp8				
INT	20H	-	11001101			20H				

3. Разработка метода решения

Для решения поставленной задачи необходимо разработать на языке высокого уровня программу, переводящую текст на языке Assembler в машинные коды. При этом данная программа должна осуществлять лексический анализ входного текста и генерировать два файла: *.bin (файл объектного кода) и *.lst (файл листинга) в соответствии с постановкой задачи.

За один просмотр далеко не всегда можно сгенерировать код (например, когда есть обращение к метке, которая далеко впереди), поэтому в курсовом проекте реализуем двухпросмотровый ассемблер. При первом просмотре ассемблер просматривает исходную программу и строит таблицу идентификаторов, т.е. имён полей данных и меток, используемых в программе, и их относительных адресов. Кроме того, при первом проходе подсчитывается длина объектного кода, но сам объектный код не генерируется. При втором просмотре, ассемблер, используя таблицу идентификаторов, генерирует объектный код для каждой команды.

Достоинствами двухпроходного транслятора является экономия памяти. Недостатки - один и тот же файл необходимо просмотреть и проанализировать целых два раза, что снижает быстродействие.

Так как основную сложность представляет собой реализация перевода мнемокода инструкции в машинный код, разработаем таблицы, которые будут содержать информацию о реализуемых командах, регистрах, метках и переменных. Для лучшей проверки структуры программы, будет разработана таблица определения фрагмента программы, где для каждого фрагмента программы будут определены инструкции, которые могут встречаться только в этом фрагменте.

Структурная схема транслятора и назначение основных её блоков приведены на рисунке 3.1.

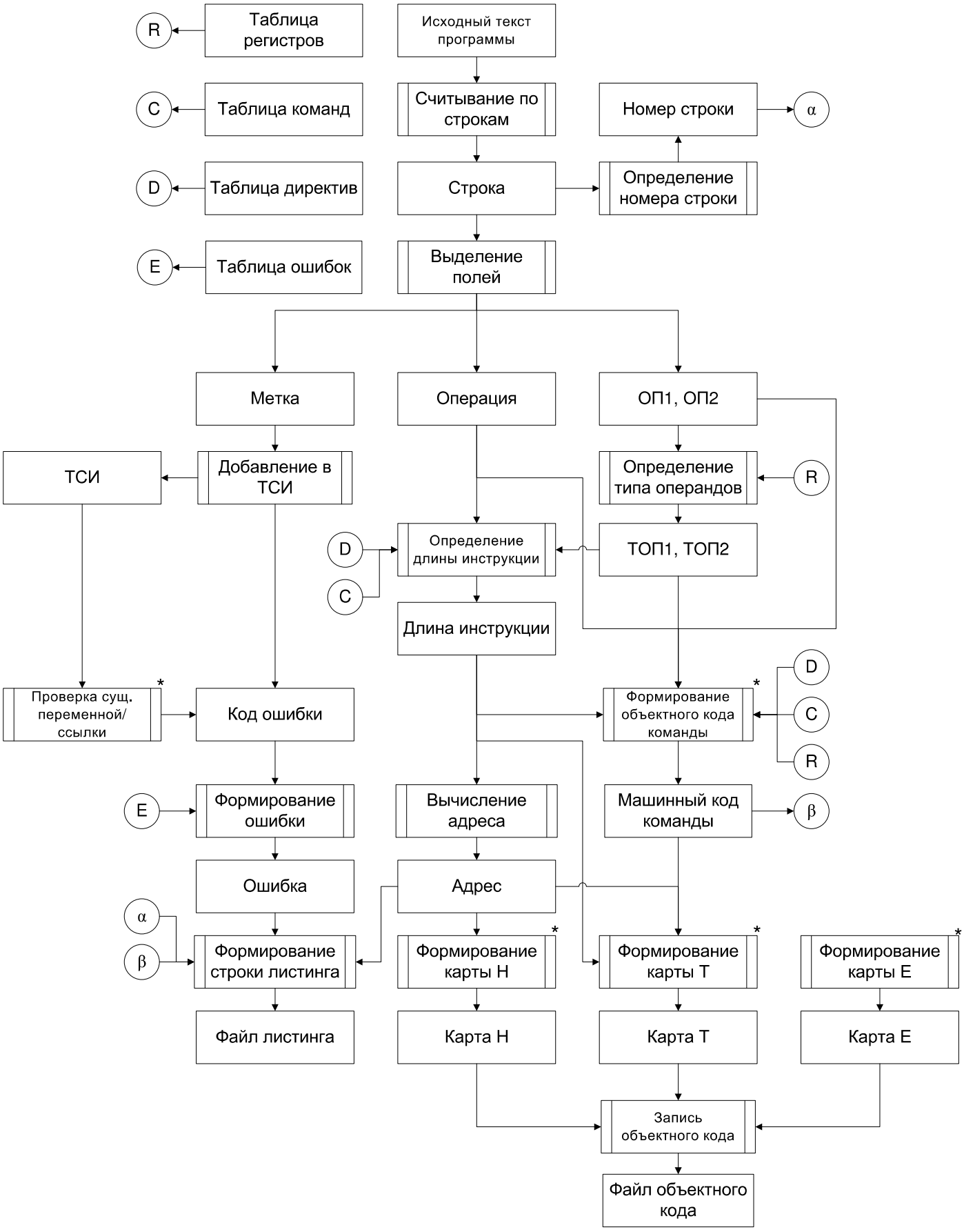

Рисунок 3.1 — Структурная схема транслятора

Карта H, формируется в конце первого просмотра, блоки отмеченные «*» будут выполняться на втором просмотре.

Основные блоки структурной схемы:

- Asm-файл – содержит текст программы на языке Ассемблер.
- Выделение полей – обработка входной программы путём выделения в каждой строке мнемокодов метки, операции, первого и второго операнда.
- Таблица символических имён – содержит имена переменных и меток, которые были найдены в программе, и соответствующие им адреса.
- Таблица директив – содержит список директив с соответствующими каждой директиве мнемоническим именем, форматом операндов и длиной.
- Таблица команд – содержит список команд с соответствующими каждой команде мнемоническим именем, количеством и форматом операндов, длиной и кодом операции.
- Таблица регистров – содержит список регистров с соответствующими каждому регистру мнемоническим именем, типом регистра и его кодом.
- Таблица ошибок – содержит заготовленные сообщения на определённый код ошибки.
- Файл Lst – конечный файл, содержащий листинг программы.
- Файл Obj – конечный файл, содержащий объектный код.

4. Выбор и обоснование структур данных

При разработке программного обеспечения большую роль играет проектирование хранилища данных, и представление всех данных в виде множества связанных структур данных. Хорошо спроектированное хранилище данных оптимизирует использование ресурсов (таких как время выполнения операций, используемый объём оперативной памяти, число обращений к дисковым накопителям), требуемых для выполнения наиболее критичных операций.

Таблица регистров, таблица директив и таблица команд будут представлять собой массивы записей со следующими полями-константами:

1. Таблица команд:
   - мнемоническое имя команды – строковый тип;
   - количество операндов команды – целый тип;
   - форматы операндов команды – массив строкового типа;
   - длина команды – строковый тип;
   - машинный код команды – строковый тип;
2. Таблица регистров:
   - имя регистра – строковый тип;
   - тип регистра (однобайтный или двухбайтный) – строковый тип;
   - машинный код регистра – строковый тип.
3. Таблица директив:
   - имя директивы – строковый тип;
   - длина директивы – строковый тип;
   - форматы операндов – массив строкового типа.

Таблица символических имён имет следующие поля-константы:

- имя метки/переменной – строковый тип;
- адрес метки/переменной – строковый тип;
- тип (метка или переменная) – строковый тип.

Таблица определения позиции в программе будет иметь структуру записи, со следующими полями-константами:

- текущий фрагмент – массив значений целого типа;
- следующий фрагмент – массив значений целого типа;
- команды текущего сегмента – массив значений строкового типа;
- команды следующего сегмента – массив значений строкового типа;

Таблица ошибок будет представлена в виде массива строкового типа, каждый элемент которого будет содержать сообщение на каждый код ошибки.

Исходный файл преобразуется в массив записей, который имеет четыре поля: поле метки, поле команды, поле первого операнда и поле второго операнда.

Для глобальных переменных введены структуры данных следующего типа:

- Счётчик адреса команд – переменная целого типа;
- Длина инструкции – переменная целого типа;
- Список текущих ошибок – переменная строкового типа;
- Код инструкции – переменная строкового типа;
- Длина инструкции – переменная целого типа;
- Флаг ошибок – переменная булевского типа.

5. Алгоритм решения задачи

Так как была выбрана реализация двухпросмотрового транслятора, то необходимо разработать алгоритм для первого и второго просмотра отдельно. Во время второго просмотра, текст ошибки по соответствующему коду ошибки будет записываться не в список ошибок, а сразу в файл листинга вместе с текущей строкой.

Первый просмотр:

1. Открыть файл с текстом исходной программы.
2. Номер строки, СЧАК и счётчик фрагмента установить в ноль.
3. Код ошибки (КО), длину команды и код обработки директивы (КОД) установить в ноль.
4. Чтение очередной строки из файла и увеличение номера строки на 1. Если КО ≠ 0, то перейти к п.12
5. Если конец файла, то прейти к п.13
6. **Выделение полей в исходной строке** и определение типа метки.
7. Если поле метки не пусто, записать метку в ТСИ. Если повторное определение метки, то запись номера строки в таблицу ошибок первого просмотра.
8. Формирование типов операндов. Если КО ≠ 0, то перейти к п.12
9. **Обработка директив:**
   - 9.1. Если КОД = 0, то записать имя сегмента;
   - 9.2. Если КОД = 1, то увеличиваем СЧАК на значение длины команды и переходим к п.3;
   - 9.3. Если КОД = 2, то проверяем соответствует ли имя сегмента имени директивы. Если КО ≠ 0, то перейти к п.12
   - 9.4. Если КОД = 3, то переходим к п.13;
   - 9.5. Если КОД = 4, то записать переменную и ее адрес в ТСИ. Если КО ≠ 0, то перейти к п.12
   - 9.6. Если КОД = 5, то МНК – не директива. Осуществляем поиск МНК в таблице команд. Если КО ≠ 0, то перейти к п.12
   - 9.7. Если КОД = 6, то перейти к п.12
10. Определение длины команды. Если КО ≠ 0, то перейти к п.12
11. Увеличение СЧАК на длину команды.
12. Переход к п.3.
13. Сравниваем текущее значение счётчика фрагмента со значением в таблице директив для END . Если КО ≠ 0, то перейти к п.15
14. Сформировать карту Н.
15. Конец первого просмотра.

Второй просмотр:

1. Открытие файла с текстом исходной программы.
2. Номер строки, СЧАК и счётчик фрагмента установить в ноль.
3. Код ошибки (КО), длину команды и код обработки директивы (КОД) установить в ноль.
4. Чтение очередной строки из файла и увеличение номера строки на 1. Если КО ≠ 0, то перейти к п.13
5. Если конец файла, то прейти к п.18
6. **Выделение полей в исходной строке** и определение типа метки.
7. Формирование типов операндов. Если КО ≠ 0, то перейти к п.13
8. **Обработка директив:**
   - 8.1. Если КОД = 0, то записать имя сегмента;
   - 8.2. Если КОД = 1, то увеличиваем СЧАК на значение, указанное в операнде и переходим к п.17;
   - 8.3. Если КОД = 2, то проверяем соответствует ли имя сегмента имени директивы. Если КО ≠ 0, то перейти к п.13.
   - 8.4. Если КОД = 3, то переходим к п.19
   - 8.5. Если КОД = 5, то МНК – не директива. Осуществляем поиск МНК в таблице команд. Если КО ≠ 0, то перейти к п.13
   - 8.6. Если КОД = 6, то перейти к п.13
9. Определение длины команды. Если КО ≠ 0, то перейти к п.13
10. Формирование кода команды.
11. Сформировать и записать в объектный файл карту Т.
12. Формирование строки протокола.
13. Обращение к формированию и записи ошибок первого просмотра.
14. Обращение к формированию и записи ошибок второго просмотра.
15. Запись строки протокола в файл листинга.
16. Увеличение СЧАК на длину команды.
17. Переход к п.3.
18. Сравниваем текущее значение счётчика фрагмента со значением в таблице директив для END . Если КО ≠ 0, то вывести ошибку неверной структуры программы, установить флаг ошибки=1 и перейти к п.20
19. Формирование и запись карты Е в объектный файл.
20. Если флаг ошибки равен 1, то удалить объектный файл.
21. Конец.

Выделение полей в исходной строке:

1. Убрать комментарий.
2. Разбить строку по пробелам на фрагменты и определить их количество.
3. Если количество фрагментов равно 3 то:
   - *3.1.* В поле *метки* записать первый фрагмент.
   - *3.2.* В поле *МНК* записать второй фрагмент.
   - *3.3.* В поле *операнды* записать третий фрагмент.
4. Если количество фрагментов меньше трёх, то:

4.1. В поле *МНК* записать первый фрагмент.
4.2. В поле *операнды* записать второй фрагмент (если он есть).

