|  |  |  |
| --- | --- | --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  РОССИЙСКОЙ ФЕДЕРАЦИИ  Федеральное государственное бюджетное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Кафедра математического обеспечения вычислительных систем | |
| **декомпиляция машинного кода архитектуры x86-64 на язык ПРОГРАММИРОВАНИЯ c** | | |
| *Курсовая работа* | | |
|  | | |
|  | | |
|  | | Работу выполнил студент  группы ПМИ-4 3 курса механико-математического факультета  Пухов Н.А. |
|  | | Научный руководитель:  старший преподаватель кафедры МОВС  Лядова Л.Н.  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. |
|  | |  |
| Пермь 2021 | | |

**Аннотация**

текст.

Оглавление

Оглавление 3

Список сокращений 6

Введение 7

Глава 1. Анализ существующих решений 9

Глава 2. Теоретические основы декомпиляции 13

2.1. Постановка задачи декомпиляции 13

2.2. Конвейер декомпилятора 13

2.2.1. Декодирование 14

2.2.2. Трансляция инструкций языка ассемблер в P-код 14

2.2.3. Построение графа потока управления из P-кода 15

2.2.4. Построение абстрактного синтаксического дерева 16

2.2.5. Оптимизация 17

2.2.6. Символизация 18

2.2.7. Типизация 19

2.2.8. Генерация кода 19

2.3. Стандартное преобразование типов языка C 19

2.3.1. Арифметические преобразования типов 19

2.3.2. Преобразование указателей 21

2.3.3. Преобразование чисел со знаком в числа без знака 22

2.3.4. Преобразование чисел без знака в числа со знаком 22

2.3.5. Преобразование чисел с плавающей запятой 22

2.3.6. Преобразования между целочисленным типом и типом с плавающей запятой 22

2.4. Язык ассемблера архитектуры X86-64 22

2.4.1. Размеры операндов команд 23

2.4.2. Команды общего назначения 23

2.4.3. Команды блока MMX 25

2.4.4. Команды блока XMM 26

2.5. Маски выражений 30

2.6. Адресация элементов многомерных массивов 31

2.7. Линеаризация графа потока управления 32

2.7.1. Условное выражение if 32

2.7.2. Условное выражение if-else 32

2.7.3. Цикл while 33

Глава 3. Проектирование декомпилятора 34

3.1. Этап анализа 34

3.1.1. Диаграмма прецедентов 34

3.1.2. Диаграмма активностей 35

3.2. Этап проектирования 36

3.2.1. Создание классов 36

3.2.2. Диаграммы классов 38

3.3. Этап реализации 40

3.3.1. Использование сторонних библиотек 40

3.3.2. Диаграмма компонентов 41

3.4. Реализация основных алгоритмов 41

3.4.1. Алгоритм построения графа потока управления 41

3.4.2. Итерационный алгоритм вычисления типов выражений 42

3.4.3. Алгоритм оптимизации выражений, связанных с памятью 43

Глава 4. Тестирование и сравнение 45

4.1. Тестрование декомпилятора 45

4.2. Сравнение с Ghidra Decompiler 45

4.2.1. Векторные и математические операции 45

4.2.2. Интенсивная работа со стеком 47

Заключение 48

Библиографический список 49

Приложение A. Описание графических элементов основных существующих нотаций 55

A.1. Методология функционального моделирования *SADT* 55

А.2. Моделирование потоков данных в нотации *DFD* 56

А.3. Моделирование структур данных в нотации *ERD* 59

А.4. Моделирование функциональных требований в нотации диаграммы прецедентов 61

А.5. Моделирование бизнес-процессов в нотации диаграммы активностей 64

А.6. Моделирование поведения системы в нотации диаграммы последовательностей 65

А.7. Моделирование статической структуры в виде диаграммы классов 67

А.8. Построение моделей реализации в нотации диаграммы компонентов 69

А.9. Выводы 71

Приложение B. Описание бизнес-процесса выдачи книги 73

Приложение C. Техническое задание 74

**1.** **Введение** 75

**2.** **Основание для разработки** 75

**3.** **Назначение разработки** 75

4. Требования к программе или программному изделию 75

**2.** **Требования к программной документации** 80

**3.** **Технико-экономические показатели** 81

**4.** **Стадии и этапы разработки** 81

**5.** **Порядок контроля и приемки** 82

Список сокращений

DSM – Domain-Specific Modeling (предметно-ориентированное моделирование).

DSL – Domain-Specific Language (предметно-ориентированный язык).

ИС – информационная система.

ИТ, IT – информационные технологии.

Введение

На сегодняшний день существует большое количество разнообразных программ с открытым исходным кодом. Исходный код таких программ написан с использованием высокоуровневых языков программирования, среди которых C, C++, Java, Python и другие. Мотивация использования языков программирования с высоким уровнем абстракции обусловлена желанием иметь понятный, расширяемый код, который имеет возможность исполняться на разных видах операционных систем и архитектур процессора. Однако существует множество открытых программ, которые были написаны под определенную архитектуру процессора для конкретной операционной системы с использованием языка ассемблера. Среди такого рода программ можно выделить драйверы устройств или прошивки для микропроцессорной техники. Более того, программисты часто используют вставки кода на ассемблере в своих программах, написанных на высокоуровневых языках.

Использование ассемблера как низкоуровневого кода, который плохо понятен человеку, необходимо с целью повышения производительности программ. Более того, ассемблерный код имеет больше возможностей по оперированию устройством, чем код, написанный на языке высокого уровня. Но возникает большая проблема, связанная с чтением такого низкоуровневого кода другими людьми, которые могут даже не знать ассемблер. Решение данной проблемы состоит в использовании декомпилятора, который транслирует код из низкоуровневого представления в высокоуровневое. Среди существующих декомпиляторов можно выделить: IDA Hex-Rays Decompiler, Ghidra Decompiler, RetDec, Snowman, однако все они имеют определенные недостатки и не всегда генерируют высокоуровневый код в понятном человеку виде.

Объектом данного исследования является низкоуровневый код в ассемблерном или машинном представлении для архитектуры процессоров X86-64. Предмет исследования – представление такого кода в высокоуровневой форме.

Цель данной работы – декомпиляция кода на языке ассемблера архитектуры X86-64 в код высокоуровневого языка C в понятном человеку виде.

Для достижения поставленной цели должны быть решены следующие задачи:

1. Проанализировать существующие бесплатные программные реализации, которые позволяют декомпилировать код на языке ассемблера архитектуры X86-64 в высокоуровневое представление. Выявить критические проблемы этих реализаций, а именно понять, какие примеры низкоуровневого кода они не могут должным образом декомпилировать.

2. Разработать собственные пути решения выявленных проблем. Для этого создать свой собственный программный продукт или взять готовую реализацию и исправить эти проблемы в ней.

# Анализ существующих решений

В данной главе производится критический анализ существующих программных реализаций декомпиляции кода по следующим критериям:

1. Бесплатность и открытость исходного кода
2. Возможность встраивания
3. Высокая скорость работы
4. Декомпиляция заданного участка машинного кода
5. Использование промежуточного языка виртуальной машины
6. Высокое качество генерируемого кода
7. Полноценная поддержка векторных и математических операций

На данный момент разработано несколько декомпиляторов, которые, в частности, позволяют транслировать код с языка ассемблера или непосредственно машинный код архитектуры X86-64 в высокоуровневый код языка C. Среди популярных декомпиляторов можно выделить:

1. IDA Hex-Rays Decompiler. Данный декомпилятор является платным с закрытым исходным кодом. Он является плагином к программе IDA Pro. Имеет поддержку архитектур процессора x86-64, ARM, MIPS. Отличительная особенность – наличие отладчика, благодаря которому можно в динамике увидеть работу сгенерированного кода.
2. RetDec. Данный декомпилятор был представлен компанией Avast. Он является бесплатным с открытым исходным кодом и имеет поддержку множества архитектур процессоров, среди которых x86-64, ARM, MIPS. Использует промежуточный язык виртуальной машины LLVM. Имеет возможность восстановления иерархии классов и виртуальных таблиц. Генерирует код на языках C и Python. Основной недостаток – генерация кода уступает по качеству коду, который генерирует Ghidra Decompiler. Это в первую очередь проявляется в использовании плохо читаемых предопределенных функций и макросов вместо стандартных операторов языка C. Еще одним недостатком является отсутствие возможности декомпиляции заданного участка кода, а не всего программного образа.
3. Snowman. Особенностью данного декомпилятора является возможность трансляции кода в язык C++. Также он может быть легко встроен в любую программу и может декомпилировать заданный участок машинного кода. Однако он имеет такой же недостаток, что у декомпилятора RetDec, а именно плохо читаемый код во многих случаях.
4. Ghidra Decompiler. Данный декомпилятор был представлен агентством национальной безопасности США. Использует промежуточный язык PCode абстрактной виртуальной машины. Главное его отличие от всех остальных реализаций заключается в использовании виртуальной машины Java в качестве среды для выполнения основной программы Ghidra, хотя он сам написан на языке C++. Отсюда вытекает главный недостаток – низкая скорость работы декомпилятора. Еще один недостаток – это отсутствие возможности встроить данный декомпилятор в свою программу, а также декомпилировать заданный участок машинного кода. Однако главное преимущество данного декомпилятора – генерация им хорошо читаемого кода в большинстве случаев. Это достигается за счет более совершенных эвристических алгоритмов генерации высокоуровневого кода.

В табл. 1.1 представлено соотнесение декомпиляторов изложенным выше требованиям.

Таблица 1.1. Соотнесение декомпиляторов требованиям

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Требование** | **IDA Hex-Rays Decompiler** | **RetDec** | **Snowman** | **Ghidra Decompiler** |
| Бесплатность и открытость исходного кода | − | + | + | + |
| Возможность встраивания | − | + | + | − |
| Декомпиляция заданного участка машинного кода | − | − | + | − |
| Высокая скорость работы | + | + | + | − |
| Использование промежуточного языка виртуальной машины | − | + | − | + |
| Высокое качество генерируемого кода | + | − | − | + |
| Полноценная поддержка векторных и математических операций | − | − | − | − |

Важно отметить, что общая проблема всех перечисленных декомпиляторов – плохая работа с тем кодом, где преобладают векторные или математические операции и где наличествует пересылка данных из одного участка памяти в другой. Также большинство декомпиляторов могут анализировать и декомпилировать только цельные программные образы вместо заданных участков машинного кода.

Для решения двух названных выше основных проблем существует два варианта: улучшение готовой реализации или создание своей собственной с нуля. Рассмотрим первый вариант.

1. IDA Hex-Rays Decompiler. Данный декомпилятор не имеет открытого исходного кода, поэтому не может быть улучшен.
2. RetDec. Данный декомпилятор не подходит по многим причинам, в частности он не имеет возможности качественной декомпиляции заданного участка машинного кода. Таким образом улучшение данного декомпилятора приведёт к необходимости фундаментальных изменений, что нецелесообразно с точки зрения временных затрат.
3. Snowman. Данный декомпилятор не использует промежуточный язык виртуальной машины, что является критическим недостатком, поскольку не дает возможности виртуально вычислять те константные выражения, которые используют память компьютера. Таким образом улучшение данного декомпилятора приведёт к необходимости фундаментальных изменений, что нецелесообразно с точки зрения временных затрат.
4. Ghidra Decompiler. Данный декомпилятор не может встраиваться в другие программы, имеет привязку к виртуальной машине JVM, поэтому не может быть улучшен.

На основании вышеприведенных умозаключений по каждому декомпилятору было принято решение создать собственную программную реализацию с нуля.

# Теоретические основы декомпиляции

В данной главе описываются теоретические основы процесса декомпиляции и его основные этапы.

## Постановка задачи декомпиляции

Пусть - отображение, которое преобразует высокоуровневый код  на некотором языке программирования  в машинный код  некоторого процессора . Будем называть отображение  компилятором. Тогда отображение  будет являться идеальным декомпилятором.

Для компилятора, транслирующего код на языке C в машинный код архитектуры процессора X86-64, соответствующего идеального декомпилятора не может существовать. Это связано с тем, что в процессе компиляции безвозвратно теряется информация о названии переменных и функций, удаляются комментарии в коде. При использовании оптимизатора в компиляторе, теряется также первоначальная структура исходного кода.

Таким образом, задача сводится к построению приближенной функции , такой что . Функцию  назовем декомпилятором.

## Конвейер декомпилятора

Процесс декомпиляции можно представить в виде нескольких основных этапов преобразования данных из низкоуровневого кода в высокоуровневый код. На рис. 2.1 изображена схема конвейера декомпилятора, которая будет использована для решения поставленной задачи. Можно выделить несколько основных этапов:

1. Декодирование
2. Трансляция в P-код
3. Построение графа потока управления
4. Построение абстрактного синтаксического дерева
5. Оптимизация
6. Символизация
7. Типизация
8. Генерация кода

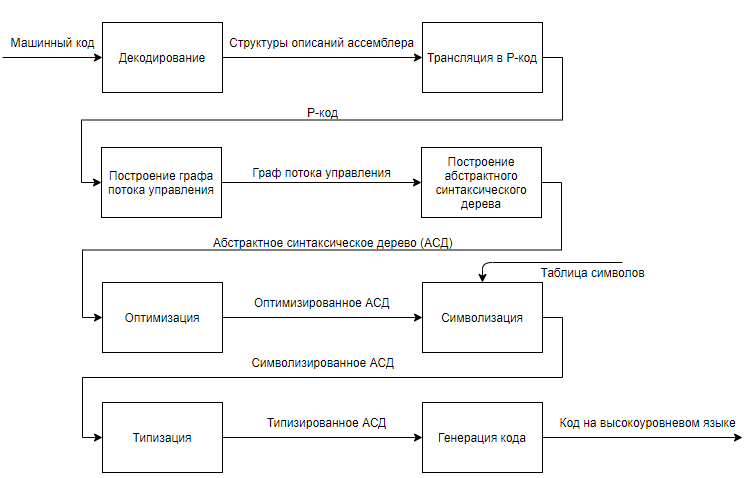


Рисунок 2.1. Конвейер декомпилятора

### Декодирование

Первым этапом является декодирование машинного кода, который представляет из себя последовательность байт, в структуры описания инструкций языка ассемблер. Структуры описания инструкций содержат информацию о типе инструкции, значениях и типах операндов, их количестве и т.д. Например, из инструкции “mov [rsp+0x10], 0x1” можно извлечь следующую информацию:

1. Тип инструкции – mov.
2. Первый операнд имеет указатель, который состоит из базового адреса, хранящегося в регистре “rsp”, и числового смещения “0x10”.
3. Второй операнд имеет числовое значение “0x1”.

### Трансляция инструкций языка ассемблер в P-код

Вторым этапом декомпиляции является преобразование структур описания инструкций языка ассемблер в аппаратно-независимый P-код. P-код – концепция аппаратно-независимого исполняемого кода в программировании, часто его определяют как «Ассемблер для гипотетического процессора», который служит необходимым промежуточным абстрактным слоем между аппаратно-зависимым языком ассемблер X86-64 и высокоуровневым языком C. P-код содержит минимальный набор команд, благодаря которому можно реализовать любой алгоритм, таким образом P-код имеет полноту по Тьюрингу. На рис. 2.2 изображен пример листинга P-кода для инструкций “AND” и “OR”.

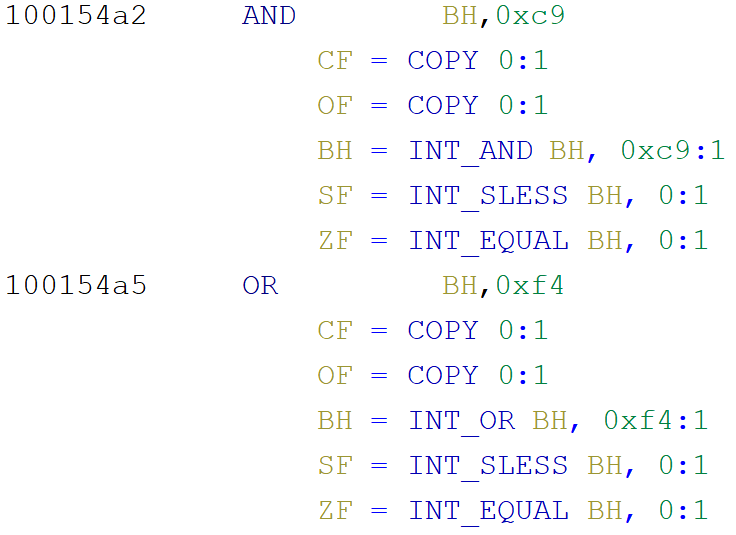


Рисунок 2.2. Пример P-кода

### Построение графа потока управления из P-кода

Третьим этапом является построение ориентированного графа потока управления из сгенерированного на предыдущем этапе листинга инструкций P-кода. В качестве вершин графа выступают блоки инструкций P-кода, которые не содержат команд условных или безусловных переходов за исключением последней команды, которая всегда осуществляет переход в другую область P-кода. В качестве рёбер ориентированного графа используются направления условных или безусловных переходов. Из одной вершины может выходить до двух ребер.

Вершина, из которого нет выходящих ребер, является конечной и всегда содержит последней инструкцией команду возврата, представляющую из себя безусловный переход в другую область программы. Конечных вершин может быть несколько. Вершина, в которую не входит ни одно ребро, является начальной вершиной в графе потока управления и всегда представлена одна. На рис. 2.3 изображен пример графа потока управления с двумя конечными вершинами. В блоках 1.1 и 1.2 содержится команда условного перехода, таким образом поток управления делится на два дочерних потока.

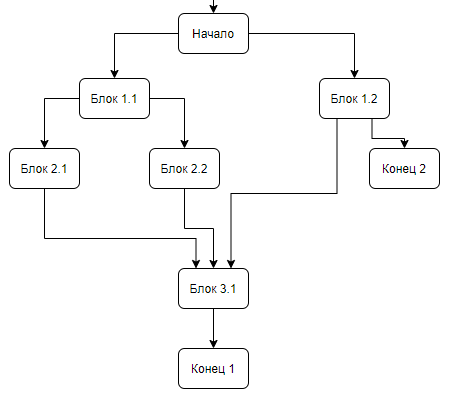


Рисунок 2.3. Пример графа потока управления

### Построение абстрактного синтаксического дерева

Четвертым этапом является построение абстрактного синтаксического дерева из построенного на предыдущем этапе графа потока управления. Абстрактное синтаксическое дерево – это промежуточное представление высокоуровневого кода в виде ориентированного дерева, в котором внутренние вершины сопоставлены с операторами языка программирования, а листья — с соответствующими операндами. Таким образом, листья являются пустыми операторами и представляют только переменные и константы. Пример абстрактного синтаксического дерева приведен на рис. 2.4.



Рисунок 2.4. Пример абстрактного синтаксического дерева

### Оптимизация

Пятый этап заключается в оптимизации полученного абстрактного синтаксического дерева, в частности сложных условных и константных выражений. Выражение – это набор численных операций, который преобразует входные числовые данные в некоторый результат, выраженный конкретным числовым значением. Оптимизация проводится с целью сделать сгенерированные выражения из P-кода более унифицированными и читаемыми для человека. Метрик читабельности выражения может быть перечислено несколько, но основная метрика – длина выражения: чем меньше длина выражения в символах, тем более читаемым оно является. Основные примеры оптимизации перечислены ниже:

1. Перемещение числовых операндов вправо: “5 + [rax]” → “[rax] + 5”.
2. Вычисление числовых операндов: “5 + 2” → “7”.
3. Раскрытие скобок: “([rax] + 5) \* 2” → “[rax] \* 2 + 10”.
4. Вычисление нулевых выражений: “[ax] & 0xFFFF0000” → “0”.

Стоит отметить, что вопреки метрике длины выражения оптимизация, заключающаяся в раскрытии скобок, часто увеличивает длину выражения. Однако, это необходимо для этапа типизации, на котором будет происходить процесс создания массивов, который требует полного раскрытия всех линейных выражений.

На этапе оптимизации также происходит оптимизация графа потока управления: несколько связанных блоков с простым условием перехода превращаются в один блок со сложным условием перехода. На рис. 2.5 представлен пример такой оптимизации.

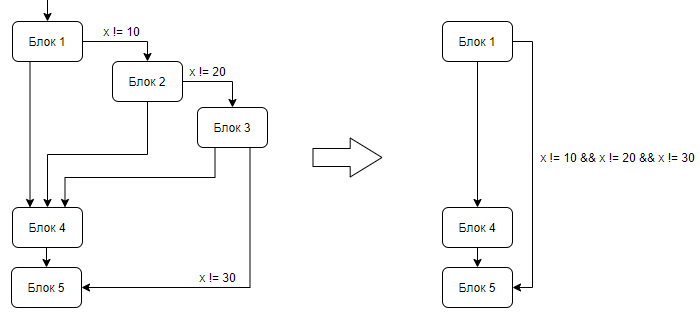


Рисунок 2.5. Пример абстрактного синтаксического дерева

Блоки 1, 2 и 3 имеют простое условие перехода. При этом данные блоки являются пустыми, они не должны содержать высокоуровневого кода. После оптимизации блоки 2 и 3 удаляются, а блок 1 приобретает новое условие, которое стало сложным.

### Символизация

Шестым этапом является процесс соотнесения узлам абстрактного синтаксического дерева символов. Символом в контексте теории компиляции является наименование и тип сущности, которой могут быть переменные или функции. Более подробно, сущностью могут являться:

1. Функции.
2. Параметры функции.
3. Локальные переменные (стековые или регистровые).
4. Глобальные переменные.

При разработке декомпилятора было принято следующее допущение: участок машинного кода будет декомпилирован полностью тогда и только тогда, когда будут определены все необходимые символы. Это означает, что для полной декомпиляции кода достаточно найти или вычислить символы для перечисленных выше сущностей в абстрактном синтаксическом дереве.

Если пользователем не были указаны символы, декомпилятор должен создать необходимые символы автоматически. Автоматически созданные символы будут иметь выбранные декомпилятором название и тип.

### Типизация

Седьмой этап заключается в итерационном прохождении по полученному на предыдущем этапе символизированному абстрактному синтаксическому дереву и вычислении для каждого узла этого дерева собственного типа. Правило вычисления типов узлов базируется на стандарте целевого высокоуровневого языка. Правила вычисления типов для языка C приведены в следующем параграфе.

### Генерация кода

Последний восьмой этап состоит в генерации высокоуровневого кода из типизированного абстрактного синтаксического дерева. Процесс генерации заключается в сопоставлении каждому узлу этого дерева некоторого текстового представления.

## Стандартное преобразование типов языка C

Этап типизации заключается в вычислении типов для каждого выражения. В данном разделе представлена информация по вычислению типов для выражений языка программирования C.

### Арифметические преобразования типов

Для арифметических выражений языка C существуют правила вычисления типов по умолчанию, которые приведены в табл. 2.1.

Таблица 2.1. Стандартные арифметические преобразования типов языка C

|  |  |
| --- | --- |
| **Выполненные условия** | **Преобразование** |
| Любой операнд имеет тип long double. | Другой операнд преобразуется в тип long double. |
| Предыдущее условие не выполнено, и любой из операндов имеет тип double. | Другой операнд преобразуется в тип double. |
| Предыдущие условия не выполнены, и любой из операндов имеет тип float. | Другой операнд преобразуется в тип float. |
| Предыдущие условия не выполнены (ни один из операндов не является операндом с плавающей запятой). | Операнды получают целочисленные продвижения следующим образом:  — Если любой из операндов имеет тип unsigned long, то другой операнд преобразуется в тип unsigned long.  — Если предыдущее условие не выполнено, и, если любой из операндов имеет тип long, а другой тип unsigned int, оба операнда преобразуются в тип unsigned long.  — Если предыдущие два условия не выполняются, и если любой из операндов имеет тип long, то другой операнд преобразуется в тип long.  — Если предыдущие три условия не выполняются, и если любой из операндов имеет тип unsigned int, то другой операнд преобразуется в тип unsigned int.  — Если ни одно из вышеперечисленных условий не выполняется, оба операнда преобразуются в тип int. |

В следующем коде, изображенном на рис. 2.5, демонстрируются правила преобразования, описанные в таблице.

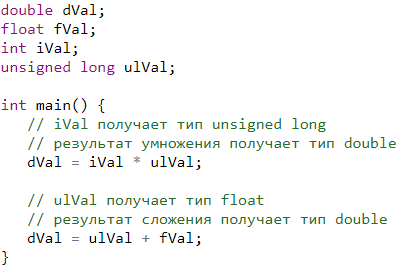


Рисунок 2.5. Код на языке C

Первая операция в примере представляет умножение двух целочисленных типов, iVal и ulVal. Условие выполнено, так как операнд не имеет типа float, а один операнд имеет тип unsigned int. Поэтому, другой операнд iVal преобразуется в тип unsigned int. Затем результат присваивается в переменную dVal. Один операнд имеет тип double, поэтому тип результата умножения становится double.

Второй оператор в предыдущем примере показывает добавление float и целочисленный тип: fVal и ulVal. Переменная ulVal преобразуется в тип float (третье условие в таблице). Результат сложения преобразуется в тип double (второе условие в таблице) и присваивается dVal.

### Преобразование указателей

Указатель на класс можно преобразовать в указатель на базовый класс в двух случаях:

1. Когда указанный базовый класс доступен и преобразование однозначно.
2. Указатель на класс можно преобразовать в указатель на базовый класс при использовании явного преобразования типов.

Указатели на тип void можно преобразовать в указатели на любой другой тип.

Указатель на функцию можно преобразовать в тип void\*, если тип void\* имеет достаточный размер для хранения указателя.

### Преобразование чисел со знаком в числа без знака

Объекты целочисленных типов со знаком можно преобразовывать в соответствующие типы без знака. При возникновении этих преобразований фактический битовый шаблон не изменяется. Однако изменяется интерпретация данных.

### Преобразование чисел без знака в числа со знаком

Объекты целочисленных типов без знака можно преобразовывать в соответствующие типы со знаком. Однако если значение без знака находится за пределами представимого диапазона типа со знаком, результат не будет иметь правильное значение.

### Преобразование чисел с плавающей запятой

Объект типа с плавающей запятой можно безопасно преобразовать в более точный тип с плавающей запятой, то есть без потери значимости. Например, преобразования из float в double double в long double являются безнадежными, а значение не изменяется.

### Преобразования между целочисленным типом и типом с плавающей запятой

Некоторые выражения могут вызывать преобразование объектов плавающего типа в целочисленные типы и наоборот. Если объект целочисленного типа преобразуется в тип с плавающей запятой, а исходное значение не может быть представлено точно, результатом является либо следующее выше, либо следующее меньшее представимое значение.

При преобразовании объекта с плавающего типа в целочисленный тип дробная часть усекается или округляется в сторону нуля. Например, число 1.4 преобразуется в число 1. Если усеченное значение выше наибольшего допустимого значения или меньше наименьшего представимого значения, результат будет неопределенным.

## Язык ассемблера архитектуры X86-64

В языке ассемблера архитектуры x86-64 команды можно разделить на следующие категории:

1. Команды общего назначения
2. Системные команды
3. Команды сопроцессора (x87 FPU)
4. Команды управления состоянием сопроцессора и SIMD
5. Команды технологии MMX
6. Команды расширения SSE
7. Команды расширения SSE2
8. Команды расширения SSE3
9. Команды расширения AVX

Наиболее часто используются команды общего назначения, команды технологии MMX, команды расширения SSE.

### Размеры операндов команд

В описаниях команд могут встречаться обозначения размеров операндов команд, которые представлены в табл. 2.2.