5. Если поле *операнды* = SEGMENT или ENDS, то:
5.1. В поле *метки* записать поле *МНК.*
5.2. В поле *МНК* записать поле *операнды.*

6. Выделение в поле *операнды* ОП1 и ОП2 (разбиение поля по запятой).

Обработка директив:

1. Сравниваем извлечённый мнемокод с мнемокодами из таблицы директив. Если такого МНК нет, то сформировать КО и код обработки директив (КОД), равный 1;
2. Значение типа *Метки* (имя директивы) сравнить со значением, указанным в столбце «имя директивы» таблицы директив. Если значение не соответствует, то сформировать КО и КОД, равный 1;
3. Значение ТОП1 сравнить с таблицей директив. Если значение не соответствует, то сформировать КО и КОД, равный 1;
4. Если у директивы есть ОП2, то сформировать КО и КОД, равный 1;
5. Сравниваем текущее значение счётчика фрагмента с соответствующим значением в таблице директив (столбец «номер текущего фрагмента»). Если значение не соответствует, то счётчику фрагментов присвоить значение из столбца «номер следующего фрагмента» и сформировать КО и КОД = 1;
   Иначе счётчику фрагментов присваиваем значение из столбца «номер следующего фрагмента» и присваиваем КОД значение из столбца «код директивы»:
   - SEGMENT КОД = 0;
   - ORG КОД = 1;
   - ENDS КОД = 2;
   - END КОД = 3;
   - DB, DW КОД = 4;

6. Описание программы

Программная модель разработана на языке высокого уровня программирования Java в среде программирования Eclipse Mars.

Реализован графический интерфейс для удобной загрузки, сохранения и трансляции готового файла с мнемокодом программы либо непосредственного ввода в текстовое окно. По окончании трансляции выводится сообщение об успешности выполнения задачи или сообщение об ошибке.

6.1 Назначение программы

Программный продукт может выполнять следующие основные функции:

1. Загрузка готового текста на языке Ассемблер в программную модель.
2. Редактирование с внесением изменений в текст на языке Ассемблера для дальнейшей трансляции.
3. Перевод текста на языке Ассемблер в машинные коды.
4. Вывод файла листинга транслируемой программой и указаним в нем возможных ошибок.

6.2 Требования к программному и техническому обеспечению

Для выполнения программы транслятора и запуска виртуальной машины Java рекомендуемы следующие технические и программные характеристики системы:

- компьютер типа IBM PC с центральным процессором Intel с тактовой частотой не ниже 1.0Гц или совместимым;
- установленные на компьютер библиотеки виртуальной машины Java и сама виртуальная машина;
- 128 Mb оперативной памяти;
- жёсткий диск с объёмом свободного пространства не менее 2 Mb;
- монитор VGA с разрешением не меньше 640x480x16;
- операционная система Microsoft не ниже Windows 95, Unix системы.

6.3 Логическая структура программы

Язык Java объектно-ориентированный, поэтому разумно будет разбить программу на классы и методы.

Логическая структура программы реализована на основе алгоритма трансляции программы в исполнительный файл. Программная модель состоит из интерфейсной (видимой пользователю) и структурной (невидимой пользователю) частей.

Программа состоит из следующих классов:

- class Assembler — класс, в котором реализован графический интерфейс программы
- class Structure — структурный класс, связывает между собой все остальные классы и реализует непосредственно транслятор

- class PositionInProgramm — класс содержит информацию для определения позиции в программе
- class SymbolTable — класс, реализующий таблицу символических имён
- class TableOfCommand — класс хранит данные о командах и реализует методы работы с ними
- class TableOfDirective — класс хранит данные о директивах и реализует методы работы с ними
- class TableOfRegisters — класс хранит данные о регистрах и реализует методы работы с ними
- class Cards — класс, реализующий карты

6.4 Используемые переменные и их типы

Глобальные переменные:

private String SegmentName – имя сегмента.

public boolean PrintCards – переменная, отвечающая за формирование файла карт; true – файл карт будет формироваться, false – файл карт формироваться не будет.

public boolean PrintSymb – переменная, отвечающая за формирование файла таблица символических имен; true – файл таблица символических имен будет формироваться, false – файл таблица символических имен формироваться не будет.

private String Listing – переменная, в которую будет заноситься листинг программы.

private boolean ErrorFlag – флаг ошибок.

private int Schak – счётчик команд.

private String ListOfErrors – переменная, куда будут заноситься ошибки, найденные в программе.

private String Code – переменная, в которой содержится машинный код инструкции в текущей строке.

private TableOfCommand TabOfCom = new TableOfCommand(5) – экземпляр класса таблицы команд. Содержит следующие локальные поля:

- String MnemonicName – мнемокод команды.
- int OperandNumbers – количество операндов.
- String[] OperandsFormat = new String[2] – массив форматов операндов.
- String length – длина команды.
- String code – машинный код команды.

-TableOfCommand[] ComTab = new TableOfCommand[5] – массив записей команд.

private TableOfDirective TabOfDir = new TableOfDirective(4) – экземпляр класса таблицы директив. Содержит следующие локальные поля:

- String DirName – мнемокод директивы.
- String length – длина директивы.

- String[] OperandsFormat = new String[2] – массив форматов операндов.
- TableOfDirective[] DirTab = new TableOfDirective[4] – массив записей директив.
- private TableOfRegisters TabOfReg = new TableOfRegisters(16) – экземпляр класса таблица регистров. Содержит следующие локальные поля:
- String RegName – мнемокод регистра.
- String type – тип регистра (однобайтный или двухбайтный).
- String code – двоичный код регистра.

private ArrayList<SymbolTable> SymbolTab = new ArrayList() - экземпляр класса таблица символических имен. Содержит следующие локальные поля:
- String Name – имя метки или переменной.
- String Offset – адрес метки или переменной.
- String Type – тип (метка или переменная).

private Cards H – экземпляр класса карты. Содержит следующие локальные поля:
- char Signature – символ карты (в данном случае ‘H’).
- String SegName – имя сегмента.
- int AdressOfCode – длину сегмента.

private ArrayList<Cards> T = new ArrayList() – массив экземпляров класса карты команд. Содержит следующие локальные поля:
- char Signature – символ карты (в данном случае ‘T’).
- int AdressOfCode – адрес инструкции.
- int CodeLength – длина кода.
- String Code – сам код.

private Cards E – экземпляр класса карты команд. Содержит следующие локальные поля:
- char Signature – символ карты (в данном случае ‘E’).
- String Enter – точка входа.
- private PositionInProgramm position = new PositionInProgramm();

Массив ошибок:

```
String[] Error = {
        "Ошибка в структуре программы",
        "Несоответствие типов и/или количества операндов",
        "Неизвестная команда (инструкция)",
        "Ссылка на неопределённое имя или метку",
        "Повторное определение метки или переменной",
        "Неверная адресация",
        "Переполнение таблиц символов"};
```

Массив команд и директив:

String[] ComAndDir = { "SEGMENT", "ORG", "DB", "DW", "ENDS", "END", "OFFSET", "MOV", "ADD", "SHR", "JS", "INT" }

Массив типов:

String[] Types = { "Dir", "Dir", "Dir", "Dir", "Dir", "Dir", "Dir", "Com", "Com", "Com", "Com", "Com" };

Локальные переменные:

String Metka – переменная для хранения метки в текущей строке.

String Command – переменная для хранения команды в текущей строке.

String Op1 – переменная для хранения первого операнда в текущей строке.

String Op2 – переменная для хранения второго операнда в текущей строке.

String CurrentLine – текущая строка.

String[] CurrentCommand – массив из текущих команд, т.е. инструкций, которые могут встречаться в данном фрагменте программы.

Op1Kind, Op2Kind – типы первого и второго операнда соответственно.

Result – результат определения типа операнда.

6.5 Спецификация методов

Процедуры и функции, используемые в структурной части:

- public boolean Begin (String MnemonicCode) – функция начала трансляции
  Входные данные: текст программы на языке Ассемблер (MnemonicCode)
  Результат: true – ошибки в процессе трансляции, false – ошибок не возникло
  Переменные и константы:
  - MnemonicCode
  Вызываемые методы:
  - firstPass()
  - secondPass()
  - writeListing()
  - writeObject()
  - writeSymbol()
  - writeCards()
  - writeListing()

- void firstPass (String t) – процедура первого просмотра
  Входные данные: текст программы на языке Ассемблер (t)
  Переменные и константы:
  - MnemonicCodeLetters
  - Text
  - Lines
  - CurrentLine

- CurrentCommand
- NextCommand
- PosFragment
- SegmentName
- position
- Error
- StructureError
- OperandsError
- UnknownCommand
- ReferError
- MultipleError
- AddressingError
- ComAndDir
- Types
- Schak
- CodeLen
- Metka
- Command
- Op1
- Op2
- SymbolTab
- TabOfCom
- OperationType
- Op1Kind
- Op2Kind
- H

Вызываемые методы:
- DeleteComments()
- OurComAndDir()
- OurComAndDir(Letters, CurrentCommand)
- Err()
- FindType()
- OpKind()
- getInstrLen()
- getNextPosSeg()
- getCurrCom()
- getNextCom()

- boolean secondPass (String t) – функция второго просмотра
  Входные данные: текст программы на языке Ассемблер (t)
  Результат: true – ошибки в процессе трансляции, false – ошибок не возникло
  Переменные и константы:
  - MnemonicCode

- Letters
- Text
- Lines
- CurrentLine
- CurrentCommand
- NextCommand
- PosFragment
- SegmentName
- position
- Error
- StructureError
- OperandsError
- UnknownCommand
- ReferError
- MultipleError
- AddressingError
- ComAndDir
- Types
- ErrorFlag
- Listing
- Schak
- ListOfErrors
- CodeLen
- Code
- Metka
- Command
- Op1
- Op2
- SymbolTab
- TabOfCom
- OperationType
- Op1Kind
- Op2Kind
- H
- T
- E

Вызываемые методы:

- DeleteComments()
- OurComAndDir()
- OurComAndDir(Letters, CurrentCommand)
- HaveErr()
- FindType()
- OpKind()
- getInstrLen()

- getNextPosSeg()
- getCurrCom()
- getNextCom()
- getCode()

- void writeObject() – процедура записи машинных команд в объектный код
  Переменные и константы:
  - ObjectCode
  - H
  - Signature
  - SegName
  - AdressOfCode
  - T
  - CodeLength
  - Code
  - E
  - file

- void writeListing() – процедура записи листинга в файл
  Переменные и константы:
  - ListingFile
  - Listing
  - file

- void writeSymbol() – процедура записи таблицы символических имен в файл
  Переменные и константы:
  - SymbolFile
  - Name
  - Offset
  - Type
  - file

- void writeCards() – процедура записи карт в файл
  Переменные и константы:
  - Cardfile
  - H
  - Signature
  - SegName
  - AdressOfCode
  - T
  - CodeLength
  - Code
  - E
  - file

- String getCode(String Command, String Op1, String Op2, String Op1Kind, String Op2Kind, int CodeLen) – функция получения кода инструкции
  Входные данные: инструкция (Command), первый операнд (Op1), второй операнд (Op2), типы первого и второго операнда (Op1Kind, Op2Kind), длина инструкции (CodeLen)
  Результат: машинный код инструкции
  Переменные и константы:
  - CodeLen
  - Code
  - Command
  - Op1
  - Op2
  - SymbolTab
  - TabOfCom
  - TabOfReg
  - OperationType
  - Op1Kind
  - Op2Kind
  - H
  - T
  - E

  Вызываемые методы:
  - HexSchak()
  - from_16_to_10()
  - getRegCode()
  - getCode()

- String OpKind(String Op, int pass) – функция определения типа операнда
  Входные данные: операнд (Op) и номер просмотра (pass)
  Результат: тип операнда
  Переменные и константы:
  - Result
  - Op
  - pass
  - Value
  - TabOfDir
  - TabOfReg

  Вызываемые методы:
  - isVariable()
  - isLabel()
  - DirEn()
  - from_16_to_10 ()
  - isReg()

- getRegType()

- String DeleteComments(String s) – функция для удаления комментария
  Входные данные: строка, из которой будут удалены комментарии (s)
  Результат: строка без комментариев

- boolean DoubleVarOrLabel(String label) – функция для определения повторного определения метки или переменной
  Входные данные: метка или переменная (label)
  Результат: true – метка или переменная уже содержится в таблице символических имён, false – метки или переменной нет в таблице символических имён.
  Переменные и константы:
  - SymbolTab
  - Name

- boolean isVariable(String var) – функция для определения принадлежности передаваемого выражения к переменной из таблицы символических имён
  Входные данные: выражение (var)
  Результат: true – выражение является переменной, false – выражение не является переменной
  Переменные и константы:
  - SymbolTab
  - Name
  - Type

- boolean isLabel(String label) – функция для определения принадлежности передаваемого выражения к метке из таблицы символических имён
  Входные данные: выражение (label)
  Результат: true – выражение является меткой, false – выражение не является меткой.
  Переменные и константы:
  - SymbolTab
  - Name
  - Type

- String getOffset(String varlabel) – функция получения адреса метки или переменной
  Входные данные: метка или переменная (varlabel)
  Результат: адрес метки или переменной
  Переменные и константы:
  - SymbolTab
  - Name
  - Offset

- int getOpNum(String Name) – функция для получения количества операндов у данной инструкции
  Входные данные: мнемоническое имя инструкции (Name)
  Результат: количество операндов
  Переменные и константы:
  - ComTab
  - MnemonicName
  - OperandNumber

- boolean isRightOperands(String Com, String TOp1, String TOp2) – функция для определения факта правильности операндов
  Входные данные: команда (Com), тип первого и второго операндов (TOp1, TOp2)
  Результат: true – инструкции при данных операндах имеет правильный формат, false – инструкция при данных операндах имеет неправильный формат
  Переменные и константы:
  - ComTab
  - MnemonicName
  - OperandsFormat
  - Com
  - TOp1
  - TOp2

- int getInstrLen(String Com, String TOp1, String TOp2) – функция для определения длины инструкции
  Входные данные: команда (Com), тип первого и второго операндов (TOp1, TOp2)
  Результат: длина инструкции
  Переменные и константы:
  - ComTab
  - MnemonicName
  - OperandsFormat
  - Com
  - TOp1
  - TOp2.