Таблица 2.2. Обозначения размеров операндов команд

|  |  |
| --- | --- |
| **Обозначение** | **Описание** |
| Byte | 8-битное целое (байт) |
| Word | 16-битное целое (слово) |
| DWord | 32-битное целое (двойное слово) |
| QWord | 64-битное целое (учетверенное слово) |
| Float | Вещественное число одинарной точности (32 бита) |
| Double | Вещественное число двойной точности (64 бита) |

### Команды общего назначения

Список команд общего назначения представлен в табл.

Таблица 2.3. Команды общего назначения

|  |  |
| --- | --- |
| **Команда** | **Описание** |
| MOV | Присваивание |
| CMOVxx | Условное присваивание |
| XCHG | Обмен значений |
| BSWAP | Перестановка байтов |
| XADD | Обмен и сложение |
| CMPXCHG | Сравнение и обмен |
| CMPXCHG8B | Сравнение и обмен 8 байтов |
| PUSH | Поместить значение в стек |
| POP | Взять значение из стека |
| PUSHA/PUSHAD | Поместить значения регистров общего назначения в стек |
| POPA/POPAD | Взять значения регистров общего назначения из стека |
| IN | Прочитать значение из порта ввода/вывода |
| OUT | Записать значение в порт ввода/вывода |
| CWD | Преобразовать Word в DWord |
| CDQ | Преобразовать DWord в QWord |
| CBW | Преобразовать Byte в Word |
| CWDE | Преобразовать Word в DWord в регистре eax |
| MOVSX | Присвоить и расширить с учетом знака |
| MOVZX | Присвоить и расширить нулевым значением |
| ADD | Сложение |
| ADC | Сложение с переносом |
| SUB | Вычитание |
| SBB | Вычитание с заемом |
| IMUL | Знаковое умножение |
| MUL | Беззнаковое умножение |
| IDIV | Знаковое деление |
| DIV | Беззнаковое деление |
| INC | Инкремент |
| DEC | Декремент |
| NEG | Смена знака |
| CMP | Сравнение |
| DAA | Десятичная коррекция после сложения |
| DAC | Десятичная коррекция после вычитания |
| AAA | ASCII коррекция после сложения |
| AAS | ASCII коррекция после вычитания |
| AAM | ASCII коррекция после умножения |
| AAD | ASCII коррекция перед делением |
| AND | Побитовое логическое И |
| OR | Побитовое логическое ИЛИ |
| XOR | Побитовое логическое Исключающее ИЛИ |
| NOT | Побитовое логическое НЕ |
| SAR | Арифметический сдвиг вправо |
| SHR | Логический сдвиг вправо |
| SAL/SHL | Арифметический/логический сдвиг влево |
| SHRD | Двойной сдвиг вправо |
| SHLD | Двойной сдвиг влево |
| ROR | Вращение вправо |
| ROL | Вращение влево |
| RCR | Вращение вправо через флаг переноса |
| RCL | Вращение влево через флаг переноса |
| BT | Проверка бита |
| BTS | Проверка и установка бита |
| BTR | Проверка и сброс бита |
| BTC | Проверка и инверсия бита |
| BSF | Проверка бита в прямом направлении |
| BSR | Проверка бита в обратном направлении |
| SETxx | Установить значение байта в зависимости от флага |
| TEST | Логическое сравнение |
| JMP | Безусловный переход |
| Jxx | Условный переход |
| JCXZ/JECXZ | Переход, если cx/ecx равен 0 |
| LOOP | Цикл со счетчиком в ecx |
| LOOPZ/LOOPE | Цикл со счетчиком в ecx и выходом при нуле / равенстве |
| LOOPNZ/LOOPNE | Цикл со счетчиком в ecx и выходом при не нуле / неравенстве |
| CALL | Вызов подпрограммы |
| RET | Возврат из подпрограммы |
| IRET | Возврат из прерывания |
| INT | Вызов программного прерывания |
| INTO | Вызов прерывания по переполнению |
| BOUND | Переход при выходе значения за заданные рамки |
| ENTER | Высокоуровневый вход в процедуру |
| LEAVE | Высокоуровневый выход из процедуры |

### Команды блока MMX

Команды блока MMX представлены в табл. 2.4

Таблица 2.4. Команды блока MMX

|  |  |
| --- | --- |
| **Команда** | **Описание** |
| EMMS | Подготовка сопроцессора к исполнению команд |
| MASKMOVQ | Запись байт в память из регистра MMX по маске (выборочная) |
| MOVD | Перемещение двойного слова |
| MOVNTQ | Запись 64 бит в память из регистра MMX (без использования кэш-памяти) |
| MOVQ | Переместить учетверенное слово (64 бит) |
| PACKSSWD | Упаковка со знаковым насыщением слов в байты |
| PACKSSDW | Упаковка со знаковым насыщением двойных слов в слова |
| PACKUSWB | Упаковка с беззнаковым насыщением слов в байты |
| PADDB | Сложение упакованных байт |
| PADDW | Сложение упакованных слов |
| PADDD | Сложение упакованных двойных слов |
| PADDSB | Сложение упакованных байт со знаковым насыщением |
| PADDSW | Сложение упакованных слов со знаковым насыщением |
| PADDUSB | Сложение упакованных байт с беззнаковым насыщением |
| PADDUSW | Сложение упакованных слов с беззнаковым насыщением |
| PAND | Упакованное логическое И |
| PANDN | Упакованное логическое И-НЕ |
| PAVGB | Упакованное среднее (байт) |
| PAVGW | Упакованное среднее (слов) |
| PCMPEQB | Сравнение на равенство упакованных байт |
| PCMPEQW | Сравнение на равенство упакованных слов |
| PCMPEQD | Сравнение на равенство упакованных двойных слов |
| PCMPGTB | Сравнение по условию "больше чем" упакованных байт |
| PCMPGTW | Сравнение по условию "больше чем" упакованных слов |
| PCMPGTD | Сравнение по условию "больше чем" упакованных двойных слов |
| PEXTRW | Извлечение 16-битного слова из регистра MMX по маске |
| PINSRW | Вставка 16-битного слова в регистр MMX |
| PMADDWD | Упакованное знаковое умножение слов операндов с последующим сложением промежуточных результатов в двойном слове |
| PMAXSW | Возврат максимальных упакованных знаковых слов |
| PMAXUB | Возврат максимальных упакованных беззнаковых байт |
| PMINSW | Возврат минимальных упакованных знаковых слов |
| PMINUB | Возврат минимальных упакованных беззнаковых байт |
| PMOVMSKB | Перемещение байтовой знаковой маски в целочисленный регистр |
| PMULHUW | Упакованное беззнаковое умножение слов с возвратом старшего слова результата |
| PMULHW | Упакованное знаковое умножение слов с возвратом старшего слова результата |
| PMULLW | Упакованное знаковое умножение слов с возвратом младшего слова результата |
| POR | Упакованное логическое ИЛИ |
| PSADBW | Суммарная разница значений пар беззнаковых упакованных байт |
| PSHUFW | Перераспределение упакованных слов |
| PSLLW | Логический сдвиг влево упакованных слов |
| PSLLD | Логический сдвиг влево упакованных двойных слов |
| PSLLQ | Логический сдвиг влево упакованных учетверенных слов |
| PSRAW | Арифметический сдвиг вправо упакованных слов |
| PSRAD | Арифметический сдвиг вправо упакованных двойных слов |
| PSRLW | Логический сдвиг вправо упакованных слов |
| PSRLD | Логический сдвиг вправо упакованных двойных слов |
| PSRLQ | Логический сдвиг вправо упакованных учетверенных слов |
| PSUBB | Вычитание упакованных байт |
| PSUBW | Вычитание упакованных слов |
| PSUBD | Вычитание упакованных двойных слов |
| PSUBSB | Вычитание упакованных байт со знаковым насыщением |
| PSUBSW | Вычитание упакованных слов со знаковым насыщением |
| PSUBUSB | Вычитание упакованных байт с беззнаковым насыщением |
| PSUBUSW | Вычитание упакованных слов с беззнаковым насыщением |
| PUNPCKHBW | Распаковка старших упакованных байт в слова |
| PUNPCKHWD | Распаковка старших упакованных слов в двойные слова |
| PUNPCKHDQ | Распаковка старших упакованных двойных слов в учетверенные слова |
| PUNPCKLBW | Распаковка младших упакованных байт в слова |
| PUNPCKLWD | Распаковка младших упакованных слов в двойные слова |
| PUNPCKLDQ | Распаковка младших упакованных двойных слов в учетверенные слова |
| PXOR | Упакованное логическое исключающее ИЛИ |