- String getCode(String Com, String TOp1, String TOp2) – функция для определения длины инструкции
  Входные данные: команда (Com), тип первого и второго операндов (TOp1, TOp2)
  Результат: машинный код инструкции
  Переменные и константы:

- ComTab
- MnemonicName
- OperandsFormat
- Com
- TOp1
- TOp2
- code

- String getRegType(String r) – функция для получения типа регистра
  Входные данные: имя регистра ®
  Результат: тип регистра
  Переменные и константы:
  - RegTab
  - RegName
  - type

- String getRegCode(String r) – функция для получения двоичного кода регистра
  Входные данные: имя регистра
  Результат: двоичный код регистра
  Переменные и константы:
  - RegTab
  - RegName
  - code

6.6 Сообщения, выдаваемые программой

Программная модель выдаёт следующие сообщения:

а) диалоговые:

- «Трансляция успешно завершена!» – при выполнении трансляции без ошибок;
- «Трансляция завершилась с ошибкой!» – если в процессе трансляции произошла ошибка.

б) содержащиеся в файле листинга:

- «Не указано имя сегмента»;
- «Не указано директива SEGMENT»;
- «Неожиданное окончание сегмента»;
- «Неправильное определение метки»;
- «Между операндами отсутствует запятая»;
- «Нет директивы закрытия сегмента»;
- «Имя переменной не определено»;
- «Имя сегмента при его закрытии указано неправильно»;
- «Не правильная структура входной программы!»
- «Ошибка в структуре программы»;
- «Несоответствие типов и/или количества операндов»;
- «Неизвестная команда (инструкция)»;

- «Ссылка на неопределённое имя или метку»;
- «Повторное определение метки или переменной»;
- «Неверная адресация».

6.7 Вызов и загрузка программы

Программная модель может быть выполнена путём запуска исполнительного файла Assembler.jar. Либо через среду разработки и отладки Eclipse.

Вызывается интерфейсная часть, в тестовом поле которого можно осуществить ввод с клавиатуры и редактирование текста программы на языке Ассемблер или загрузить готовый. При нажатии на кнопку «Транслировать» появится окно выбора директории, в которую необходимо осуществить трансляцию и будет осуществлён непосредственно сам процесс трансляции программы, находящейся в данный момент в текстовом окне «Исходный текст программы». Закрытие программы происходит стандартным способом Windows для фреймов.

Пример окна представлен на рисунке 6.2.

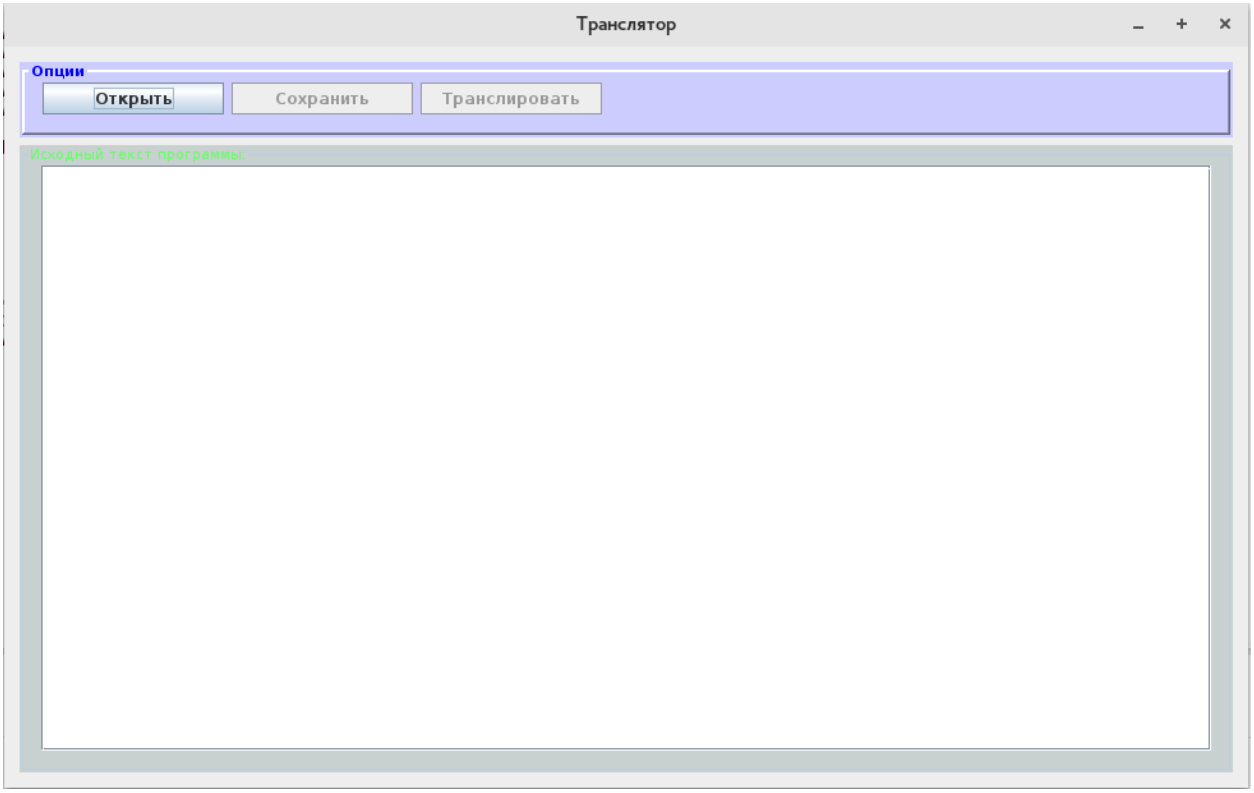

Рисунок 6.2 — Окно программы

7. Тестирование программы

7.1 Разработка тестовых примеров

Тестирование программной модели будем проводить на одном правильном тестовом тексте программы на языке Ассемблера, а также на неверном тексте, в который включим предусмотренные ошибки; результатом работы программы должен быть файл листинга с ошибками.

7.2 Результаты тестирования

Тест №1

Проверка работы программной модели при входных данных в виде текста программы на языке Ассемблер, не содержащего ошибок:

```
Program   SEGMENT
   ORG 100H
   MOV      CH,CL
   MOV      AX, 12345
   MOV      CL,0FFH
   MOV      SI,AX
   MOV      [BX+SI+0FH],DL
   MOV      BP,OFFSET A
   MOV      BX,OFFSET B
   MOV      DL,[BP+DI+56]
   MOV      AX,[BX+SI+123]
   MOV      [BP+SI+4AH],BL
    XCHG     BX,AX
   LOOP      Q
   XCHG      CL,DL
   XCHG      [BP+DI+3EH],CL
Q:  IMUL     AX
  XCHG      SI,[BX+DI+0CH]
   INT 20H
A   DB       80 , 88h
B   DW       0A38H, 0FFH
Program   ENDS
END
```

Результаты теста:

Листинг:

```
1.    :                    Program   SEGMENT
2.    :                     ORG 100H
3. 0100:  8AFA              MOV      CH,CL
4. 0102:  C7C03930          MOV      AX, 12345
```

```
5. 0106:  C6C1FF          MOV     CL,0FFH
6. 0109:  8BFC            MOV     SI,AX
7. 010B:  88900F          MOV     [BX+SI+0FH],DL
8. 010E:  C7C53101        MOV     BP,OFFSET A
9. 0112:  C7C33301        MOV     BX,OFFSET B
10. 0116:  8A9A38         MOV     DL,[BP+DI+56]
11. 0119:  8B807B         MOV     AX,[BX+SI+123]
12. 011C:  889A4A         MOV     [BP+SI+4AH],BL
13. 011F:  87D8           XCHG     BX,AX
14. 0121:  E206           LOOP     Q
15. 0123:  86CA           XCHG     CL,DL
16. 0125:  868B003E       XCHG     [BP+DI+3EH],CL
17. 0129:  F7E8           Q:  IMUL     AX
18. 012B:  87B1000C       XCHG     SI,[BX+DI+0CH]
19. 012F:  CD20           INT 20H
20. 0131:  5088           A   DB      80 , 88h
21. 0133:  380AFF00       B   DW      0A38H, 0FFH
22.    :                  Program   ENDS
23.    :                  END
```

Карты:

```
H PROGRAM 0137
T 0100 02 8AFA
T 0102 04 C7C03930
T 0106 03 C6C1FF
T 0109 02 8BFC
T 010B 03 88900F
T 010E 04 C7C53101
T 0112 04 C7C33301
T 0116 03 8A9A38
T 0119 03 8B807B
T 011C 03 889A4A
T 011F 02 87D8
T 0121 02 E206
T 0123 02 86CA
T 0125 04 868B003E
T 0129 02 F7E8
T 012B 04 87B1000C
T 012F 02 CD20
T 0131 02 5088
T 0133 04 380AFF00
E 0000
```

Объектный код:

```
4850 524f 4752 414d 3701 5401 0002 8afa
5401 0204 c7c0 3930 5401 0603 c6c1 ff54
0109 028b fc54 010b 0388 900f 5401 0e04
c7c5 3101 5401 1204 c7c3 3301 5401 1603
8a9a 3854 0119 038b 807b 5401 1c03 889a
4a54 011f 0287 d854 0121 02e2 0654 0123
0286 ca54 0125 0486 8b00 3e54 0129 02f7
e854 012b 0487 b100 0c54 012f 02cd 2054
0131 0250 8854 0133 0438 0aff 0045 0000
```

Тест №2

Проверка работы программной модели при входных данных в виде текста программы на языке Ассемблер, содержащего ошибки:

```
  SEGMENT
   ORG 100H
   MOV     CH,CL
   MOV     AX, 12345
   MOD     CL,0FFH
   MOV     SI,AX
   MOV     [BX+SI+0FH],DL
   MOV     BP,OFFSET A
   MOV     BX,OFFSET
   MOV     DL
   MOV     AX,[BX+SI+123]
   MOV     [BP+SI+4AH],BL
   ADD     AL,58
 S:  ADD     BX,AX
   Js      Q
   ADD     CL,DL
   ADD     AL,[BP+SI+0EFFH]
   ADD     [BP+DI+3EAFH],CL
S:  ADD     SI,[BX+DI+0CH]
   SHR     AX
   INT 20H
A  DB      80 , 88h
B  DW      0A38H, 0FFH
Program  ENDS
END
```

Результаты теста:

Листинг:

1. Не указано имя сегмента: SEGMENT
2. : ORG 100H

3. 0100: 8AFA MOV CH,CL
4. 0102: C7C03930 MOV AX, 12345
5. Неизвестная команда (инструкция):
MOD CL,0FFH
6. 0106: 8BFC MOV SI,AX
7. 0108: 88900F MOV [BX+SI+0FH],DL
8. 010B: C7C51701 MOV BP,OFFSET A
9. Несоответствие типов и/или количества операндов:
MOV BX,OFFSET
10. Между операндами отсутствует запятая:
MOV DL
11. 010F: 8B807B MOV AX,[BX+SI+123]
12. 0112: 889A4A MOV [BP+SI+4AH],BL
13. Неизвестная команда (инструкция):
ADD AL,58
14. Неизвестная команда (инструкция):
S: ADD BX,AX
15. Неизвестная команда (инструкция):
Js Q
16. Неизвестная команда (инструкция):
ADD CL,DL
17. Неизвестная команда (инструкция):
ADD AL,[BP+SI+0EFFH]
18. Неизвестная команда (инструкция):
ADD [BP+DI+3EAFH],CL
19. Неизвестная команда (инструкция):
S: ADD SI,[BX+DI+0CH]
20. Неизвестная команда (инструкция):
SHR AX
21. 0115: CD20 INT 20H
22. 0117: 5088 A DB 80 , 88h
23. 0119: 380AFF00 B DW 0A38H, 0FFH
24. Имя сегмента при его закрытии указано неправильно:
Program ENDS
25. Нет директивы закрытия сегмента:
END
25. : END

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были изучены форматы команд микропроцессора i8088, разработаны для них структуры данных, структурная схема и сами алгоритмы для первого и второго просмотра трансляции, а также тестовые примеры, проверяющие правильность работы программы. Были получены практические навыки в написании программ трансляции выражений на языке низкого уровня. Программа написана на языке программирования Java в среде Eclipse Mars.