### Команды блока XMM

Команды блока XMM представлены в табл. 2.5

Таблица 2.5. Команды блока XMM

|  |  |  |
| --- | --- | --- |
| **Команда** | **Описание** | |
| ADDPS | Векторное сложение Float | |
| ADDSS | Скалярное сложение Float | |
| ANDNPS | Поразрядное логическое И-НЕ над Float | |
| ANDPS | Поразрядное логическое И над Float | |
| CMPPS | Векторное сравнение Float | |
| CMPSS | Скалярное сравнение Float | |
| COMISS | Скалярное упорядоченное сравнение с установкой EFLAGS | |
| CVTPI2PS | Преобразование двух упакованных DWord в два упакованных Float | |
| CVTPS2PI | Преобразование двух упакованных Float в два упакованных DWord | |
| CVTSI2SS | Скалярное преобразование знакового DWord в Float | |
| CVTSS2SI | Скалярное преобразование Float в знаковое DWord | |
| CVTTPS2PI | Преобразование усечением двух Float в два знаковых DWord | |
| CVTTSS2SI | Скалярное преобразование усечением Float в знаковое DWord | |
| DIVPS | Векторное деление Float | |
| DIVSS | Скалярное деление Float | |
| FXRSTOR | Восстановление без проверки наличия незамаскированных исключений с плавающей точкой состояния сопроцессора, целочисленного и потокового MMX-расширений из памяти | |
| FXSAVE | Сохранение состояния сопроцессора, целочисленного и потокового MMX-расширений в памяти | |
| LDMXCSR | Загрузка регистра состояния/управления MXCSR из памяти | |
| MAXPS | Вычисление максимальных упакованных значений Float | |
| MAXSS | Скалярное вычисление максимального значения Float | |
| MINPS | Вычисление минимальных упакованных значений Float | |
| MINSS | Скалярное вычисление минимального значения Float | |
| MOVAPS | Перемещение выровненных 128 бит | |
| MOVHLPS | Перемещение верхних упакованных значений Float в нижние | |
| MOVHPS | Перемещение верхних упакованных значений Float | |
| MOVLHPS | Перемещение нижних упакованных значений Float в верхние | |
| MOVLPS | Перемещение невыровненных нижних упакованных значений Float | |
| MOVMSKPS | Перемещение знаковой маски в целочисленный регистр | |
| MOVNTPS | Запись в память 128 бит из регистра XMM, минуя кэш | |
| MOVSS | Скалярное перемещение Float | |
| MOVUPS | Перемещение невыровненных упакованных Float | |
| MULPS | Векторное умножение Float | |
| MULSS | Скалярное умножение Float | |
| ORPS | Поразрядное логическое ИЛИ над Float | |
| RCPPS | Векторное вычисление обратных значений Float | |
| RCPSS | Скалярное вычисление обратного значения Float | |
| RSQRTPS | Векторное вычисление обратного значения квадратного корня Float | |
| RSQRTSS | Скалярное вычисление обратного значения квадратного корня Float | |
| SHUFPS | Перераспределение упакованных значений Float | |
| SQRTPS | Векторное вычисление квадратного корня Float | |
| SQRTSS | Скалярное вычисление квадратного корня Float | |
| STMXCSR | Сохранение регистра управления/состояния MXCSR в памяти | |
| SUBPS | Векторное вычитание Float | |
| SUBSS | Скалярное вычитание Float | |
| UCOMISS | Неупорядоченное скалярное сравнение с установкой флагов в EFLAGS | |
| UNPCKHPS | Перенос верхних упакованных значений Float с чередованием | |
| UNPCKLPS | Перенос нижних упакованных значений Float с чередованием | |
| XORPS | Поразрядное логическое исключающее ИЛИ над Float | |
| ADDPD | | Векторное сложение Double | |
| ADDSD | | Скалярное сложение Double | |
| ANDPD | | Поразрядное логическое И над Double | |
| ANDNPD | | Поразрядное логическое И-НЕ над Double | |
| CLFLUSH | | Сброс в память строки кэша, содержащей адрес | |
| CMPPD | | Векторное сравнение Double | |
| CMPSD | | Скалярное сравнение Double | |
| COMISD | | Скалярное сравнение упорядоченных значений и установка EFLAGS | |
| CVTDQ2PD | | Преобразование двух упакованных DWord в два Double | |
| CVTDQ2PS | | Преобразование четырех упакованных знаковых DWord в четыре упакованных Float | |
| CVTPD2DQ | | Преобразование двух упакованных Double в два упакованных DWord | |
| CVTPD2PI | | Преобразование двух упакованных Double в два упакованных DWord | |
| CVTPD2PS | | Преобразование двух упакованных Float в два упакованных Double | |
| CVTSD2SI | | Преобразование скалярного Double в DWord | |
| CVTSD2SS | | Преобразование скалярного Double в Float | |
| CVTSI2SD | | Преобразование скалярного DWord в Double | |
| CVTSS2SD | | Преобразование скалярного Float в Double | |
| CVTTPD2PI | | Преобразование усечением двух упакованных Double в два DWord | |
| CVTTPD2DQ | | Преобразование усечением двух упакованных Double в два DWord | |
| CVTTPS2DQ | | Преобразование усечением четырех упакованных Float в четыре знаковых DWord | |
| CVTTSD2SI | | Преобразование усечением скалярного Double в Float | |
| DIVPD | | Векторное деление Double | |
| DIVSD | | Скалярное деление Double | |
| LFENCE | | Упорядочить операции загрузки | |
| MASKMOVDQU | | Выборочная запись байт из источника в память | |
| MAXPD | | Вычисление максимальных упакованных значений Double | |
| MAXSD | | Скалярное вычисление максимального значения Double | |
| MFENCE | | Упорядочить операции загрузки и сохранения | |
| MINPD | | Вычисление минимальных упакованных значений Double | |
| MINSP | | Скалярное вычисление минимального значения Double | |
| MOVAPD | | Перемещение упакованных выровненных Double | |
| MOVD | | Перемещение DWord | |
| MOVDQA | | Перемещение выровненных 128 бит | |
| MOVDQU | | Перемещение невыровненных 128 бит | |
| MOVDQ2Q | | Перемещение младшего QWord | |
| MOVHPD | | Перемещение старшего упакованного Double | |
| MOVLPD | | Перемещение младшего упакованного Double | |
| MOVMSKPD | | Извлечение 2-битной знаковой маски упакованных Double | |
| MOVNTDQ | | Сохранение двух QWord в память без использования кэша | |
| MOVNTI | | Сохранение DWord из регистра общего назначения в память без использования кэша | |
| MOVNTPD | | Сохранение двух Double в память без использования кэша | |
| MOVQ | | Перемещение QWord | |
| MOVQ2DQ | | Перемещение QWord из MMX-регистра в младшее QWord XMM-регистра | |
| MOVSD | | Перемещение скалярного Double | |
| MOVUPD | | Перемещение невыровненных упакованных Double | |
| MULPD | | Векторное умножение Double | |
| MULSD | | Скалярное умножение Double | |
| ORPD | | Поразрядное логическое ИЛИ над Double | |
| PACKSSWB | | Упаковка со знаковым насыщением слов в байты | |
| PACKSSDW | | Упаковка со знаковым насыщением двойных слов в слова | |
| PACKUSWB | | Упаковка с беззнаковым насыщением слов в байты | |
| PADDB | | Сложение упакованных байт | |
| PADDW | | Сложение упакованных слов (Word) | |
| PADDD | | Сложение упакованных двойных слов (DWord) | |
| PADDQ | | Сложение учетверенных слов (QWord) | |
| PADDSB | | Сложение упакованных байт со знаковым насыщением | |
| PADDSW | | Сложение упакованных слов со знаковым насыщением | |
| PADDUSB | | Сложение упакованных байт с беззнаковым насыщением | |
| PADDUSW | | Сложение упакованных слов с беззнаковым насыщением | |
| PAND | | Упакованное логическое И | |
| PANDN | | Упакованное логическое И-НЕ | |
| PAUSE | | Улучшить выполнение цикла ожидания-занятости | |
| PAVGB | | Упакованное среднее байт | |
| PAVGW | | Упакованное среднее слов | |
| PCMPEQB | | Сравнение на равенство упакованных байт | |
| PCMPEQW | | Сравнение на равенство упакованных слов | |
| PCMPEQD | | Сравнение на равенство упакованных двойных слов | |
| PCMPGTB | | Сравнение по условию "больше чем" упакованных байт | |
| PCMPGTW | | Сравнение по условию "больше чем" упакованных слов | |
| PCMPGTD | | Сравнение по условию "больше чем" упакованных двойных слов | |
| PEXTRW | | Извлечение 16-битного слова из XMM-регистра | |
| PINSRW | | Вставка 16-битного слова в регистр XMM | |
| PMADDWD | | Упакованное знаковое умножение слов с последующим сложением промежуточных результатов в DWord | |
| PMAXSW | | Возврат максимальных упакованных знаковых слов | |
| PMAXUB | | Возврат максимальных упакованных беззнаковых байт | |
| PMINSW | | Возврат минмальных упакованных знаковых слов | |
| PMINUB | | Возврат минимальных упакованных беззнаковых байт | |
| PMOVMSKB | | Перемещение байтовой знаковой маски в целочисленный регистр | |
| PMULHUW | | Упакованное беззнаковое умножение слов с возвратом старших слов результата | |
| PMULHW | | Упакованное знаковое умножение слов с возвратом старших слов результата | |
| PMULLW | | Упакованное знаковое умножение слов с возвратом младших слов результата | |
| PMULUDQ | | Умножение DWord и сохранение результата в XMM-регистре. | |
| POR | | Упакованное логическое ИЛИ | |
| PSADBW | | Суммарная разница значений пар беззнаковых упакованных байт | |
| PSHUFD | | Перераспределение упакованных двойных слов | |
| PSHUFHW | | Перераспределение упакованных слов в старших DWord | |
| PSHUFLW | | Перераспределение упакованных слов в младших DWord | |
| PSLLDQ | | Побайтный сдвиг влево | |
| PSLLW | | Логический сдвиг слов влево | |
| PSLLD | | Логический сдвиг двойных слов влево | |
| PSLLQ | | Логический сдвиг учетверенных слов влево | |
| PSRAW | | Арифметический сдвиг слов влево | |
| PSRAD | | Арифметический сдвиг двойных слов влево | |
| PSRLDQ | | Побайтный сдвиг вправо | |
| PSRLW | | Логический сдвиг слов вправо | |
| PSRLD | | Логический сдвиг двойных слов вправо | |
| PSRLQ | | Логический сдвиг учетверенных слов вправо | |
| PSUBB | | Вычитание упакованных байт | |
| PSUBW | | Вычитание упакованных слов | |
| PSUBD | | Вычитание упакованных двойных слов | |
| PSUBQ | | Вычитание учетверенных слов | |
| PSUBSB | | Вычитание упакованных слов со знаковым насыщением | |
| PSUBSW | | Вычитание упакованных двойных слов со знаковым насыщением | |
| PSUBUSB | | Вычитание упакованных байт с беззнаковым насыщением | |
| PSUBUSW | | Вычитание упакованных слов с беззнаковым насыщением | |
| PUNPCKHBW | | Распаковка старших упакованных байт в слова | |
| PUNPCKHWD | | Распаковка старших упакованных слов в двойные слова | |
| PUNPCKHDQ | | Распаковка старших упакованных двойных слов в учетверенные слова | |
| PUNPCKHQDQ | | Распаковка старших упакованных учетверенных слов в двойные учетверенные слова | |
| PUNPCKLBW | | Распаковка младших упакованных байт в слова | |
| PUNPCKLWD | | Распаковка младших упакованных слов в двойные слова | |
| PUNPCKLDQ | | Распаковка младших упакованных двойных слов в учетверенные слова | |
| PUNPCKLQDQ | | Распаковка младших упакованных учетверенных слов в двойные учетверенные слова | |
| PXOR | | Упакованное логическое исключающее ИЛИ | |
| SHUFPD | | Перестановка упакованных значений Double | |
| SQRTPD | | Векторное вычисление квадратного корня Double | |
| SQRTSD | | Скалярное вычисление квадратного корня Double | |
| SUBPD | | Векторное вычитание Double | |
| SUBSD | | Скалярное вычитание Double | |
| UCOMISD | | Неупорядоченное скалярное сравнение Double с установкой флагов в EFLAGS | |
| UNPCKHPD | | Перенос верхних упакованных значений Double с чередованием | |
| UNPCKLPD | | Перенос нижних упакованных значений Double с чередованием | |
| XORDP | | Поразрядное логическое исключающее ИЛИ над Double | |

## Маски выражений

Каждое выражение при указании конкретных числовых входных данных выдает определенное числовое значение. Существует диапазон всех возможных значений, которые может выдать определенное выражение. Этот диапазон может быть выражен в виде битовой маски. Например, диапазон возможных значений регистра “al” в архитектуре процессора X86-64 составляет от 0 до 255, таким образом маска будет числом 0xFF. При этом выражение “al & 0xF” будет иметь диапазон значений от 0 до 16, поскольку старшие четыре бита регистра “al” будут иметь нулевое значение из-за битового умножения на число 0xF, и тогда итоговая маска данного выражения будет 0xF.

Маски выражений активно используются на этапе оптимизации выражений. Ниже приведены примеры, когда маски помогают значительно сократить длину выражения:

1. Побитовая дизъюнкция: “[al] | 0xFF” → “0xFF”
2. Побитовая конъюнкция: “[al] & 0xFF” → “[al]”

## Адресация элементов многомерных массивов

Массивы, в том числе многомерные, располагаются в памяти в виде непрерывной последовательности байт. У каждого массива есть базовый адрес, по которому располагается первый элемент массива.

Адресация любого элемента одномерного массива производится по формуле (2.1).

|  |  |
| --- | --- |
| , | (2.1) |

где A – массив, i – индекс элемента, B – базовый адрес массива, S – размер элемента массива.

Адресация любого элемента n-мерного массива производится по формуле (2.2).

|  |  |
| --- | --- |
| , | (2.2) |

где A – массив,  – индекс элемента в k-ом измерении, B – базовый адрес массива, – размер элемента массива в k-ом измерении,  - размер типа элемента массива.

Для n-мерного массива справедливо следующее соотношение (2.3).

|  |  |
| --- | --- |
|  | (2.3) |

## Линеаризация графа потока управления

Одной из основных задач декомпилятора является линеаризация графа потока управления, то есть представление графа потока управления в виде последовательности вложенных друг в друга условных выражений и циклов. Главная метрика качества алгоритма линеаризации – это количество операторов goto, которое должно быть сведено к минимуму. Другая немаловажная метрика – глубина вложенности условных выражений и циклов, и эта глубина также должна быть минимальна. Учёт двух перечисленных метрик позволит сделать структуру генерируемого кода наиболее читаемой для человека.

В следующих параграфах представлены соответствия между блоками графа потока управления и кодом на языке C.

### Условное выражение if

Условное выражение без ветки else представляется в виде трех блоков, связанных между собой так, как это показано на рис. 2.6.

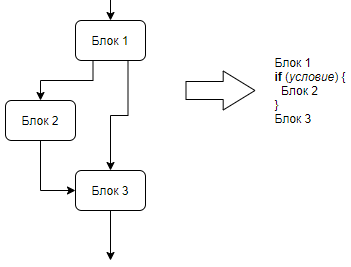


Рисунок 2.6. Конструкция if без ветки else

### Условное выражение if-else

Условное выражение с веткой else представляется в виде четырёх блоков, связанных между собой так, как это показано на рис. 2.7.

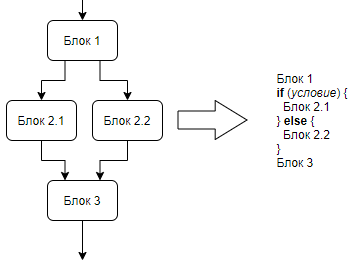


Рисунок 2.7. Конструкция if c веткой else

### Цикл while

Цикл while представляется в виде трех блоков, связанных между собой так, как это показано на рис. 2.8.

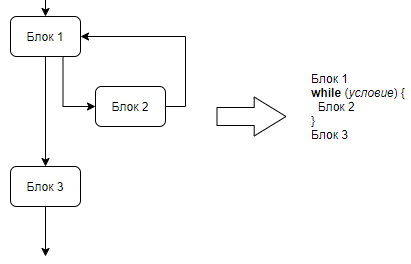


Рисунок 2.8. Конструкция цикла while

# Проектирование декомпилятора

Проектирование любой информационной системы является важной составляющей успешной разработки программного обеспечения. Этап проектирования помогает заложить правильный фундамент программы. По утверждению Стива Макконнелла в своей книге «Совершенный код» реализация этого этапа помогает избежать примерно 90% ошибок.

Было принято использовать объектный подход, поэтому будет использован язык моделирования UML.

UML – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем.

## Этап анализа

На этапе анализа необходимо выделить функциональные требования и роли для информационной системы «Декомпилятор», а также построить соответствующие диаграмму прецедентов и диаграмму активностей.

### Диаграмма прецедентов

Взаимодействовать с ИС будет пользователь. Пользователь предоставляет ИС участок машинного кода или код, написанный на языке ассемблер архитектуры X86-64, тем самым инициирует запуск конвейера декомпилятора. Конвейер декомпилятора состоит из нескольких обязательных этапов, которые были описаны в второй главе. Диаграмма прецендентов представлена на рис. 3.1.

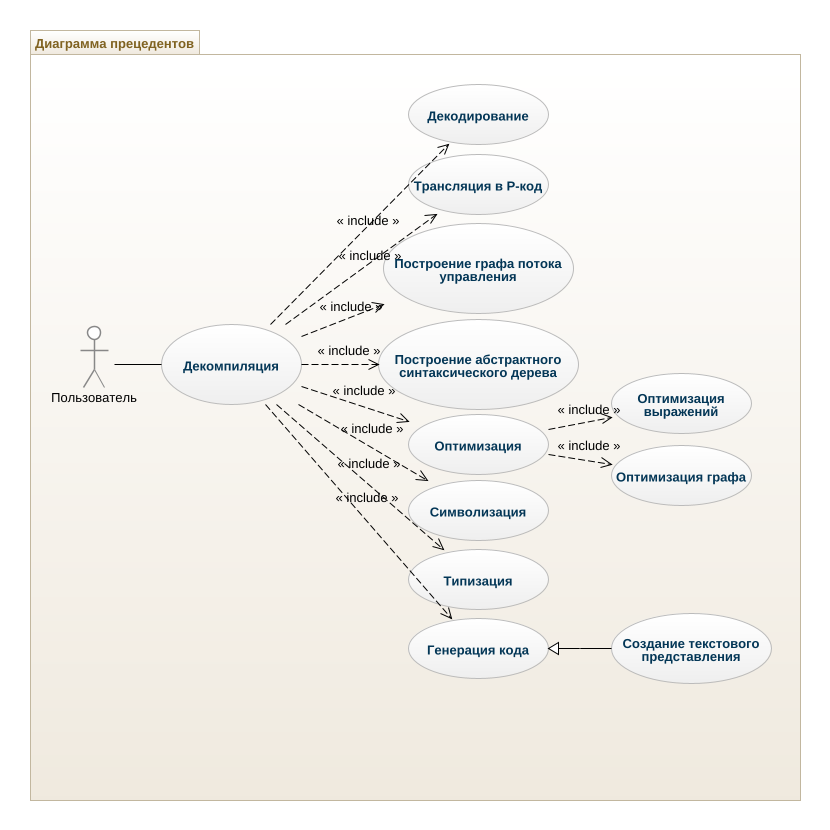


Рисунок 3.1. Диаграмма прецедентов

### Диаграмма активностей

Для понимания динамики процесса декомпиляции была использована диаграмма активностей, которая продемонстрирована на рис. 3.2. Необходимость в использовании цикла на диаграмме заключается в том, что построение абстрактного синтаксического дерева с последующими этапами его преобразования должно происходить многократно до тех пор, пока не будут распознаны сигнатуры для всех функций в виртуальных вызовах. Избежание многократности выполнения данных этапов процесса декомпиляции невозможно по причине того, что заранее невозможно идентифицировать сигнатуры функций в виртуальных вызовах, поскольку в данном случае явно не указывается адрес функции, которую необходимо вызвать.