Результаты испытаний показали, что на тестовых примерах программа работает корректно.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Тертычный А.И. Методические указания к выполнению курсовой работы «Разработка транслятора для языка Ассемблер и универсального отладчика-эмулятора» по дисциплине «Системное программирование» для студентов специальности 7.091501 дневной и заочной форм обучения/ А.И. Тертычный, Е.М. Шалимова – Севастополь: издательство СевГТУ, 2001.-20с.
2. 2. Дао Л. Программирование микропроцессора 8088: Пер. с англ. – М.: Мир, 1988. – 357 с. Издательство: Москва «Мир» 1988.
3. Абель П. Язык Ассемблера для IBM PC и программирования: Пер. с англ. – М.: Высшая школа, 1992.-477 с.
4. Донован Дж. Системное программирование / Дж. Донован. –М.: Мир,1975.–540с.
5. Баррон Д. Ассемблеры и загрузчики / Д. Баррон – М.: Мир, 1974. – 74с.

Приложение А. Текст программы

PositionInProgramm.java

```
package com.worm2fed.sp.kursach;
// ********************************************************************** //
// ****************** ТАБЛИЦА ОПРЕДЕЛЕНИЯ ПОЗИЦИИ В ПРОГРАММЕ ************ //
// ********************************************************************** //

public class PositionInProgramm {
	int[] PosInSeg = { 0, 1, 2, 3, 4 }, NextPosSeg = { 1, 2, 3, 4, 4 };
	String[] CurrCom = { "SEGMENT", "ORG,MOV,XCHG,IMUL,LOOP,INT", "DB,DW",
"ENDS", "END" },
			NextCom = { "ORG,MOV,XCHG,IMUL,LOOP,INT", "DB,DW", "ENDS",
"END", "" };

	int getNextPosSeg(int i) {
		return NextPosSeg[i];
	}

	String getCurrCom(int i) {
		return CurrCom[i];
	}

	String getNextCom(int i) {
		return NextCom[i];
	}
}
```

Cards.java

```
package com.worm2fed.sp.kursach;
//************************************************************************ //
//***************************** ТАБЛИЦА КАРТ ***************************** //
//************************************************************************ //

public class Cards {
	char Signature;
	String SegName, Code, Enter;
	int CodeLength, AdressOfCode;

	Cards(char s, String sn, int ca) {
		Signature = s;
		SegName = sn;
		AdressOfCode = ca;
	}

	Cards(char s, int ca, int cl, String code) {
		Signature = s;
		AdressOfCode = ca;
		CodeLength = cl;
		Code = code;
	}

	Cards(char s, String e) {
		Signature = s;
		Enter = e;
	}
}
```

SymbolTable.java

```
package com.worm2fed.sp.kursach;
// ********************************************************************** //
// ******************* ТАБЛИЦА СИМВОЛИЧЕСКИХ ИМЁН ********************** //
// ********************************************************************** //

public class SymbolTable {
	String Name;
	String Offset;
	String Type;

	SymbolTable(String n, String off, String type) {
		this.Name = n;
		this.Offset = off;
		this.Type = type;
	}

	String getAdress(String name) {
		return Offset;
	}
}
```

TableOfCommand.java

```
package com.worm2fed.sp.kursach;
// ********************************************************************** //
// ******************* ТАБЛИЦА КОММАНД ********************************** //
// ********************************************************************** //

public class TableOfCommand {
	String MnemonicName, length, code;
	int OperandNumbers;
	String[] OperandsFormat = new String[2];
	TableOfCommand[] ComTab = new TableOfCommand[5];

	TableOfCommand() {
		MnemonicName = "";
		OperandNumbers = 0;
		OperandsFormat[0] = "";
		OperandsFormat[1] = "";
		length = "";
		code = "";
	}

	TableOfCommand(int i) {
		for (int j = 0; j < i; j++)
			ComTab[j] = new TableOfCommand();

		ComTab[0].MnemonicName = "MOV";
		ComTab[0].OperandNumbers = 2;
		ComTab[0].OperandsFormat[0] =
"Reg8,Reg16,Reg8,Reg16,Reg16,Reg8,Reg8,Reg16,Reg16,Mem8,Mem8,Mem16,Mem16";
		ComTab[0].OperandsFormat[1] =
"Reg8,Reg16,Imm8,Imm8,Imm16,Mem8,Mem16,Mem8,Mem16,Reg8,Reg16,Reg8,Reg16";
		ComTab[0].length = "2,2,3,4,4,3,4,3,4,3,3,4,4,3,4";
		ComTab[0].code = "8A11,8B11,C6,C7,C7,8A,8A,8B,8B,88,89,88,89";

		ComTab[1].MnemonicName = "XCHG";
		ComTab[1].OperandNumbers = 2;
		ComTab[1].OperandsFormat[0] =
"Reg8,Reg16,Reg8,Reg16,Mem8,Reg16,Mem16,Mem16";
```

```
            ComTab[1].OperandsFormat[1] =
"Reg8,Reg16,Mem8,Mem8,Reg8,Mem16,Reg16,Reg8";
            ComTab[1].length = "2,2,4,4,4,4,4,4";
            ComTab[1].code = "86,87,86,87,86,87,87,86";

            ComTab[2].MnemonicName = "IMUL";
            ComTab[2].OperandNumbers = 1;
            ComTab[2].OperandsFormat[0] = "Reg8,Reg16";
            ComTab[2].OperandsFormat[1] = " , ";
            ComTab[2].length = "2,2";
            ComTab[2].code = "F6,F7";

            ComTab[3].MnemonicName = "LOOP";
            ComTab[3].OperandNumbers = 1;
            ComTab[3].OperandsFormat[0] = "Label";
            ComTab[3].OperandsFormat[1] = " ";
            ComTab[3].length = "2";
            ComTab[3].code = "E2";

            ComTab[4].MnemonicName = "INT";
            ComTab[4].OperandNumbers = 1;
            ComTab[4].OperandsFormat[0] = "Imm8";
            ComTab[4].OperandsFormat[1] = " ";
            ComTab[4].length = "2";
            ComTab[4].code = "CD";
      }

      int getOpNum(String Name) {
            int i;

            a: for (i = 0; i < ComTab.length; i++)
                  if (ComTab[i].MnemonicName.equals(Name))
                        break a;

            return ComTab[i].OperandNumbers;
      }

      boolean isRightOperands(String Com, String TOp1, String TOp2) {
            int i;

            for (i = 0; i < ComTab.length; i++)
                  if (Com.equals(ComTab[i].MnemonicName))
                        break;

            String[] t1 = ComTab[i].OperandsFormat[0].split(",+");
            String[] t2 = ComTab[i].OperandsFormat[1].split(",+");
            int j;

            for (j = 0; j < t1.length; j++)
                  if (t1[j].equals(TOp1) && t2[j].trim().equals(TOp2))
                        return true;

            return false;
      }

      int getInstrLen(String Com, String TOp1, String TOp2) {
            int i;

            for (i = 0; i < ComTab.length; i++)
                  if (Com.equals(ComTab[i].MnemonicName))
                        break;
```

```
            String[] t1 = ComTab[i].OperandsFormat[0].split(",+");
            String[] t2 = ComTab[i].OperandsFormat[1].split(",+");
            String[] l = ComTab[i].length.split(",+");
            int j;

            for (j = 0; j < t1.length; j++)
                  if (t1[j].equals(TOp1) && t2[j].trim().equals(TOp2))
                        break;

            return Integer.parseInt(l[j]);
      }

      String getCode(String Com, String TOp1, String TOp2) {
            int i;

            for (i = 0; i < ComTab.length; i++)
                  if (Com.equals(ComTab[i].MnemonicName))
                        break;

            String[] t1 = ComTab[i].OperandsFormat[0].split(",+");
            String[] t2 = ComTab[i].OperandsFormat[1].split(",+");
            String[] code = ComTab[i].code.split(",+");
            int j;

            for (j = 0; j < t1.length; j++)
                  if (t1[j].equals(TOp1) && t2[j].trim().equals(TOp2))
                        break;

            return code[j];
      }
}
```

TableOfDirective.java

```
package com.worm2fed.sp.kursach;

// ************************************************************************** //
// ******************* ТАБЛИЦА ДИРЕКТИВ ********************************** //
// ************************************************************************** //
public class TableOfDirective {
      String DirName, length;
      String[] OperandsFormat = new String[2];
      TableOfDirective[] DirTab = new TableOfDirective[4];

      TableOfDirective() {
            DirName = "";
            length = "";
            OperandsFormat[0] = "";
            OperandsFormat[1] = "";
      }

      TableOfDirective(int i) {
            for (int j = 0; j < i; j++)
                  DirTab[j] = new TableOfDirective();

            DirTab[0].DirName = "ORG";
            DirTab[0].length = "";
            DirTab[0].OperandsFormat[0] = "Imm8,Imm16";
            DirTab[0].OperandsFormat[1] = " , ";

            DirTab[1].DirName = "OFFSET";
```

```
		DirTab[1].length = "4";
		DirTab[1].OperandsFormat[0] = "Imm16";
		DirTab[1].OperandsFormat[1] = " ";

		DirTab[2].DirName = "DB";
		DirTab[2].length = "1,2";
		DirTab[2].OperandsFormat[0] = "Imm8,Imm8";
		DirTab[2].OperandsFormat[1] = " ,Imm8";

		DirTab[3].DirName = "DW";
		DirTab[3].length = "2,4,2,4,4,4";
		DirTab[3].OperandsFormat[0] = "Imm8,Imm8,Imm16,Imm16,Imm8,Imm16";
		DirTab[3].OperandsFormat[1] = " ,Imm8, ,Imm16,Imm16,Imm8";
	}

	boolean DirEn(String Dir) {
		for (int i = 0; i < DirTab.length; i++)
			if (Dir.contains(DirTab[i].DirName))
				return true;

		return false;
	}

	boolean isRightOperands(String Com, String TOp1, String TOp2) {
		int i;

		for (i = 0; i < DirTab.length; i++)
			if (Com.equals(DirTab[i].DirName))
				break;

		String[] t1 = DirTab[i].OperandsFormat[0].split(",+");
		String[] t2 = DirTab[i].OperandsFormat[1].split(",+");
		int j;

		for (j = 0; j < t1.length; j++)
			if (t1[j].equals(TOp1) && t2[j].trim().equals(TOp2))
				return true;

		return false;
	}

	int getInstrLen(String Com, String TOp1, String TOp2) {
		int i;

		for (i = 0; i < DirTab.length; i++)
			if (Com.equals(DirTab[i].DirName))
				break;

		String[] t1 = DirTab[i].OperandsFormat[0].split(",+");
		String[] t2 = DirTab[i].OperandsFormat[1].split(",+");
		String[] l = DirTab[i].length.split(",+");
		int j;

		for (j = 0; j < t1.length; j++)
			if (t1[j].trim().equals(TOp1) && t2[j].trim().equals(TOp2))
				break;

		return Integer.parseInt(l[j]);
	}
}
```

TableOfRegisters.java

```
package com.worm2fed.sp.kursach;
// ********************************************************************** //
// ****************** ТАБЛИЦА РЕГИСТРОВ ********************************** //
// ********************************************************************** //

public class TableOfRegisters {
	String RegName, type, code;
	TableOfRegisters[] RegTab = new TableOfRegisters[16];

	TableOfRegisters() {
		RegName = "";
		type = "";
		code = "";
	}

	TableOfRegisters(int i) {
		for (int j = 0; j < i; j++)
			RegTab[j] = new TableOfRegisters();

		RegTab[0].RegName = "AL";
		RegTab[0].type = "Reg8";
		RegTab[0].code = "000";

		RegTab[1].RegName = "AH";
		RegTab[1].type = "Reg8";
		RegTab[1].code = "100";

		RegTab[2].RegName = "AX";
		RegTab[2].type = "Reg16";
		RegTab[2].code = "000";

		RegTab[3].RegName = "BL";
		RegTab[3].type = "Reg8";
		RegTab[3].code = "011";

		RegTab[4].RegName = "BH";
		RegTab[4].type = "Reg8";
		RegTab[4].code = "111";

		RegTab[5].RegName = "BX";
		RegTab[5].type = "Reg16";
		RegTab[5].code = "011";

		RegTab[6].RegName = "CL";
		RegTab[6].type = "Reg8";
		RegTab[6].code = "001";

		RegTab[7].RegName = "CH";
		RegTab[7].type = "Reg8";
		RegTab[7].code = "101";

		RegTab[8].RegName = "CX";
		RegTab[8].type = "Reg16";
		RegTab[8].code = "001";

		RegTab[9].RegName = "DL";
		RegTab[9].type = "Reg8";
		RegTab[9].code = "010";

		RegTab[10].RegName = "DH";
```

```
		RegTab[10].type = "Reg8";
		RegTab[10].code = "110";

		RegTab[11].RegName = "DX";
		RegTab[11].type = "Reg16";
		RegTab[11].code = "010";

		RegTab[12].RegName = "SP";
		RegTab[12].type = "Reg16";
		RegTab[12].code = "100";

		RegTab[13].RegName = "BP";
		RegTab[13].type = "Reg16";
		RegTab[13].code = "101";

		RegTab[14].RegName = "SI";
		RegTab[14].type = "Reg16";
		RegTab[14].code = "110";

		RegTab[15].RegName = "DI";
		RegTab[15].type = "Reg16";
		RegTab[15].code = "111";
	}

	boolean isReg(String r) {
		for (int i = 0; i < RegTab.length; i++)
			if (r.equals(RegTab[i].RegName))
				return true;

		return false;
	}

	String getRegType(String r) {
		int i;

		for (i = 0; i < RegTab.length; i++)
			if (r.equals(RegTab[i].RegName))
				break;

		return RegTab[i].type;
	}

	String getRegCode(String r) {
		int i;

		for (i = 0; i < RegTab.length; i++)
			if (r.equals(RegTab[i].RegName))
				break;

		return RegTab[i].code;
	}
}
```