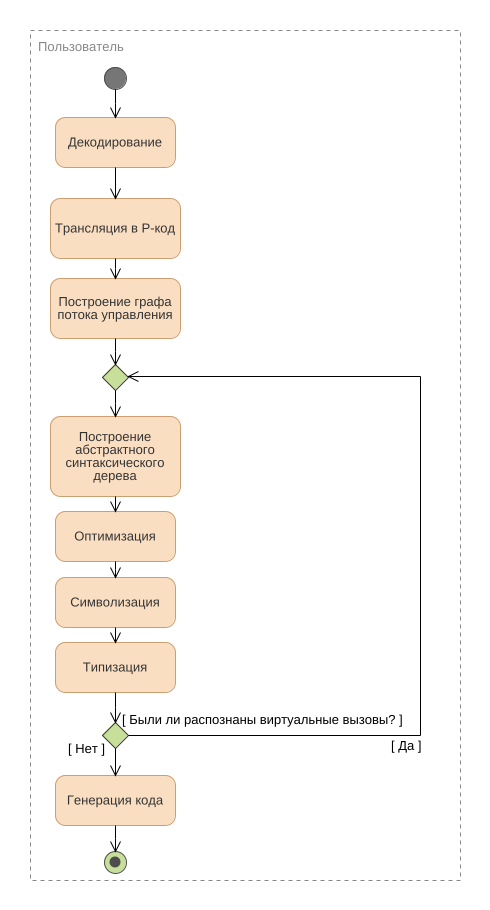


Рисунок 3.2. Диаграмма активностей

## Этап проектирования

Этап проектирования будет заключаться в создании диаграммы классов. Построенная иерархия классов должна соответствовать архитектуре конвейера декомпилятора, о котором было сказано во второй главе.

### Создание классов

В ходе проектирования были выделены следующие классы, каждый из которых имеет своё предназначение в ИС:

1. Decompiler. Главный класс, содержащий методы, которые позволяют инициировать запуск процесса декомпиляции, а также настроить основные параметры.
2. FunctionPCodeGraph. Класс, являющиеся графом потока управления. Содержит объекты класса PCodeBlock.
3. PCodeBlock. Класс, являющиеся блоком графа потока управления, который содержит листинг инструкций P-кода. Содержит две исходящие ссылки на другие объекты класса PCodeBlock, а также список объектов класса PCodeBlock, которые ссылаются на данный блок.
4. DecoderX86. Класс, инкапсулирующий логику по декодированию инструкций машинного кода архитектуры X86-64. Данный класс распознает тип и свойства машинной инструкции, далее начинает процесс построения списка соответствующих инструкций P-кода.
5. RegisterFactoryX86. Класс, являющиеся фабрикой регистров под архитектуру X86-64.
6. BitMask64. Класс, являющиеся 64-битной маской выражения. Описывает диапазон возможных значений. Содержит операции дизъюнкции, конъюнкции, сравнения.
7. ImageAnalyzer. Класс, содержащий алгоритм построения графа потока управления по списку инструкций P-кода. Имеет возможность работать в режиме потока байт машинного кода, то есть строить граф потока управления по ходу получения машинного кода, транслируя его одновременно с использованием объекта класса DecoderX86 в P-код.
8. InstructionInterpreter. Класс, являющиеся интерпретатором P-кода. Принимая поток инструкций P-кода объект данного класса строит абстрактное синтаксическое дерево.
9. DecompiledCodeGraph. Класс, который представлен в виде графа потока управления, но который содержит выражения абстрактного синтаксического дерева.
10. GraphBlockLinker. Класс, позволяющий сделать символьное связывание блоков P-кода графа потока управления, а именно разрешить все внешние символов каждого блока, которые создаются в момент интерпретации P-кода по запросу определенного регистра.
11. ExprOptimization. Класс, позволяющий произвести оптимизацию заданного выражения по правилам, которые кратко описаны во второй главе.
12. GraphOptimization. Класс, позволяющий произвести оптимизацию структуры графа потока управления. В частности, эта оптимизация заключается в удалении и замене блоков с простым условием на блоки со сложным условием, тем самым обеспечивая минимальную вложенность условных конструкций.
13. SdaBuilding. Класс, позволяющий произвести символизацию абстрактного синтаксического дерева.
14. SdaDataTypesCalculating. Класс, позволяющий произвести типизацию абстрактного синтаксического дерева.
15. SdaGraphMemoryOptimization. Класс, позволяющий произвести оптимизацию выражений, связанных с памятью, за счет удаления лишних присваиваний и оптимизации метрики длины выражений, что делает код более читаемым.
16. LinearViewSimpleOutput. Класс, позволяющий произвести генерацию итогового высокоуровневого кода на языке C.

Также были выделены вспомогательные классы, описание которых здесь не приводится.

### Диаграмма классов

Подробное содержание каждого выделенного класса представлено на рис. 3.3 и рис. 3.4.

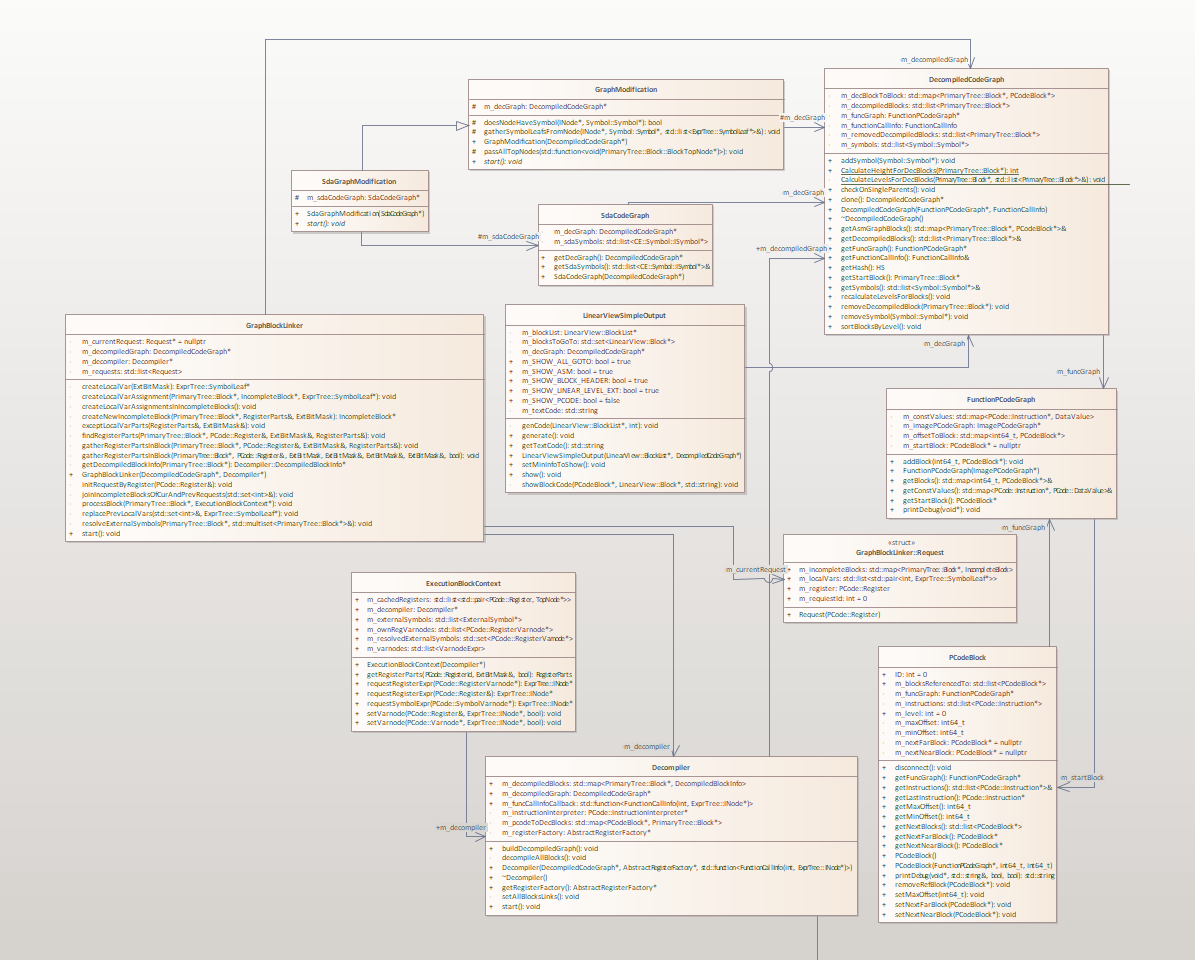


Рисунок 3.3. Диаграмма классов (первая часть)

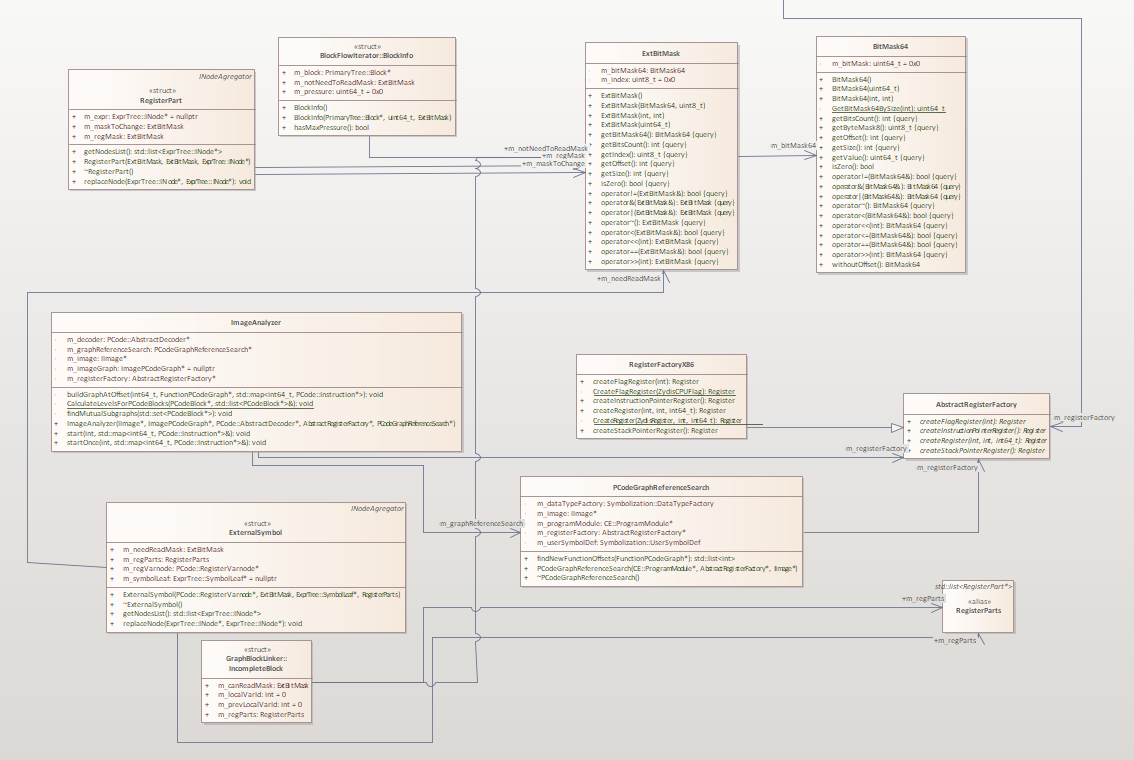


Рисунок 3.4. Диаграмма классов (вторая часть)

## Этап реализации

На этапе реализации мы должны реализовать ИС и понять, как она будет выглядеть в виде цельной системы.

### Использование сторонних библиотек

Для ускорения процесса разработки было принято решение использовать сторонние библиотеки:

1. SDA. Данная библиотека нужна для удобного манипулирования типами языка C.
2. ZydisDisassembler. Данная библиотека нужна для реализации процесса декодирования машинного кода архитектуры X86-64 в структуры описаний ассемблера.
3. Asmjit/Asmtk. Данный набор библиотек необходим для трансляции текстового кода на языке ассемблер архитектуры X86-64 в машинный код.

### Диаграмма компонентов

Диаграмма компонентов, представленная на рис. 3.5, позволяет графически изобразить структуру будущего приложения.

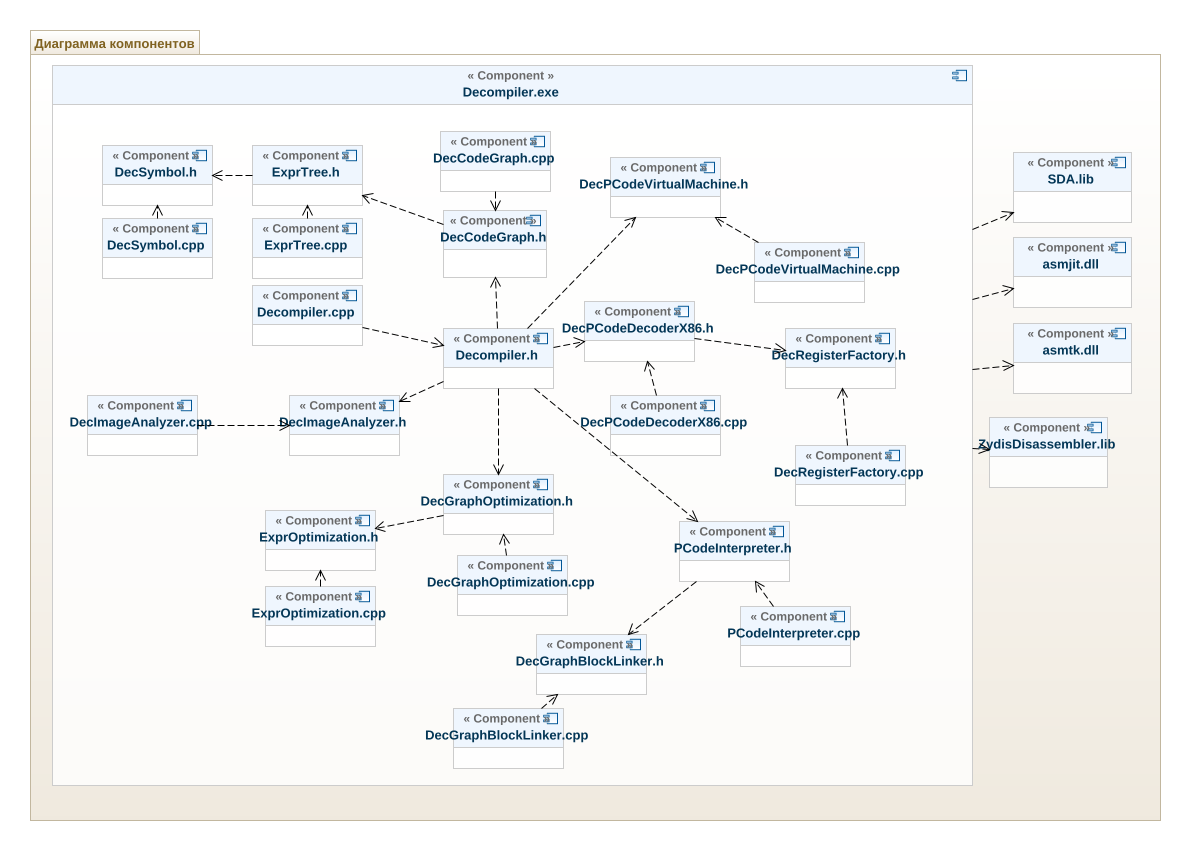


Рисунок 3.5. Диаграмма компонентов

## Реализация основных алгоритмов

В данном параграфе перечислены основные алгоритмы, которые были использованы для процесса декомпиляции.

### Алгоритм построения графа потока управления

Был самостоятельно разработан алгоритм, позволяющий на основе списка инструкций P-кода построить граф блоков, связанных между собой командами условного и безусловного перехода.

Краткое описание алгоритма:

1. Указывается текущее начальное смещение относительно базового адреса в памяти машинного кода, который необходимо декомпилировать. Также создается начальный блок P-кода, в который будут добавляться инструкции.
2. Декодируется инструкция, которая располагается по текущему смещению. Результат декодирования, который представлен в виде списка примитивных инструкций P-кода, сохраняется в карте вида “смещение – инструкция”. Если по текущему смещению невозможно декодировать инструкцию, то текущее смещение приобретает новое значение из непустого списка смещений, которые необходимо посетить. После этого, осуществляется переход на шаг 1. Если этот список оказывается пустым, алгоритм прекращает свою работу.
3. В множестве посещенных смещений сохраняется текущее смещение.
4. Осуществляет итерация по списку инструкций P-кода, полученному на шаге 2, при этом каждая инструкция добавляется в блок P-кода, который находится по текущему значению. Если встретилась инструкция безусловного перехода, то текущее смещение изменяется на то, что находится в первом операнде инструкции данного перехода. Если встретилась инструкция условного перехода, то смещение, которое находится в первом операнде инструкции, добавляется в список смещений, которые надо посетить позже.
5. По текущему смещению создается новый блок P-кода.
6. Переход на шаг 1.

Данный алгоритм реализован в классе ImageAnalyzer.

### Итерационный алгоритм вычисления типов выражений

После построения символизированного абстрактного синтаксического дерева (далее АСД), декомпилятор должен вычислить для каждого узла этого дерева тип данных языка программирования C. Для этого был самостоятельно разработан алгоритм вычисления типов.

Краткое описание алгоритма:

1. Осуществляется проход по всем узлам АСД снизу вверх.
2. Если встретилась бинарная операция, то по правилам преобразования типов данных языка C, которые описаны во второй главе, назначается определенный тип для данной операции, а также происходит явное или неявное приведение к этому типу операндов этой операции. Если среди этих операндов оказался символ, созданный автоматически, то изменяется его тип данных без привидения типа.
3. Если были сделаны новые приведения типов, то осуществляется переход на шаг 1, иначе алгоритм завершает свою работу.

Данный алгоритм реализован в классе SdaDataTypesCalculating.

Пример работы данного алгоритма изображен на рис. 3.6. В данном примере показывается вычисление типа float для нескольких переменных. После этого меняется вид кода, в частности отображение вещественных чисел.

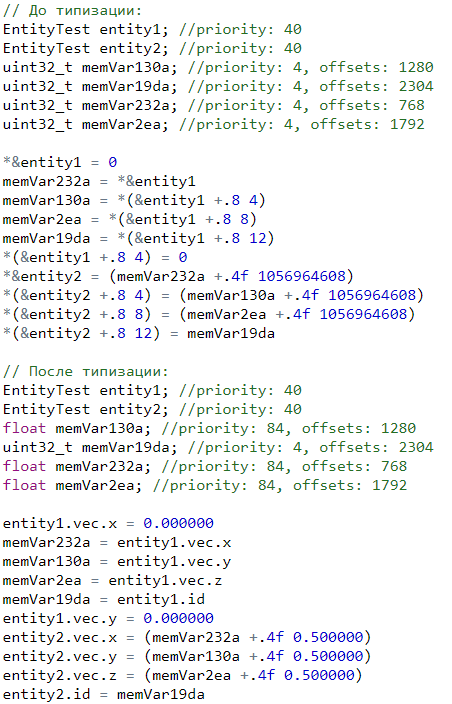


Рисунок 3.6. Пример работы алгоритма вычисления типов

### Алгоритм оптимизации выражений, связанных с памятью

Это основной самостоятельно разработанный алгоритм, позволяющий сделать код значительно более читаемым за счет удаления лишних присваиваний, связанных с пересылкой данных из одного участка памяти другой.

Краткое описание алгоритма:

1. Для каждого блока кода абстрактного синтаксического дерева осуществляется итерация по всем выражениям.
2. Если встретилось выражение присваивания, то определяется, происходит чтение из памяти или запись в память значения. Если чтение из памяти в целевую переменную, то создается снимок значения и проекция этого снимка на указанную в присваивании область памяти, а также происходит связывание этой области памяти с целевой переменной. Если запись в память, то создается в этой области памяти новое значение.
3. Если встретилось выражение, в котором происходит чтение из памяти, то просматривается область памяти, в которую были записаны снимки значений, и происходит замена выражения чтения памяти на этот снимок.

Данный алгоритм реализован в классе SdaGraphMemoryOptimization.

Пример работы данного алгоритма представлен на рис. 3.7. В результате оптимизации были удалены три операции присваивания, код стал меньше в размере и приобрел более читаемый вид.

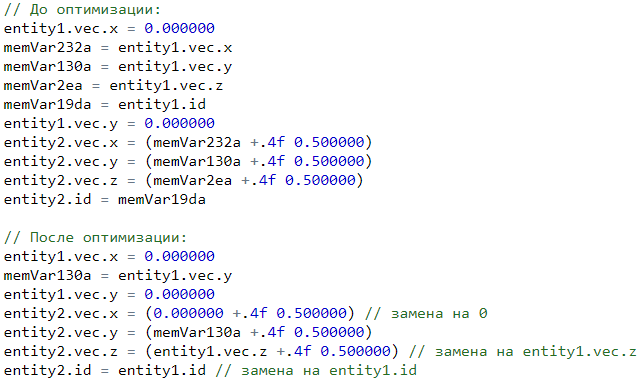


Рисунок 3.7. Пример оптимизации выражений памяти

# Тестирование и сравнение

Тестирование является очень важным этапом в разработке программного обеспечения, поскольку позволяет выявить большинство ошибок еще до того, как разработанный продукт появится в доступе у конечных пользователей.

## Тестрование декомпилятора

Тестирование разработанного декомпилятора было произведено вручную на 20 разных примерах машинного кода. Список данных примеров можно найти в файле DecSampleTests.cpp.

## Сравнение с Ghidra Decompiler

Сравнение будет произведено с декомпилятором Ghidra Decompiler в силу его большой популярности, а также потому, что он выдаёт в среднем более качественный код, чем остальные декомпиляторы, которые были рассмотрены в первой главе.

### Векторные и математические операции

Разработанный декомпилятор в отличие от Ghidra Decompiler успешно справляется с задачей декомпиляции сложных математических векторных выражений.

На рис. 4.1 показан фрагмент результата декомпиляции функции умножения двух матриц определенного вида. Разработанный декомпилятор не использует в своем выводе ничего лишнего, в то время как Ghidra Decompiler использует предопределенный макрос SUB164, операции сдвига, лишние локальные переменные, что делает код крайне нечитаемым.



Рисунок 4.1. Фрагмент результата декомпиляции умножения матриц

На рис. 4.2 приведен ещё один пример декомпиляции математической функции. Разработанный декомпилятор снова не использует в своем выводе ничего лишнего, в то время как Ghidra Decompiler использует множество предопределенных макросов и неизвестных XMM регистров.

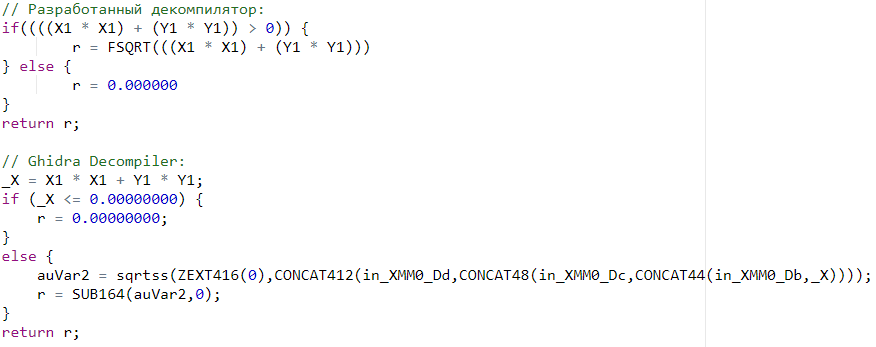


Рисунок 4.2. Фрагмент результата декомпиляции математической функции

### Интенсивная работа со стеком

Разработанный декомпилятор в отличие от Ghidra Decompiler также хорошо справляется с машинным кодом, где происходит интенсивная работа со стеком через векторные SSE инструкции.

На рис. 4.3 изображено сравнение результатов декомпиляции разработанного декомпилятора и Ghidra Decompiler для участка машинного кода, где происходят сложные векторные операции с памятью стека. В данном случае Ghidra Decompiler сделал неполный вывод всего кода, смешав его к тому же с малопонятными макросами и операциями побитового сдвига.

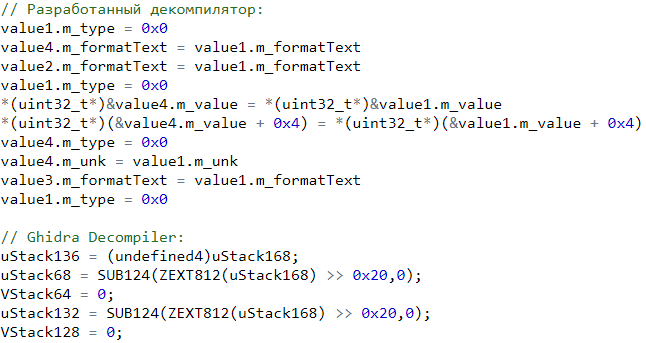


Рисунок 4.3. Фрагмент результата декомпиляции

Заключение

В ходе выполненной работы был разработан декомпилятор, который.