Structure.java

```
package com.worm2fed.sp.kursach;

import java.io.*;
import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
```

```
public class Structure {
	public boolean PrintCards, PrintSymb;
	public String file;

	private int PosFragment, Schak, CodeLen, Value;
	private String SegmentName, Listing, Code, ListOfErrors;
	private boolean ErrorFlag;
	private RandomAccessFile ObjectCode;
	private PrintWriter ListingFile, Cardfile, SymbolFile;

	private TableOfCommand TabOfCom = new TableOfCommand(5);
	private TableOfDirective TabOfDir = new TableOfDirective(4);
	private TableOfRegisters TabOfReg = new TableOfRegisters(16);
	private ArrayList<SymbolTable> SymbolTab = new ArrayList<SymbolTable>();
	private Cards H;
	private ArrayList<Cards> T = new ArrayList<Cards>();
	private Cards E;
	private PositionInProgramm position = new PositionInProgramm();

	final int StructureError = 0;
	final int OperandsError = 1;
	final int UnknownCommand = 2;
	final int ReferError = 3;
	final int MultipleError = 4;
	final int AddressingError = 5;

	String[] Error = { "Ошибка в структуре программы", "Несоответствие типов
и/или количества операндов",
			"Неизвестная команда (инструкция)", "Ссылка на неопределённое
имя или метку",
			"Повторное определение метки или переменной", "Неверная
адресация", "Переполнение таблиц символов" };

	String[] ComAndDir = { "SEGMENT", "ORG", "DB", "DW", "ENDS", "END",
"OFFSET", "MOV", "XCHG", "IMUL", "LOOP",
			"INT" };
	String[] Types = { "Dir", "Dir", "Dir", "Dir", "Dir", "Dir", "Dir", "Com",
"Com", "Com", "Com", "Com" };

	Structure() {
		PosFragment = 0;
		SegmentName = "";
		Listing = "";
		ErrorFlag = false;
		Schak = 0;
		ListOfErrors = "";
		CodeLen = 0;
		Code = "";
	}

	public boolean Begin(String MnemonicCode) throws IOException {
		// первый просмотр
		firstPass(MnemonicCode);

		/*
		 * писать карты в объектный файл будем только в случае успешного
второго
		 * просмотра
		 */
		if (secondPass(MnemonicCode)) {
			// запись листинга
```

```
                writeListing();
                // запись объектного кода
                writeObject();

                // запись карт
                if (PrintCards)
                      writeCards();

                // запись таблицы символических имен
                if (PrintSymb)
                      writeSymbol();

                return true;
          } else {
                // запись листинга
                writeListing();

                return false;
          }
    }

    // первый просмотр
    void firstPass(String t) {
          String[] Letters;
          String Text = t.toString();
          // разбивание текста на массив строк
          String[] Lines = Text.split("\\n+");

          /*
           * основной цикл, в котором последовательно происходит проверка всех
           * строк программы
           */
          for (int i = 0; i < Lines.length; i++) {
                /*
                 * текущая строка: в верхний регистр и убираем пробелы слева и
                 * справа
                 */
                String CurrentLine = Lines[i].toUpperCase().trim();
                CurrentLine = DeleteComments(CurrentLine); // убираем
комментарии

                // пропускаем пустые строки
                if (CurrentLine.trim().equals(""))
                      continue;

                // команды в текущем месте сегмента
                String[] CurrentCommand =
position.getCurrCom(PosFragment).split(",+");
                // Команды в следующем после текущего места сегмента
                String[] NextCommand = position.getNextCom(PosFragment).split(",
+");

                // если мы находимся в области определения сегмента, то...
                if (PosFragment == 0) {
                      // разбиваем текущую линию на отдельные слова
                      Letters = CurrentLine.split(" +");

                      // если такая команда вообще существует...
                      if (OurComAndDir(Letters, ComAndDir)) {
                            // если сегмент есть вообще
                            if (OurComAndDir(Letters, CurrentCommand)) {
                                  // если сразу идёт сегмент, то ошибка...
```

```
                                    if (Letters[0].equals(CurrentCommand[0]))
                                          Err((i + 1) + ". " + "Не указано имя
сегмента\n");
                                    else if (Letters[1].equals(CurrentCommand[0]))
                                          SegmentName = Letters[0].toUpperCase();
                                    else
                                          Err((i + 1) + ". " + "Имя сегмента
состоит из более чем 1 слова\n");
                              } else {
                                    // если команда из другой области
                                    if (OurComAndDir(Letters, ComAndDir))
                                          Err((i + 1) + ". " +
Error[StructureError] + ".\n");
                                    // если просто забыли указать СЕГМЕНТ
                                    else
                                          Err((i + 1) + ". " + "Забыли указать
ключевое слово SEGMENT\n");
                              }

                              if (i == (Lines.length - 1)) {
                                    Err((i + 1) + ". " + "Неожиданное окончание
сегмента\n");
                                    break;
                              }

                              // двигаемся дальше
                              PosFragment = position.getNextPosSeg(PosFragment);
                              continue;
                        } else {
                              Err((i + 1) + ". " + Error[UnknownCommand] + "\n");
                              continue;
                        }
                  }

                  // если находимся в области команд, то...
                  if (PosFragment == 1) {
                        // разбиваем текущую строку на отдельные слова
                        Letters = CurrentLine.split("[ :]+");

                        // если такая команда вообще существует...
                        if (OurComAndDir(Letters, ComAndDir)) {
                              // если она является нормальной для данной области
                              if (OurComAndDir(Letters, CurrentCommand)) {
                                    // подготовка к распознаванию метки, команды и
                                    // операндов...
                                    String Metka = "", Command = "", Op1 = "", Op2
= "";

                                    // если метка есть
                                    if (CurrentLine.indexOf(':') != -1)
                                          // то выделить и запомнить её
                                          Metka = CurrentLine.substring(0,
CurrentLine.indexOf(':'));

                                    // метка уже не нужна в основной строке
                                    CurrentLine =
CurrentLine.substring(CurrentLine.indexOf(':') + 1, CurrentLine.length()).trim();

                                    Pattern p = Pattern.compile("[a-zA-Z]{1}[a-zA-
Z\\d]*");
                                    Matcher m = p.matcher(Metka);
                                    boolean b = m.matches();
```

```
                                    if (!Metka.equals("")) {
                                          if (!b) {
                                                Err((i + 1) + ". " + "Неправильное
определение метки" + "\n");
                                                continue;
                                          } else if (DoubleVarOrLabel(Metka))
                                                Err((i + 1) + ". " +
Error[MultipleError] + "\n");
                                          else
                                                SymbolTab.add(new
SymbolTable(Metka, HexSchak(Schak, 4), "Label"));
                                    }

                                    int index = 0;
                                    char c = CurrentLine.charAt(index);

                                    // поиск команды
                                    while (index != CurrentLine.length() - 1 &&
c != ' ') {
                                          Command += c;
                                          index++;
                                          c = CurrentLine.charAt(index);
                                    }

                                    // избавляемся от команды в основной строке
                                    CurrentLine = CurrentLine
                                                .substring(CurrentLine.indexOf(Com
mand) + Command.length(), CurrentLine.length())
                                                .trim();
                                    // если после команды нет операндов
                                    if (CurrentLine.equals("")) {
                                          Err((i + 1) + ". " +
Error[OperandsError] + "\n");
                                          continue;
                                    }

                                    // находим тип операции
                                    String OperationType = FindType(Command);
                                    // если неизвестный тип (неправильно
расположили
                                    // операцию и операнды)
                                    if (OperationType.equals("Unknown")) {
                                          Err((i + 1) + ". " +
Error[UnknownCommand] + "\n");
                                          continue;
                                          // если обнаружена команда
                                    } else if (OperationType.equals("Com")) {
                                          // находим количество операндов для
данной команды
                                          int OpNum = TabOfCom.getOpNum(Command);
                                          // если операндов должно быть 2
                                          if (OpNum == 2) {
                                                // поиск операндов
                                                if (CurrentLine.indexOf(',') !=
-1) {
                                                      Op1 =
CurrentLine.substring(0, CurrentLine.indexOf(',')).trim();
                                                      Op2 =
CurrentLine.substring(CurrentLine.indexOf(',') + 1, CurrentLine.length())
                                                                  .trim();
                                                      // если не нашли запятую
```

```
                                                } else {
                                                      Err((i + 1) + ". " + "Между
операндами отсутствует запятая" + "\n");
                                                      continue;
                                                }
                                                // если 1 операнд
                                          } else
                                                Op1 = CurrentLine.substring(0,
CurrentLine.length()).trim();
                                          // если операция - директива
                                    } else if (OperationType.equals("Dir"))
                                          Op1 = CurrentLine.substring(0,
CurrentLine.length()).trim();

                                    // определение типа первого и второго операнда
                                    String Op1Kind = OpKind(Op1, 1);
                                    String Op2Kind = OpKind(Op2, 1);

                                    // в случае неверной адресации
                                    if (Op1Kind.equals("UnkAdr") ||
Op2Kind.equals("UnkAdr")) {
                                          Err((i + 1) + ". " +
Error[AddressingError] + "\n");
                                          continue;
                                    }

                                    // если нашли несоответствие типов операндов
                                    if ((OperationType.equals("Com") && !
TabOfCom.isRightOperands(Command, Op1Kind, Op2Kind))
                                                || (OperationType.equals("Dir")
                                                            && !
TabOfDir.isRightOperands(Command, Op1Kind, Op2Kind))) {
                                          Err((i + 1) + ". " +
Error[OperandsError] + "\n");
                                          continue;
                                    }

                                    if (OperationType.equals("Dir")) {
                                          Schak = Value;
                                          continue;
                                    }

                                    // увеличиваем СЧАК
                                    Schak += TabOfCom.getInstrLen(Command,
Op1Kind, Op2Kind);
                                    // если попали в следующую область
                              } else if (OurComAndDir(Letters, NextCommand)) {
                                    PosFragment =
position.getNextPosSeg(PosFragment);
                                    i--;
                                    continue;
                                    // если сразу идет закрытие сегмента
                              } else if (OurComAndDir(Letters,
position.getCurrCom(3).split(",+"))) {
                                    PosFragment = 3;
                                    i--;
                                    continue;
                                    // если сразу идёт окончание программы
                              } else if (OurComAndDir(Letters,
position.getCurrCom(4).split(",+"))) {
                                    PosFragment = 4;
```

```
                                    Err((i + 1) + ". " + "Нет директивы закрытия
сегмента" + "\n");
                                    i--;
                                    continue;
                                    // если команда не подходящая под предыдущие
описания
                              } else {
                                    Err((i + 1) + ". " + Error[StructureError] +
"\n");
                                    continue;
                              }
                              // если нет ...
                        } else {
                              Err((i + 1) + ". " + Error[UnknownCommand] + "\n");
                              continue;
                        }
                  }

                  // если находимся в области данных, то...
                  if (PosFragment == 2) {
                        // разбиваем текущую строку на отдельные слова
                        Letters = CurrentLine.split(" +");
                        // если команда существует...
                        if (OurComAndDir(Letters, ComAndDir)) {
                              // если она является нормальной для данной области
                              if (OurComAndDir(Letters, CurrentCommand)) {
                                    if (Letters[0].equals("") != true &&
OurComAndDir(Letters[1], CurrentCommand)) {
                                          // инициализация двух возможных
операндов и самой
                                          // директивы