Библиографический список

1. *Брыксин Т.А*. Платформа для создания специализированных визуальных сред разработки программного обеспечения : дис. … канд. техн. наук: 05.13.11 / *Брыксин Тимофей Александрович*. СПб., 2016. – 159 с.
2. *Замятина Е.Б*. Мультиязыковое моделирование с использованием DSM-платформы MetaLanguage / *Е.Б. Замятина, Л.Н. Лядова, А.О. Сухов* // М: Информация и связь, 2013. – Вып. 5. – С. 11-14.
3. *Клековкин Г.А.* Геометрическая теория графов / *Г.А. Клековкин, Л.П. Коннова, В.В. Коннов* // М: Издательство Юрайт, 2018. – 240 с.
4. *Кознов Д.В.* Основы визуального моделирования. Учебное пособие / *Д.В. Кознов* // М: Интернет-Университет Информационных Технологий, 2007. – 248 с.
5. *Лядова Л.Н.* Языковой инструментарий системы MetaLanguage / *Л.Н. Лядова, А.О. Сухов* // Математика программных систем: межвуз. сб. науч. ст. Пермь: Изд-во Перм. гос. ун-та, 2008. – Вып. 5. – С. 40-51.
6. *Лядова Л.Н.* Гиперграфы с полюсами как основа создания редакторов визуальных языков / *А.О. Сухов, Л.Н. Лядова, С.А. Порязов* // Математика программных систем: межвуз. сб. науч. тр. Пермь: Перм. гос. нац. исслед. ун-т., 2018. – Вып. 15. – С. 97-104.
7. *Павлинов А.А.* Комплекс средств разработки проблемно-ориентированных визуальных языков / *А.А. Павлинов, Д.В. Кознов, А.Ф. Перегудов, Д.Ю. Бугайченко, А.С. Казакова, Р.И. Чернятчик, Т.А. Фесенко, А.Н. Иванов* // Вестник Санкт-Петербургского университета. – СПб, 2007. – Сер. 10, Вып. 1 – С. 86-96.
8. *Платонова М.В.* Графическое моделирование на базе Eclipse в системах управления: бакалаврская работа. Санкт-Петербургский государственный университет, Санкт-Петербург, 2019.
9. *Поляков В.А.* Средства задания исполнимой семантики визуальных языков в системе QReal: дипломная работа. Санкт-Петербургский государственный университет, Санкт-Петербург, 2013.
10. *Самохвалов Э.Н.* Использование метаграфов для описания семантики и прагматики информационных систем / *Э.Н. Самохвалов, Г.И. Ревунков, Ю.Е. Гапанюк* // Вестник МГТУ им. Н.Э. Баумана. Сер. «Приборостроение». Москва: МГТУ им. Н.Э. Баумана, 2015. – Вып. 1. – С. 83-99.
11. *Стручков И.В.* Формализм для описания программных систем и вычислительных процессов циклической параллельной обработки данных реального времени / *И.В. Стручков, В.М. Ицыксон* // Информационно-управляющие системы. СПб: ФГАОУ ВО «Санкт-Петербургский государственный университет аэрокосмического приборостроения», 2006. – Вып. 2. – С. 8-13.
12. *Сухов А.О.* Инструментальные средства создания визуальных предметно-ориентированных языков моделирования / *А.О. Сухов* // Фундаментальные исследования. – 2013. – № 4-4 – С. 848-852.
13. *Сухов А.О*. Использование графовых грамматик для трансформации моделей / А.О. Сухов, А.П. Серый // Материалы конференции "CSEDays 2012". Екатеринбург: Изд-во Урал. ун-та, 2012. – С. 48-55.
14. *Сухов А.О.* Методы трансформации визуальных моделей / *А.О. Сухов* // Технологии разработки информационных систем ТРИС-2012. Материалы конференции. Таганрог: Издательство Технологического института ЮФУ, 2012. – С.120-124.
15. *Сухов А.О.* Разработка инструментальных средств создания визуальных предметно-ориентированных языков : дис. … канд. физ.-мат. наук: 05.13.11 / *Сухов Александр Олегович*. М., 2013. 256 с.
16. *Сухов А.О.* Сравнение систем разработки визуальных предметно-ориентированных языков / *А.О. Сухов* // Математика программных систем: межвузовский сборник научных статей. Пермь: Пермский государственный национальный исследовательский университет, 2012. – Вып. 9. – С. 84-111.
17. *Сухов А.О.* Теоретические основы разработки DSL-инструментария с использованием графовых грамматик / *А.О. Сухов* // Информатизация и связь. – 2011. – № 3. – С. 35-37.
18. *Терехов А.Н.* Архитектура среды визуального моделирования QReal/ *А.Н. Терехов, Т.А. Брыксин, Ю.В. Литвинов, К.К. Смирнов, Г.А. Никандров, В.Ю. Иванов, Е.И. Такун*// Системное программирование. СПб: Издательство Санкт-Петербургского государственного университета, 2009. – Вып. 4. – С. 172-197.
19. *Филатов Д.Ю.* Разработка редактора визуальных моделей, основанного на *P*-графах / *Д.Ю. Филатов, Л.Н. Лядова* // В кн.: Технологии разработки информационных систем (ТРИС-2017): 104 Материалы VIII Международной научно-технической конференции. Ростов н/Д : Южный федеральный университет, 2017. С. 113-118.
20. *Шаврин С.М.* Моделирование и проектирование информационных систем: учеб.-метод. пособие / *С.М. Шаврин, Л.Н. Лядова, С.И. Чуприна* // Перм. гос. ун-т. – Пермь, 2007. – 154 с.
21. *Balasubramanian D.* The Graph Rewriting and Transformation Language: GReAT / *D. Balasubramanian, A. Narayanan, C. Buskirk, G. Karsai* // Proceedings of the Third International Workshop on Graph Based Tools, 2006. – Vol. 1. – 8 p.
22. *Basu A.* Graphs, Hypergraphs, and Metagraphs / *A. Basu, R. Blanning* // Metagraphs and Their Applications. New York: Springer US, 2007. – Vol. 15. – 173 p.
23. *Bezivin, J*. An Introduction to the ATLAS Model Management Architecture / *J. Bezivin, F. Jouault, D. Touzet* // Research Report № 05.01. – LINA, 2005. – 24 p.
24. *Courcelle B.* Recognizable Sets of Graphs, Hypergraphs and Relational Structures: A Survey / *B. Courcelle* // Developments in Language Theory. International Book Series «Lecture Notes in Computer Science», 2005. – Vol. 3340. – P. 1-11.
25. *Dean T.* Survey of Graph Rewriting applied to Model Transformations / *F. Parra, T. Dean* // Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development, 2014. – P. 431-441.
26. *Ehrig H.* Fundamentals of Algebraic Graph Transformation / *H. Ehrig, K. Ehrig, U. Prange, G. Taentzer* // New York: Springer-Verlag, 2006. – 388 p.
27. *Fugetta A.* A classification of CASE technology / *A. Fugetta* // Computer. IEEE Computer Society, 1993. – Vol. 26. – P. 25-38.
28. *Grosu R.* Visual Formalisms Revisited / *R. Grosu, Gh. Stefanescu, M. Broy* // Proceedings 1998 International Conference on Application of Concurrency to System Design. Fukushima, Japan, 1998 – P. 41-51.
29. *Harel D.* On Visual Formalisms / *D. Harel* // Communications of the ACM. – 1988. – Vol. 31. – P. 514-530.
30. *Imran S.* Domain Specific Modeling Language for Early Warning System: Using IDEF0 for Domain Analysis / *S. Imran, F. Foping, J. Feehan, I. Dokas* // International Journal of Computer Science Issues, 2010. – Vol. 7(5). – P. 10-17.
31. *James P.* Designing Domain Specific Languages – A Craftsman’s Approach for the Railway Domain Using CASL / *P. James, A. Knapp, T. Mossakowski, M. Roggenbach* // International Workshop on Algebraic Development Techniques, 2012. – P. 178-194.
32. *Jeong K.* IDEF method-based simulation model design and development / K. Jeong, L. Wu, J. Hong // Journal of Industrial Engineering and Management, 2009. – Vol. 2(2) – P. 337-359.
33. *Kelly S.* MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE Environment/ *S. Kelly, K. Lyytinen, M. Rossi* // In: Proceedings of CAiSE'96, 8th Intl. Conference on Advanced Information Systems Engineering, Lecture Notes in Computer Science. Berlin: Springer-Verlag, 1996 – Vol. 1080. – P. 1-21.
34. *Mikov A.I.* Performance evaluation: textbook / *A.I. Mikov* // Krasnodar: Kuban State University, 2013. – 89 p.
35. *Power J.* Abstraction in Reasoning about Higraph-Based Systems / *J. Power, K. Tourlas* // Foundations of Software Science and Computation Structures. International Book Series «Lecture Notes in Computer Science». Berlin: Springer-Verlag, 2003. – Vol. 2620/2003. – P. 392-408.
36. *Rumpe B.* Executable Modeling with UML. A Vision or a Nightmare? / *B. Rumpe* // Issues & Trends of Information Technology Management in Contemporary Associations, Seattle. London: Idea Group Publishing, 2002. – P. 697-701.
37. *Serifi V.* Functional and Information Modeling of Production Using IDEF Methods / V. Serifi, P. Dasic, R. Jecmenica, D. Labovic // Strojniški vestnik - Journal of Mechanical Engineering, 2009. – Vol. 55. – P. 131-140.
38. *Stolen K.* What is Model Driven Architecture? / *K. Stolen, R.K. Runde* // Research Report 304. Oslo: University of Oslo, 2003. – 17p.
39. *Ward M. P.* Language Oriented Programming / *M. P. Ward* // Software – Concepts & Tools, 1994. – Vol. 15(4). – P. 147-161.
40. *Wise R.* A Systems Engineering Approach to the Development of a Domain-Specific Language for Functional Reference Architectures / R. Wise, E. Brimhall // Proceedings of the 16th Annual Conference on Systems Engineering Research, 2018. – P. 241-254.
41. OpenModelica. Introduction to Object-Oriented Modeling, Simulation, Debugging and Dynamic Optimization with Modelica using OpenModelica [Электронный ресурс]. URL: <https://openmodelica.org/doc/ModelicaShortCourse/ModelicaTutorial-slides-PeterFritzson-160202-BT.pdf> (дата обращения 09.03.2020).
42. Mathworks. Simulink Overview. Simulation and Model‑Based Design [Электронный ресурс]. URL: <https://www.mathworks.com/products/simulink.html> (дата обращения 09.03.2020).
43. COMSOL. COMSOL Multiphysics [Электронный ресурс]. URL: <https://www.comsol.ru/comsol-multiphysics> (дата обращения 09.03.2020).
44. OpenFOAM. The open source CFD Toolbox. User Guide [Электронный ресурс]. URL: <https://www.openfoam.com/documentation/user-guide/> (дата обращения 09.03.2020).
45. IBM. Введение в IBM Rational Software Architect [Электронный ресурс]. URL: <https://www.ibm.com/developerworks/ru/library/kunal/index.html> (дата обращения 09.03.2020).
46. Microsoft. Visual Studio Docs. Overview of Domain-Specific Language Tools [Электронный ресурс]. URL: <https://docs.microsoft.com/en-us/visualstudio/modeling/overview-of-domain-specific-language-tools?view=vs-2019> (дата обращения 10.01.2020).
47. Eclipse. Graphical Modeling Framework. Graphical Modeling Project (GMP) [Электронный ресурс]. URL: <https://www.eclipse.org/modeling/gmp/?project=gmf-tooling#gmf-tooling> (дата обращения 10.01.2020).
48. Eclipse Foundation. Graphical Modeling Framework/Documentation [Электронный ресурс]. URL: <https://wiki.eclipse.org/Graphical_Modeling_Framework/Documentation> (дата обращения 10.01.2020).
49. Baetens, N. Comparing graphical DSL editors: AToM3, GMF, MetaEdit+. Antwerp, Belgium [Электронный ресурс]. URL: <http://atom3.cs.mcgill.ca/people/hv/teaching/MSBDesign/201011/projects/Nick.Baetens/report/report.pdf> (дата обращения 08.02.2020).
50. IDEF. Методология процесса моделирования [Электронный ресурс]. URL: <http://www.idef.ru/idef.php> (дата обращения 02.02.2020).
51. Atomic Object. Lecture Notes on Object-Oriented Programming. UML Notation [Электронный ресурс]. URL: <https://atomicobject.com/resources/oo-programming/uml-notation> (дата обращения 03.02.2020).
52. CASEBERRY. О продукте CASEBERRY [Электронный ресурс]. URL: <https://caseberry.net/index.html> (дата обращения 22.02.2020).
53. Flexberry platform. Архитектура [Электронный ресурс]. URL: <https://flexberry.net/ru/platform-architecture.html?1> (дата обращения 22.02.2020).
54. AGG The Homebase. A brief Description of AGG [Электронный ресурс]. URL: <https://www.user.tu-berlin.de//o.runge/agg/agg-docu.html> (дата обращения 12.03.2020).
55. Eclipse Modeling Project. Eclipse VIATRA [Электронный ресурс]. URL: <https://www.eclipse.org/viatra/> (дата обращения 12.03.2020).
56. *Velter M*. MD\*/DSL Best Practices Update March 2011. Version 2.0 [Электронный ресурс]. URL: http://www.voelter.de/data/pub/DSLBestPractices-2011Update.pdf (дата обращения: 20.03.2020).
57. Описание графических элементов основных существующих нотаций

Разработанный редактор визуальных моделей должен отражать тонкости использующихся подходов к моделированию и учитывать особенности графических нотаций как структурного подхода, так и объектно-ориентированного. Для этого требуется рассмотреть основные использующиеся для разработки систем визуальные модели, выявить их отличительные черты и особенности и определить дополнительные требования к разрабатываемому редактору.

## A.1. Методология функционального моделирования *SADT*

Методология *SADT* представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель *SADT* отображает функциональную структуру объекта, а именно производимые им действия и связи между этими действиями [20].

Основным элементом данной нотации является функция, которая изображается в виде прямоугольника. Слева от данной функции стрелками отображаются входы, а справа – выходы, однако помимо данных параметров у функции также могут быть и исполнители, которые отображаются стрелкой снизу, и управляющие параметры, представляемые стрелкой сверху (рис. А.1). Исполнителями принято считать лица и механизмы, которые выполняют данную функцию, а управляющими параметрами – данные, которые в ходе выполнения функции не изменяются, но влияют на ее результат, например, алгоритм функции.



Рисунок A.1. Графическая нотация диаграмм SADT

С использованием данной методологии сначала строится корневая диаграмма, состоящая из одного блока, а затем на следующих уровнях она декомпозируется путем разбиения функции на подфункции, пока все блоки диаграммы не станут тривиальными. Важной особенностью является то, что все стрелки, инцидентные блоку на предыдущем уровне декомпозиции, должны быть отражены на диаграмме следующего уровня при детализации этого блока и, наоборот, все стрелки, не имеющие конца и начала на одном уровне иерархии, должны быть отражены на родительском уровне. Пример построения иерархии в данной нотации представлен в параграфе 1.2 данной работы, а также в [20]. Также необходимо учитывать, что количество блоков на одной диаграмме, как правило, должно быть не менее 3 и не более 6, чтобы диаграмма была проста для восприятия.

Еще одно необходимое правило – у каждого блока и дуги должно быть свое осмысленное имя, отражающее их суть.

А.2. Моделирование потоков данных в нотации *DFD*

В соответствии с данной методологией модель системы представляется иерархией диаграмм потоков данных (*DFD*), описывающих процесс преобразования информации от её входа в систему до выдачи её пользователю. Декомпозиция, также как и в методологии *SADT*, продолжается до тех пор, пока полученные процессы не станут тривиальными.

На диаграмме потоков данных используются такие элементы, как внешние сущности, потоки данных, подсистемы, процессы и накопители данных.