                                          String Directive = Letters[1], Op1 = "",
Op2 = "";

                                          // поиск операндов
                                          if (CurrentLine.indexOf(',') != -1) {
                                                Op1 =
CurrentLine.substring(CurrentLine.indexOf(Letters[1]) + Letters[1].length(),

      CurrentLine.indexOf(',')).trim();
                                                Op2 =
CurrentLine.substring(CurrentLine.indexOf(',') + 1, CurrentLine.length()).trim();
                                                // если 1 операнд
                                          } else
                                                Op1 =
CurrentLine.substring(CurrentLine.indexOf(Letters[1]) + Letters[1].length(),

      CurrentLine.length()).trim();

                                          // определение типа операндов
                                          String Op1Kind = OpKind(Op1, 1);
                                          String Op2Kind = OpKind(Op2, 1);

                                          // если нашли несоответствие типов
операндов
                                          if (!TabOfDir.isRightOperands(Directive,
Op1Kind, Op2Kind)) {
                                                Err((i + 1) + ". " +
Error[OperandsError] + "\n");
                                                continue;
                                          }
```

```
                                        if (DoubleVarOrLabel(Letters[0]))
                                              Err((i + 1) + ". " +
Error[MultipleError] + "\n");
                                        else
                                              SymbolTab.add(new
SymbolTable(Letters[0], HexSchak(Schak, 4), "VAR"));

                                        Schak +=
TabOfDir.getInstrLen(Letters[1], Op1Kind, Op2Kind);
                                  } else {
                                        Err((i + 1) + ". " + "Имя переменной не
определено" + "\n");
                                        continue;
                                  }
                                  // если попали в следующую область
                            } else if (OurComAndDir(Letters, NextCommand)) {
                                  PosFragment =
position.getNextPosSeg(PosFragment);
                                  i--;
                                  continue;
                                  // если сразу идёт окончание программы
                            } else if (OurComAndDir(Letters,
position.getCurrCom(4).split(",+"))) {
                                  PosFragment = 4;
                                  Err((i + 1) + ". " + "Нет директивы закрытия
сегмента" + "\n");
                                  i--;
                                  continue;
                                  // если команда не подходящая под предыдущие
описания
                            } else {
                                  Err((i + 1) + ". " + Error[StructureError] +
"\n");
                                  continue;
                            }
                            // если нет ...
                      } else {
                            Err((i + 1) + ". " + Error[UnknownCommand] + "\n");
                            continue;
                      }
                }

                // если находимся в области окончания сегмента, то...
                if (PosFragment == 3) {
                      // разбиваем текущую линию на отдельные слова
                      Letters = CurrentLine.split(" +");
                      // если сразу идёт закрытие сегмента, то ошибка...
                      if (Letters[0].equals(CurrentCommand[0]))
                            Err((i + 1) + ". " + "Не указано имя сегмента\n");
                      // если окончание имеется
                      else if ((Letters.length != 1) &&
Letters[1].equals(CurrentCommand[0])) {
                            if (Letters[0].equals(SegmentName)) {

                            } else
                                  Err((i + 1) + ". " + "Имя сегмента при его
закрытии указано неправильно\n");
                      } else
                            Err((i + 1) + ". " + "Слишком много букв при
закрытии сегмента!" + "\n");

                      if (i == (Lines.length - 1)) {
```

```
                              Err((i + 1) + ". " + "Неожиданное окончание
программы\n");
                              break;
                        }

                        PosFragment = position.getNextPosSeg(PosFragment);
                        continue;
                  }

                  // если находимся в области окончания программы, то...
                  if (PosFragment == 4) {
                        // если конец программы
                        if (CurrentLine.trim().equals(CurrentCommand[0])) {
                              // проверяем нет ли чего в конце
                              if ((Lines.length - 1) != i) {
                                    Err((i + 1) + ". " + "Конец у программы
присутствует " + "\n");
                                    break;
                              }
                              // ошибка структуры
                        } else
                              Err((i + 1) + ". " + "Ожидалось окончание
программы!" + "\n");
                        break;
                  }
            }
            if (PosFragment == 1) {
                  Err((Lines.length + 1) + ". "
                              + "Неожиданное окончание программы (в этой строке
нехватает директив закрытия сегмента и окончания программы)\n");
            }

            H = new Cards('H', SegmentName, Schak); // формирование карты H
            Schak = 0; // сброс СЧАКа
            PosFragment = 0; // сброс позиции в программе

      }

      // второй просмотр
      boolean secondPass(String t) {
            String[] Letters;
            String Text = t.toString();
            String[] Lines = Text.split("\\n+");

            for (int i = 0; i < Lines.length; i++) {
                  if (HaveErr(i + 1)) {
                        Listing += ListOfErrors.substring(0,
ListOfErrors.indexOf("\n")) + ":      " + Lines[i].trim() + "\n";
                        ListOfErrors =
ListOfErrors.substring(ListOfErrors.indexOf("\n") + 1, ListOfErrors.length());
                        continue;
                  }

                  String CurrentLine = Lines[i].toUpperCase().trim();
                  CurrentLine = DeleteComments(CurrentLine);

                  if (CurrentLine.trim().equals("")) {
                        Listing += (i + 1) + ". " + "   :             " +
Lines[i].trim() + "\n";
                        continue;
                  }
```

```
                String[] CurrentCommand =
position.getCurrCom(PosFragment).split(",+");
                String[] NextCommand = position.getNextCom(PosFragment).split(",
+");

                if (PosFragment == 0) {
                      Letters = CurrentLine.split(" +");
                      if (OurComAndDir(Letters, ComAndDir)) {
                            if (OurComAndDir(Letters, CurrentCommand)) {
                                  if (Letters[0].equals(CurrentCommand[0]))
                                        Listing += (i + 1) + ". " + "Не указано
имя сегмента" + ":   " + Lines[i].trim() + "\n";
                                  else if (Letters[1].equals(CurrentCommand[0]))
{
                                        SegmentName = Letters[0];
                                        Listing += (i + 1) + ". " + "   :
" + Lines[i].trim()
                                                    + "\n";
                                  } else
                                        Listing += (i + 1) + ". " + "Имя
сегмента состоит из более чем 1 слова" + ":   "
                                                    + Lines[i].trim() + "\n";
                            } else if (OurComAndDir(Letters, NextCommand)) {
                                  PosFragment =
position.getNextPosSeg(PosFragment);
                                  i--;
                                  continue;
                            } else if (i == (Lines.length - 1)) {
                                  Listing += (i + 1) + ". " + "Неожиданное
окончание сегмента" + ":   " + Lines[i].trim() + "\n";
                                  break;
                            } else {
                                  if (OurComAndDir(Letters, ComAndDir))
                                        Listing += (i + 1) + ". " +
Error[StructureError] + ":   " + Lines[i].trim() + "\n";
                                  else
                                        Listing += (i + 1) + ". " + "Забыли
указать ключевое слово SEGMENT\n" + ":   "
                                                    + Lines[i].trim() + "\n";
                            }

                            PosFragment = position.getNextPosSeg(PosFragment);
                            continue;
                      } else {
                            Listing += (i + 1) + ". " + Error[UnknownCommand] +
":   " + Lines[i].trim() + "\n";
                            continue;
                      }
                }

                if (PosFragment == 1) {
                      Letters = CurrentLine.split("[ :]+");
                      if (OurComAndDir(Letters, ComAndDir)) {
                            if (OurComAndDir(Letters, CurrentCommand)) {
                                  String Metka = "", Command = "", Op1 = "", Op2
= "";

                                  if (CurrentLine.indexOf(':') != -1)
                                        Metka = CurrentLine.substring(0,
CurrentLine.indexOf(':'));
```

```
                                    CurrentLine =
CurrentLine.substring(CurrentLine.indexOf(':') + 1, CurrentLine.length()).trim();

                                    int index = 0;
                                    char c = CurrentLine.charAt(index);

                                    while (index != CurrentLine.length() - 1 &&
c != ' ') {
                                          Command += c;
                                          index++;
                                          c = CurrentLine.charAt(index);
                                    }

                                    CurrentLine = CurrentLine
                                                .substring(CurrentLine.indexOf(Com
mand) + Command.length(), CurrentLine.length())
                                                .trim();

                                    if (CurrentLine.equals("")) {
                                          Listing += (i + 1) + ". " +
Error[OperandsError] + ":   " + Lines[i].trim() + "\n";
                                          continue;
                                    }

                                    String OperationType = FindType(Command);
                                    if (OperationType.equals("Unknown")) {
                                          Listing += (i + 1) + ". " +
Error[UnknownCommand] + ":   " + Lines[i].trim() + "\n";
                                          continue;
                                    } else if (OperationType.equals("Com")) {
                                          int OpNum = TabOfCom.getOpNum(Command);

                                          if (OpNum == 2) {
                                                if (CurrentLine.indexOf(',') !=
-1) {
                                                      Op1 =
CurrentLine.substring(0, CurrentLine.indexOf(',')).trim();
                                                      Op2 =
CurrentLine.substring(CurrentLine.indexOf(',') + 1, CurrentLine.length())
                                                                  .trim();

                                                } else {
                                                      Err((i + 1) + ". " + "Между
операндами отсутствует запятая" + "\n");
                                                      continue;
                                                }
                                          } else
                                                Op1 = CurrentLine.substring(0,
CurrentLine.length()).trim();
                                    } else if (OperationType.equals("Dir"))
                                          Op1 = CurrentLine.substring(0,
CurrentLine.length()).trim();

                                    String Op1Kind = OpKind(Op1, 2);
                                    String Op2Kind = OpKind(Op2, 2);

                                    if (Op1Kind.equals("UnkAdr") ||
Op2Kind.equals("UnkAdr")) {
                                          Listing += (i + 1) + ". " +
Error[AddressingError] + ":   " + Lines[i].trim() + "\n";
                                          continue;
                                    }
```

```
                                    if (Op1Kind.equals("UnknVar") ||
Op2Kind.equals("UnknVar")) {
                                          Listing += (i + 1) + ". " +
Error[ReferError] + ":   " + Lines[i].trim() + "\n";
                                          continue;
                                    }

                                    if ((OperationType.equals("Com") && !
TabOfCom.isRightOperands(Command, Op1Kind, Op2Kind))
                                                || (OperationType.equals("Dir")
                                                      && !
TabOfDir.isRightOperands(Command, Op1Kind, Op2Kind))) {
                                          Listing += (i + 1) + ". " +
Error[OperandsError] + ":   " + Lines[i].trim() + "\n";
                                          continue;
                                    }
                                    if (OperationType.equals("Dir")) {
                                          Listing += (i + 1) + ". " + "   :
" + Lines[i].trim()
                                                      + "\n";
                                          Schak = Value;
                                          continue;
                                    }

                                    // получение длины инструкции
                                    CodeLen = TabOfCom.getInstrLen(Command,
Op1Kind, Op2Kind);
                                    // получение кода инструкции
                                    Code = getCode(Command, Op1, Op2, Op1Kind,
Op2Kind, CodeLen);
                                    // формирование строки листинга
                                    Listing += (i + 1) + ". " + HexSchak(Schak, 4)
+ ":  " + Code.toUpperCase()
                                                + "
".substring(0, 30 - 5 * CodeLen) + Lines[i].trim() + "\n";
                                    // формирование карты T
                                    T.add(new Cards('T', Schak, CodeLen, Code));
                                    // увеличение СЧАКа на длину инструкции
                                    Schak += CodeLen;
                              } else if (OurComAndDir(Letters, NextCommand)) {
                                    PosFragment =
position.getNextPosSeg(PosFragment);
                                    i--;
                                    continue;
                              } else if (OurComAndDir(Letters,
position.getCurrCom(3).split(",+"))) {
                                    PosFragment = 3;
                                    i--;
                                    continue;
                              } else if (OurComAndDir(Letters,
position.getCurrCom(4).split(",+"))) {
                                    PosFragment = 4;
                                    Listing += (i + 1) + ". " + "Нет директивы
закрытия сегмента" + ":   " + Lines[i].trim() + "\n";
                                    i--;
                                    continue;
                              } else {
                                    Listing += (i + 1) + ". " +
Error[StructureError] + ":   " + Lines[i].trim() + "\n";
                                    continue;
                              }
```

```
                        } else {
                              Listing += (i + 1) + ". " + Error[UnknownCommand] +
":   " + Lines[i].trim() + "\n";
                              continue;
                        }
                  }

                  if (PosFragment == 2) {
                        Letters = CurrentLine.split(" +");
                        if (OurComAndDir(Letters, ComAndDir)) {
                              if (OurComAndDir(Letters, CurrentCommand)) {
                                    if (Letters[0].equals("") != true &&
OurComAndDir(Letters[1], CurrentCommand)) {
                                          String Directive = Letters[1], Op1 = "",
Op2 = "";

                                          if (CurrentLine.indexOf(',') != -1) {
                                                Op1 =
CurrentLine.substring(CurrentLine.indexOf(Letters[1]) + Letters[1].length(),

      CurrentLine.indexOf(',')).trim();
                                                Op2 =
CurrentLine.substring(CurrentLine.indexOf(',') + 1, CurrentLine.length()).trim();
                                          } else
                                                Op1 =
CurrentLine.substring(CurrentLine.indexOf(Letters[1]) + Letters[1].length(),

      CurrentLine.length()).trim();

                                          String Op1Kind = OpKind(Op1, 2);
                                          String Op2Kind = OpKind(Op2, 2);

                                          if (!TabOfDir.isRightOperands(Directive,
Op1Kind, Op2Kind)) {
                                                Err((i + 1) + ". " +
Error[OperandsError] + "\n");
                                                continue;
                                          }

                                          // получение длины инструкции
                                          CodeLen =
TabOfDir.getInstrLen(Letters[1], Op1Kind, Op2Kind);
                                          // получение кода инструкции
                                          Code = getCode(Letters[1], Op1, Op2,
Op1Kind, Op2Kind, CodeLen);
                                          // формирование строки листинга
                                          Listing += (i + 1) + ". " +
HexSchak(Schak, 4) + ":  " + Code.toUpperCase()
                                                      + "
".substring(0, 30 - 5 * CodeLen) + Lines[i].trim()
                                                      + "\n";
                                          // формирование карты T
                                          T.add(new Cards('T', Schak, CodeLen,
Code));
                                          // инкрементирование СЧАКа на длину
инструкции
                                          Schak += CodeLen;
                                    } else {
                                          Listing += (i + 1) + ". " + "Имя
переменной не определено" + ":   " + Lines[i].trim()
                                                      + "\n";
                                          continue;
```

```
                                    }
                              } else if (OurComAndDir(Letters, NextCommand)) {
                                    PosFragment =
position.getNextPosSeg(PosFragment);
                                    i--;
                                    continue;
                              } else if (OurComAndDir(Letters,
position.getCurrCom(4).split(",+"))) {
                                    PosFragment = 4;
                                    Listing += (i + 1) + ". " + "Нет директивы
закрытия сегмента" + ":   " + Lines[i].trim() + "\n";
                                    i--;
                                    continue;
                              } else {
                                    Listing += (i + 1) + ". " +
Error[StructureError] + ":   " + Lines[i].trim() + "\n";
                                    continue;
                              }
                        } else {
                              Listing += (i + 1) + ". " + Error[UnknownCommand] +
":   " + Lines[i].trim() + "\n";
                              continue;
                        }
                  }

                  if (PosFragment == 3) {
                        Letters = CurrentLine.split(" +");
                        if (Letters[0].equals(CurrentCommand[0]))
                              Listing += (i + 1) + ". " + "Не указано имя
сегмента" + ":   " + Lines[i].trim() + "\n";
                        else if ((Letters.length != 1) &&
Letters[1].equals(CurrentCommand[0])) {
                              if (Letters[0].equals(SegmentName))
                                    Listing += (i + 1) + ". " + "   :
" + Lines[i].trim()
                                                + "\n";
                              else
                                    Listing += (i + 1) + ". " + "Имя сегмента
указано неправильно" + ":   " + Lines[i].trim()
                                                + "\n";
                        } else
                              Listing += (i + 1) + ". " + "Слишком много буков при
закрытии сегмента!" + ":   " + Lines[i].trim()
                                          + "\n";

                        if (i == (Lines.length - 1)) {
                              Listing += (i + 1) + ". " + "Неожиданное окончание
программы" + ":   " + Lines[i].trim() + "\n";
                              break;
                        }

                        PosFragment = position.getNextPosSeg(PosFragment);
                        continue;
                  }

                  if (PosFragment == 4) {
                        if (CurrentLine.trim().equals(CurrentCommand[0])) {
                              if ((Lines.length - 1) != i) {
                                    Listing += (i + 1) + ". " + "Конец у программы
пристутствует\n";
                                    break;
                              }
```

```
                              Listing += (i + 1) + ". " + "   :
" + Lines[i].trim() + "\n";
                        } else
                              Listing += (i + 1) + ". " + "Ожидалось окончание
программы!" + ":   " + Lines[i].trim() + "\n";

                        break;
                  }
            }

            if (PosFragment == 1) {
                  Listing += (Lines.length + 1) + ". "
                              + "Неожиданное окончание программы (в этой строке
нехватает директив закрытия сегмента и окончания программы)"
                              + ":   " + Lines[Lines.length - 1].trim() + "\n";
            }

            // формирование карты E
            E = new Cards('E', "0000");
            // writeCards();
            return !ErrorFlag;
      }

      // Запись объектного кода
      void writeObject() {
            try {
                  ObjectCode = new RandomAccessFile(file + ".bin", "rw");

                  // запись карты H
                  ObjectCode.writeByte(H.Signature);
                  ObjectCode.writeBytes(H.SegName);
                  ObjectCode.writeByte(from_16_to_10(HexSchak(H.AdressOfCode,
4).substring(2, 4).toUpperCase()));
                  ObjectCode.writeByte(from_16_to_10(HexSchak(H.AdressOfCode,
4).substring(0, 2).toUpperCase()));

                  // запись карты T
                  for (int j = 0; j < T.size(); j++) {
                        ObjectCode.writeByte(T.get(j).Signature);

      ObjectCode.writeByte(from_16_to_10(HexSchak(T.get(j).AdressOfCode,
4).substring(0, 2).toUpperCase()));

      ObjectCode.writeByte(from_16_to_10(HexSchak(T.get(j).AdressOfCode,
4).substring(2, 4).toUpperCase()));
                        ObjectCode.writeByte(T.get(j).CodeLength);

                        for (int i = 0; i < T.get(j).CodeLength; i++)

      ObjectCode.writeByte(from_16_to_10(T.get(j).Code.substring(2 * i, 2 * i +
2).toUpperCase()));
                  }

                  // запись карты E
                  ObjectCode.writeByte(E.Signature);
                  ObjectCode.writeByte(from_16_to_10(E.Enter.substring(0,
2).toUpperCase()));
                  ObjectCode.writeByte(from_16_to_10(E.Enter.substring(2,
4).toUpperCase()));

                  ObjectCode.close();
```

```
		} catch (IOException e) {
		}
	}

	// Запись листинга
	void writeListing() {
		try {
			ListingFile = new PrintWriter(file + ".lst");

			ListingFile.write(Listing);

			ListingFile.close();
		} catch (IOException e) {
		}
	}

	// Запись таблицы символических имён
	void writeSymbol() {
		try {
			SymbolFile = new PrintWriter(file + ".sym");

			SymbolFile.println("  Имя" + "      " + "Адрес       " + "Тип");
			for (int i = 0; i < SymbolTab.size(); i++)
				SymbolFile.println("  " + SymbolTab.get(i).Name + "
" + SymbolTab.get(i).Offset + "       "
					+ SymbolTab.get(i).Type);

			SymbolFile.close();
		} catch (IOException e) {
		}
	}

	// Запись карт
	void writeCards() {
		try {
			Cardfile = new PrintWriter(file + ".map");

			// запись карты H
			Cardfile.print(H.Signature + " ");
			Cardfile.print(H.SegName + " ");
			Cardfile.print(
					HexSchak(H.AdressOfCode, 4).substring(0, 2) +
HexSchak(H.AdressOfCode, 4).substring(2, 4) + " ");
			Cardfile.println();

			// запись карты T
			for (int j = 0; j < T.size(); j++) {
				Cardfile.print(T.get(j).Signature + " ");
				Cardfile.print(HexSchak(T.get(j).AdressOfCode, 4) + " ");
				Cardfile.print(HexSchak(T.get(j).CodeLength, 2) + " ");
				Cardfile.print(T.get(j).Code + " ");
				Cardfile.println();
			}

			// запись карты E
			Cardfile.print(E.Signature + " ");
			Cardfile.print(E.Enter + " ");

			Cardfile.close();
		} catch (IOException e) {
		}
	}
```

```
	// Получение кода инструкции
	String getCode(String Command, String Op1, String Op2, String Op1Kind,
String Op2Kind, int CodeLen) {
		// получаем начальный код инструкции и продолжаем формирование кода
		String code = "";

		// если командой является MOV
		if (Command.equals("MOV")) {
			code = TabOfCom.getCode(Command, Op1Kind, Op2Kind);
			if ((Op1Kind.equals("Reg8") && Op2Kind.equals("Reg8"))
					|| (Op1Kind.equals("Reg16") &&
Op2Kind.equals("Reg16"))) {
				code += "11";
				code += TabOfReg.getRegCode(Op1) +
TabOfReg.getRegCode(Op2);
				return code = code.substring(0, 2) +
from_2_to_16(code.substring(2, 6))
						+ from_2_to_16(code.substring(6, 10));
			}
			// если происходит запись непосредственного операнда в регистр
			else if ((Op1Kind.equals("Reg8") && Op2Kind.equals("Imm8"))
					|| (Op1Kind.equals("Reg16") &&
Op2Kind.equals("Imm8"))
					|| (Op1Kind.equals("Reg16") &&
Op2Kind.equals("Imm16"))) {
				code += "11000";
				code += TabOfReg.getRegCode(Op1);
				if (Op2.indexOf("OFFSET ") != -1) // если директива ОФФСЕТ
					return code = code.substring(0, 2) +
from_2_to_16(code.substring(2, 6))
							+ from_2_to_16(code.substring(6, 10))
							+ getOffset(Op2.substring(6,
Op2.length()).trim()).substring(2, 4)
							+ getOffset(Op2.substring(6,
Op2.length()).trim()).substring(0, 2);

				code = code.substring(0, 2) +
from_2_to_16(code.substring(2, 6)) + from_2_to_16(code.substring(6, 10));
				if (Op2Kind.equals("Imm8"))
					if (Op1Kind.equals("Reg16"))
						return code += HexSchak(Value, 2) + "00";
					else
						return code += HexSchak(Value, 2);
				else if (Op2Kind.equals("Imm16"))
					return code += HexSchak(Value, 4).substring(2, 4) +
HexSchak(Value, 4).substring(0, 2);
			}
			// если происходит базовая-индексная адресация в регистр
			else if ((Op1Kind.equals("Reg8") && Op2Kind.equals("Mem8"))
					|| (Op1Kind.equals("Reg8") &&
Op2Kind.equals("Mem16")
							|| (Op1Kind.equals("Reg16") &&
Op2Kind.equals("Mem8"))
							|| (Op1Kind.equals("Reg16") &&
Op2Kind.equals("Mem16")))) {
				code += "10";

				if (Op2.contains("BX") && Op2.contains("SI"))
					code += "000";
				else if (Op2.contains("BX") && Op2.contains("DI"))
					code += "001";
```

```
                        else if (Op2.contains("BP") && Op2.contains("SI"))
                              code += "010";
                        else
                              code += "011";
                        code += TabOfReg.getRegCode(Op1);
                        code = code.substring(0, 2) +
from_2_to_16(code.substring(2, 6)) + from_2_to_16(code.substring(6, 10));

                        if (Op2.indexOf('-') == -1 && Op2.indexOf('+') == -1) {
                              // this.CodeLen = CodeLen-1;
                              return code;
                        } else if (Op2Kind.equals("Mem8"))
                              return code += HexSchak(Value, 2);
                        else if (Op2Kind.equals(""))
                              return code += getOffset(Op2);
                        else
                              return code += HexSchak(Value, 4).substring(2, 4) +
HexSchak(Value, 4).substring(0, 2);
                  }
                  // если происходит базовая-индекстна адресация из регистра
                  else if ((Op2Kind.equals("Reg8") && Op1Kind.equals("Mem8"))
                              || (Op2Kind.equals("Reg8") &&
Op1Kind.equals("Mem16")
                                          || (Op2Kind.equals("Reg16") &&
Op1Kind.equals("Mem8"))
                                          || (Op2Kind.equals("Reg16") &&
Op1Kind.equals("Mem16")))) {
                        code += "10";
                        code += TabOfReg.getRegCode(Op2);
                        if (Op1.contains("BX") && Op1.contains("SI"))
                              code += "000";
                        else if (Op1.contains("BX") && Op1.contains("DI"))
                              code += "001";
                        else if (Op1.contains("BP") && Op1.contains("SI"))
                              code += "010";
                        else
                              code += "011";

                        code = code.substring(0, 2) +
from_2_to_16(code.substring(2, 6)) + from_2_to_16(code.substring(6, 10));

                        if (Op1Kind.equals("Mem8"))
                              return code += HexSchak(Value, 2);
                        else if (Op2Kind.equals(""))
                              return code += getOffset(Op2);
                        else
                              return code += HexSchak(Value, 4).substring(2, 4) +
HexSchak(Value, 4).substring(0, 2);
                  }
            }
            // если команда XCHG
            else if (Command.equals("XCHG")) {
                  code = TabOfCom.getCode(Command, Op1Kind, Op2Kind);
                  if ((Op1Kind.equals("Reg8") && Op2Kind.equals("Reg8"))
                              || (Op1Kind.equals("Reg16") &&
Op2Kind.equals("Reg16"))) {
                        code += "11";
                        code += TabOfReg.getRegCode(Op1) +
TabOfReg.getRegCode(Op2);
                        return code = code.substring(0, 2) +
from_2_to_16(code.substring(2, 6))
                                    + from_2_to_16(code.substring(6, 10));
```

```
                }
                // если происходит базовая-индексная адресация в регистр
                else if ((Op1Kind.equals("Reg8") && Op2Kind.equals("Mem8"))
                              || (Op1Kind.equals("Reg16") &&
Op2Kind.equals("Mem8"))
                              || (Op1Kind.equals("Reg16") &&
Op2Kind.equals("Mem16"))) {
                        code += "10";
                        code += TabOfReg.getRegCode(Op1);
                        if (Op2.contains("BX") && Op2.contains("SI"))
                              code += "000";
                        else if (Op2.contains("BX") && Op2.contains("DI"))
                              code += "001";
                        else if (Op2.contains("BP") && Op2.contains("SI"))
                              code += "010";
                        else
                              code += "011";

                        code = code.substring(0, 2) +
from_2_to_16(code.substring(2, 6)) + from_2_to_16(code.substring(6, 10));

                        if (Op2.indexOf('-') == -1 && Op2.indexOf('+') == -1) {
                              // this.CodeLen = CodeLen-1;
                              return code;
                        } else if (Op2Kind.equals("Mem8"))
                              return code += HexSchak(Value, 4);
                        else if (Op2Kind.equals(""))
                              return code += getOffset(Op2);
                        else
                              return code += HexSchak(Value, 4).substring(2, 4) +
HexSchak(Value, 4).substring(0, 2);
                }
                // если происходит базовая-индекстна адресация из регистра
                else if ((Op2Kind.equals("Reg8") && Op1Kind.equals("Mem8"))
                              || (Op2Kind.equals("Reg16") &&
Op1Kind.equals("Mem8"))
                              || (Op2Kind.equals("Reg16") &&
Op1Kind.equals("Mem16"))) {
                        code += "10";
                        code += TabOfReg.getRegCode(Op2);
                        if (Op1.contains("BX") && Op1.contains("SI"))
                              code += "000";
                        else if (Op1.contains("BX") && Op1.contains("DI"))
                              code += "001";
                        else if (Op1.contains("BP") && Op1.contains("SI"))
                              code += "010";
                        else
                              code += "011";

                        code = code.substring(0, 2) +
from_2_to_16(code.substring(2, 6)) + from_2_to_16(code.substring(6, 10));

                        if (Op1Kind.equals("Mem8"))
                              return code += HexSchak(Value, 4);
                        else if (Op2Kind.equals(""))
                              return code += getOffset(Op2);
                        else
                              return code += HexSchak(Value, 4).substring(2, 4) +
HexSchak(Value, 4).substring(0, 2);
                }
          }
          // если команда IMUL
```

```
            else if (Command.equals("IMUL")) {
                  code = TabOfCom.getCode(Command, Op1Kind, Op2Kind);
                  code += "11101";
                  if (Op1Kind.equals("Reg8") || Op1Kind.equals("Reg16"))
                        code += TabOfReg.getRegCode(Op1);

                  code = code.substring(0, 2) + from_2_to_16(code.substring(2, 6))
+ from_2_to_16(code.substring(6, 10));

            }
            // если команда LOOP
            else if (Command.equals("LOOP")) {
                  code = TabOfCom.getCode(Command, Op1Kind, Op2Kind);
                  if (from_16_to_10(getOffset(Op1)) - (Schak + CodeLen) < 0)
                        return code = code.substring(0, 2)
                                    + HexSchak(256 + from_16_to_10(getOffset(Op1))
- (Schak + CodeLen), 2);
                  else
                        return code = code.substring(0, 2) +
HexSchak(from_16_to_10(getOffset(Op1)) - (Schak + CodeLen), 2);
            }
            // если команда INT
            else if (Command.equals("INT")) {
                  code = TabOfCom.getCode(Command, Op1Kind, Op2Kind);
                  return code += HexSchak(Value, 2);
            }
            // если директива DB
            else if (Command.equals("DB")) {
                  if (Op2Kind.equals("Imm8"))
                        if (Op1.indexOf('H') != -1)
                              return code =
HexSchak(from_16_to_10(Op1.substring(0, Op1.length() - 1)), 2) + HexSchak(Value,
2);
                        else
                              return code = HexSchak(Integer.parseInt(Op1), 2) +
HexSchak(Value, 2);
                  else
                        return code = HexSchak(Value, 2);
            }
            // если директива DW
            else if (Command.equals("DW")) {
                  if (Op2Kind.equals("Imm8") || Op2Kind.equals("Imm16"))
                        if (Op1.indexOf('H') != -1)
                              return code =
HexSchak(from_16_to_10(Op1.substring(0, Op1.length() - 1)), 4).substring(2, 4)
                                          +
HexSchak(from_16_to_10(Op1.substring(0, Op1.length() - 1)), 4).substring(0, 2)
                                          + HexSchak(Value, 4).substring(2, 4) +
HexSchak(Value, 4).substring(0, 2);
                        else
                              return code = HexSchak(Integer.parseInt(Op1),
4).substring(2, 4)
                                          + HexSchak(Integer.parseInt(Op1),
4).substring(0, 2) + HexSchak(Value, 4).substring(2, 4)
                                          + HexSchak(Value, 4).substring(0, 2);
                  else
                        return code = HexSchak(Value, 4).substring(2, 4) +
HexSchak(Value, 4).substring(0, 2);
            }
            return code;
      }
```

```
	boolean HaveErr(int i) {
		if (ListOfErrors.equals(""))
			return false;

		String[] m = ListOfErrors.split("\\n+");

		for (int j = 0; j < m.length; j++)
			if (Integer.parseInt(m[j].split("[.]{1}")[0]) == i)
				return true;

		return false;
	}

	// приведение Счака к нормальному виду
	String HexSchak(int s, int n) {
		String HexS = Integer.toHexString(s).toUpperCase();
		int len = HexS.length();

		if (len > 4)
			HexS = HexS.substring(len - n, len);

		for (int j = 0; j < n - len; j++)
			HexS = "0" + HexS;

		return HexS;
	}

	// метод реакции на ошибку
	void Err(String err) {
		ErrorFlag = true;
		ListOfErrors += err;
	}

	// определение типа операнда
	String OpKind(String Op, int pass) {
		String Result = "";
		// если операнда нет изначально
		if (Op.equals(Result))
			return Result;
		// если регистр
		else if (TabOfReg.isReg(Op))
			Result = TabOfReg.getRegType(Op);
		// если число
		else if (Op.charAt(0) >= '0' && Op.charAt(0) <= '9' || Op.charAt(0) ==
'-') {
			boolean minus = false;
			if (Op.charAt(0) == '-') {
				minus = true;
				Op = Op.substring(Op.indexOf('-') + 1,
Op.length()).trim();
			}

			boolean decimal = true;
			if (Op.charAt(Op.length() - 1) == 'H') {
				Op = Op.substring(0, Op.indexOf('H')).trim();
				decimal = false;
			}

			if (!number(Op))
				return Result;

			if (!decimal)
```

```
                        Value = from_16_to_10(Op);
                  else
                        Value = Integer.parseInt(Op);

                  if (minus)
                        Value *= (-1);

                  // если больше двух байт
                  if (Value < -32768 || Value > 65535)
                        return Result;

                  // если в пределах одного байта
                  if (Value >= -128 && Value <= 255)
                        Result = "Imm8";
                  // иначе 2 байта
                  else
                        Result = "Imm16";
            }
            // если директива
            else if (TabOfDir.DirEn(Op)) {
                  // если директива ОФФСЕТ
                  if (Op.indexOf("OFFSET ") != -1) {
                        // если просмотр первый, то...
                        if (pass == 1)
                              Result = "Imm16"; // Op = Op.substring(6,
                                                      //
Op.length()).trim();
                        // если просмотр второй, то...
                        else if (isVariable(Op.substring(6, Op.length()).trim()))
                              Result = "Imm16";
                        else if (isLabel(Op.substring(6, Op.length()).trim()))
                              Result = "";
                        else
                              Result = "UnknVar";
                  } else // иначе ошибка, так как такой директивы тут быть не
должно
                        return Result;
                  // если адресация из памяти
            }
            // если метка
            else if (Op.charAt(0) == '[' && Op.charAt(Op.length() - 1) == ']') {
                  String n1 = "", n2 = "", n3 = "";
                  Op = Op.substring(Op.indexOf('[') + 1, Op.indexOf(']')).trim();
                  try {
                        n1 = Op.split("[-+]+")[0].trim();
                        n2 = Op.split("[-+]+")[1].trim();
                        n3 = Op.split("[-+]+")[2].trim();
                  } catch (java.lang.ArrayIndexOutOfBoundsException e) {
                        ErrorFlag = true;
                        return Result = "UnkAdr";
                  }

                  if ((n1.equals("BX") || n1.equals("BP")) && (n2.equals("SI") ||
n2.equals("DI"))
                              && OpKind(n3, pass).equals("Imm8"))
                        Result = "Mem8";
                  else if ((n1.equals("BX") || n1.equals("BP")) &&
(n2.equals("SI") || n2.equals("DI"))
                              && (OpKind(n3, 2).equals("Imm16") || OpKind(n3,
pass).equals("Label")
                                          || OpKind(n3, 2).equals("VAR"))) {
                        Result = "Mem16";
```

```
                        if (pass == 2 && OpKind(n3, 2).equals("VAR")) {
                              if (!getOffset(n3).equals("false"))
                                    Value = from_16_to_10(getOffset(n3));
                              else {
                                    Result = "UnknVar";
                                    ErrorFlag = true;
                              }
                        }
                  } else {
                        Result = "UnkAdr";
                        ErrorFlag = true;
                  }
            }
            // остальное - как переменная или метка
            else {
                  if (pass == 1)
                        Result = "Label";
                  else if (isVariable(Op))
                        Result = "VAR";
                  else if (isLabel(Op))
                        Result = "Label";
                  else
                        Result = "UnknVar";
            }

            return Result;
      }

      boolean number(String n) {
            String p = "0123456789ABCDEF";

            for (int j = 0; j < n.length(); j++)
                  if (p.indexOf(n.charAt(j)) == -1)
                        return false;

            return true;
      }

      String from_2_to_16(String s) {
            String[] h = { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A",
"B", "C", "D", "E", "F" };

            if (s.equals("0000"))
                  return h[0];

            if (s.equals("0001"))
                  return h[1];

            if (s.equals("0010"))
                  return h[2];

            if (s.equals("0011"))
                  return h[3];

            if (s.equals("0100"))
                  return h[4];

            if (s.equals("0101"))
                  return h[5];

            if (s.equals("0110"))
                  return h[6];
```

```
            if (s.equals("0111"))
                  return h[7];

            if (s.equals("1000"))
                  return h[8];

            if (s.equals("1001"))
                  return h[9];

            if (s.equals("1010"))
                  return h[10];

            if (s.equals("1011"))
                  return h[11];

            if (s.equals("1100"))
                  return h[12];

            if (s.equals("1101"))
                  return h[13];

            if (s.equals("1110"))
                  return h[14];

            return h[15];
      }

      int from_16_to_10(String g) {
            char[] h = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A',
'B', 'C', 'D', 'E', 'F' };
            int a = 1;
            int s = 0;

            for (int i = g.length() - 1; i >= 0; i--) {
                  for (int j = 0; j < 16; j++) {
                        if (g.charAt(i) == h[j]) {
                              s += (a * j);
                              a = a * 16;
                              break;
                        }
                  }
            }
            return (s);
      }

      String FindType(String operation) {
            for (int i = 0; i < ComAndDir.length; i++)
                  if (ComAndDir[i].equals(operation))
                        return Types[i];
            return "Unknown";
      }

      /*
       * проверка на соответствие команды из поступившего массива с командой из
       * эталонного
       */
      boolean OurComAndDir(String[] CAD, String[] NAD) {
            for (int i = 0; i < CAD.length; i++)
                  for (int j = 0; j < NAD.length; j++)
                        if (NAD[j].equals(CAD[i]))
                              return true;
```

```
	return false;
}

/*
 * проверка на соответствие команды из поступившего массива с командой из
 * эталонного
 */
boolean OurComAndDir(String CAD, String[] NAD) {
	for (int i = 0; i < NAD.length; i++)
		if (NAD[i].equals(CAD))
			return true;

	return false;
}

String DeleteComments(String s) {
	for (int j = 0; j < s.length(); j++) {
		if (s.charAt(j) == ';')
			return s.substring(0, j);
	}

	return s;
}

boolean DoubleVarOrLabel(String label) {
	for (int j = 0; j < SymbolTab.size(); j++)
		if (SymbolTab.get(j).Name.equals(label))
			return true;

	return false;
}

boolean isVariable(String var) {
	int j;

	for (j = 0; j < SymbolTab.size(); j++)
		if (SymbolTab.get(j).Name.equals(var))
			break;

	if (j == SymbolTab.size())
		return false;

	if (SymbolTab.get(j).Type.equals("VAR"))
		return true;

	return false;
}

boolean isLabel(String label) {
	int j;

	for (j = 0; j < SymbolTab.size(); j++)
		if (SymbolTab.get(j).Name.equals(label))
			break;

	if (j == SymbolTab.size())
		return false;

	if (SymbolTab.get(j).Type.equals("Label"))
		return true;
```

```
            return false;
      }

      String getOffset(String varlabel) {
            int j;
            String s = "false";

            for (j = 0; j < SymbolTab.size(); j++)
                  if (SymbolTab.get(j).Name.equals(varlabel)) {
                        s = SymbolTab.get(j).Offset;
                        break;
                  }

            return s;
      }
}
```