*Внешняя сущность* – объект, который находится за границами системы и является источником или потребителем информации, что может представляться каким-либо физическим лицом или другой подсистемой. Пример изображения внешней сущности представлен на рис. A.2. Как видно, данный блок представляется «составной» структурой из двух прямоугольников.



Рисунок A.2. Пример внешней сущности

*Подсистема* – группирующий объект модели, позволяющий объединить родственные процессы и подсистемы. Сама подсистема информацию не обрабатывает, ввиду чего она обязана содержать хотя бы один процесс или накопитель данных. Пример изображения подсистемы представлен на рис. A.3. Сверху у подпроцесса указывается номер, а посередине – само название подсистемы.



Рисунок A.3. Пример подсистемы

*Процесс* – преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом. Пример изображения подсистемы представлен на рис A.4. Текст является почти аналогичным тексту подпроцесса, только снизу дополнительно указывается физическая реализация процесса.



Рисунок A.4. Пример процесса

*Накопитель данных* – абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем сами способы помещения и извлечения могут быть любыми. Пример изображения представлен на рис. A.5. Слева указывается буква «*D*» и номер накопителя, а посередине указывается наименование накопителя, отражающее содержание хранимых данных.



Рисунок A.5. Пример накопителя данных

*Поток данных* – сущность, определяющая информацию, передаваемую через некоторое соединение от источника к приемнику. Поток данных изображается в виде линии, оканчивающейся стрелкой, показывающей направление потока (рис. А.6). Каждый поток также имеет имя, которое отражает его содержание. На рис. А.6 показан пример корневой диаграммы потоков данных, отражающей процесс выдачи книги в библиотеке.



Рисунок A.6. Пример корневой диаграммы потоков данных



Рисунок А.7. Пример дочерней диаграммы потоков

Из основных правил при построении диаграмм потоков данных следует выделить необходимость придерживаться того, что потоки данных, у которых нет начала или конца на данной диаграмме, должны отображаться на родительской диаграмме. Более того, количество процессов на диаграмме должно подчиняться правилу «7 ± 2». Отличительной особенностью данной нотации от DFD является то, что объекты могут иметь различное визуальное представление в зависимости от их семантики, а также и то, что объекты могут содержать сразу несколько полей текста.

А.3. Моделирование структур данных в нотации *ERD*

Диаграммы «сущность-связь» (*ERD* – *Entity-Relation Diagrams*) описывают концептуальную схему предметной области, определяя важные для предметной области объекты (в качестве сущностей), их свойства (в качестве атрибутов) и отношения объектов друг с другом (в качестве связей). Данная модель используются при проектировании реляционных баз данных.

*Сущностью* в данной нотации представляется объект, имеющий большое значение для рассматриваемой предметной области, информация о котором подлежит хранению. Графическое изображение сущности представлено на рис. А.8.



Рисунок A.8. Пример сущности

Однако для сущности определяются некоторые правила [20]:

1. Каждая сущность должна иметь уникальное имя.
2. Каждая сущность должна иметь хотя бы один атрибут, однозначно идентифицирующий сущность, и один атрибут, принадлежащий сущности в качестве свойства.
3. Каждая сущность может обладать любым количеством связей с другими объектами.

*Связью* же является поименованная ассоциация между двумя сущностями, которая является значимой для предметной области. Одна из сущностей, как правило, называется родительской, а другая – сущностью-потомком. Всего возможно 3 типа связей:

1. Один к одному («1:1») – экземпляру некоторой сущности может соответствовать только один экземпляр другой сущности.
2. Один ко многим («1:М») – экземпляру некоторой сущности может соответствовать любое количество экземпляров другой сущности.
3. Многие ко многим («M:M») – нескольким экземплярам одной сущности может соответствовать несколько экземпляров другой сущности.

Данный атрибут связи может выражаться либо в начертании самой связи (рис. А.9), либо в соответствующих обозначениях на концах связей. Помимо этого, пунктиром, вместо сплошной линии, может отображаться необязательная связь, описывающая, что сущности может не соответствовать ни один экземпляр другой сущности.



Рисунок А.9. Типы связей в ER-диаграммах

*Атрибутом* является любая характеристика сущности, значимая для рассматриваемой предметной области. Атрибуты изображаются в виде списка имен внутри блока ассоциированной сущности, причем каждый атрибут занимает отдельную строку. Пример ER-диаграммы для АБИС (автоматизированной библиотечной информационной системы) представлен на рис. А.10.

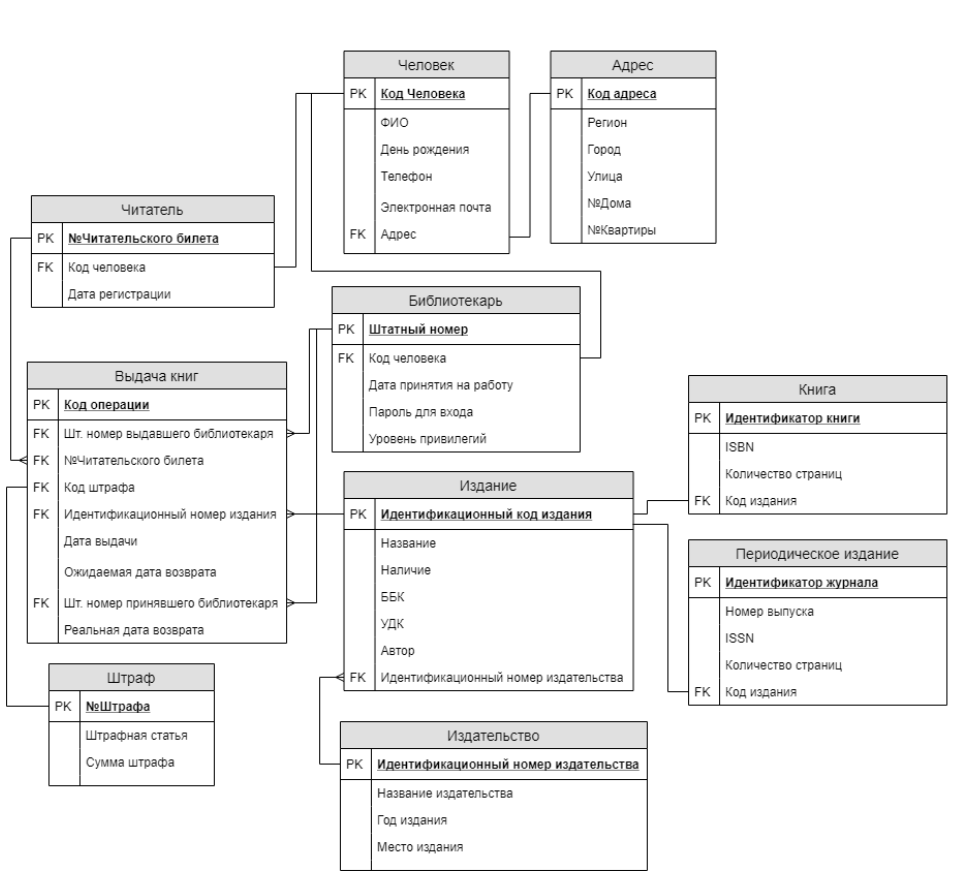


Рисунок А.10. Пример ER-диаграммы

Отличительной особенностью данной нотации от других нотаций структурного подхода является наличие различных представлений для связей, а также наличие *ограничений*, налагаемых на объекты.

А.4. Моделирование функциональных требований в нотации диаграммы прецедентов

Диаграмма прецедентов является единственной диаграммой *UML*, в которой графическая нотация носит иллюстративный характер, а особое внимание уделяется именно текстовым описаниям.

*Прецедентом* является описание множества содержательно-близких сценариев взаимодействия акторов с системой, которое выполняется с целью получения акторами некоторого полезного результата при помощи системы [20]. Прецедент представляется в виде эллипса, в котором хранится наименование прецедента (рис. А.11).



Рисунок А.11. Пример изображения прецедента

Каждый прецедент, как правило, описывается текстом, где представляется взаимодействие акторов и системы.

*Актор* – внешний по отношению к системе объект, который может быть как человек, так и другая система. Актор определяет не конкретного человека, а его роль по отношению к системе. Пример изображения актора представлен на рис. А.12.



Рисунок А.12. Пример изображения актора

Диаграмма прецедентов предполагает 3 вида отношений: ассоциация, зависимость и обобщение.

*Ассоциацией* является устойчивое структурное отношение между сущностями, причем данное отношение может быть только между актором и прецедентом. Ассоциация означает, что определенный актор участвует в определенном прецеденте. Пример использования ассоциации представлен на рис. А.13.



Рисунок А.13. Пример ассоциации

*Обобщением* называется отношение между элементом-родителем и элементом-потомком, при котором потомок является частным случаем родителя, ввиду чего наследует его структуру и поведение. Данное отношение может быть как между двумя прецедентами, так и между двумя акторами. Пример использования обобщения представлен на рис. А.14.



Рисунок А.14. Пример обобщения

*Зависимостью* называется семантическое отношение между двумя сущностями, которое означает, что одна сущность некоторым образом зависит от другой сущности. Зависимости также бывают разных типов – зависимость включения и зависимость расширения, однако изображаются они идентично. Пример использования зависимости представлен на рис. А.15.

Пример диаграммы прецедентов для библиотечной информационной системы представлен на рис. А.16.



Рисунок А.15. Пример зависимости



Рисунок А.16. Пример диаграммы прецедентов

Как видно, возможность написания текста на линии зависит от ее типа – в диаграмме прецедентов текст может быть только на линии, представляющей зависимость. Более того, наименования акторов расположены не внутри их представления на диаграмме, а снизу – что отличается от всех ранее описанных нотаций. На различные отношения тоже накладываются ограничения на то, какие элементы они могут связывать.

А.5. Моделирование бизнес-процессов в нотации диаграммы активностей

В диаграмме активностей появляется новое понятие – *плавательная дорожка*, являющаяся прямоугольной областью диаграммы, в которой располагаются активности данного участника, причем между собой они различаются пунктирной линией.

Главным элементом данной диаграммы является активность – состояние объекта, в котором он выполняет некоторые действия [20]. Активность изображается в виде прямоугольника с закругленными углами (рис. А.17) и располагается на одной из дорожек.



Рисунок А.17. Пример активности

Активности связываются друг с другом с помощью обыкновенных стрелок. Также в данной нотации присутствуют и условные переходы, изображаемые в виде ромба. Сам ромб не содержит текста, однако входящая в него стрелка должна содержать условие, а исходящие стрелки – альтернативы, причем текст на стрелках помещается строго в квадратных скобках. Пример использования условий представлен на рис. А.18.



Рисунок А.18. Пример условного перехода

Очевидно, что из каждой активности может идти только одна стрелка, что можно считать ограничением, однако в данной нотации также присутствуют сложные переходы, которые позволяют представлять распараллеливание процесса (*разветвитель*), а также и слияние нескольких потоков управления в один (*синхронизатор*). Изображение этих переходов на диаграмме представлено на рис. А.19.

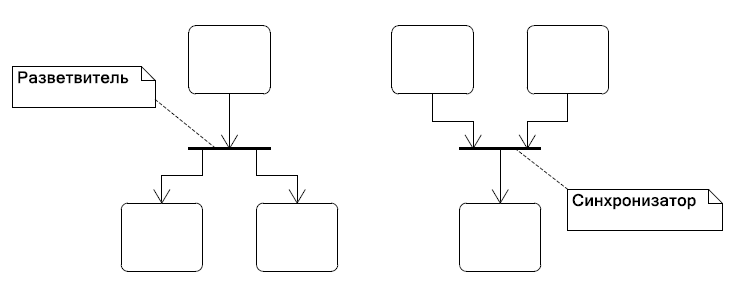


Рисунок А.19. Сложные переходы

Из определения этих переходов следует, что к разветвителю может идти только одна стрелка, а входить как минимум две, а для синхронизатора, наоборот, входить должно как минимум 2 стрелки, а выходить – ровно одна.

Важной особенностью данной диаграммы является то, что у нее есть обозначения начального и конечного узла, причем, начальный узел может быть только один, а конечных может быть несколько, однако также рекомендуется, чтобы конечный узел был только один. Соответственно, важно учитывать максимальное количество определенных элементов на диаграмме.

Пример диаграммы активностей представлен в Прил. B.

А.6. Моделирование поведения системы в нотации диаграммы последовательностей

Модель поведения строится, прежде всего, для определения системных сообщений и операций. Как правило, для этих целей используются диаграммы последовательностей и диаграммы сотрудничества. Далее будет рассмотрена только первая нотация – диаграмма последовательностей, которая при проектировании систем строится для каждого прецедента.

Основными элементами данной диаграммы являются объекты, субъекты и сообщения.

Объект представляется в виде прямоугольника, внутри которого расположено название. Субъект же представляется в виде актора, изображение которого аналогично диаграмме прецедентов. У объектов и субъектов есть линия жизни, которая направлена сверху вниз, причем для объектов она изображается в виде тонкой либо пунктирной линии, а для объектов – в виде жирной линии.

Сообщения делятся на 3 типа – синхронные (после отправки которых отправитель дожидается их обработки), асинхронные (после отправки которых отправитель может совершать другие действия, не дожидаясь обработки) и ответные (которые уведомляют отправителя о завершении обработки или передают результат этой обработки). Представление этих сообщений на диаграмме показано на рис. А.20. Как видно, тип стрелки влияет на ее семантику.

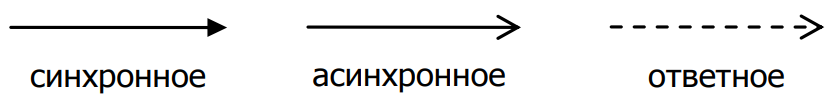


Рисунок А.20. Изображение сообщений

Также в данной диаграмме присутствует необязательный элемент, называемый периодом активации, который иллюстрирует действия объекта по обработке сообщения и представляется в виде прямоугольника, который накладывается поверх линии жизни.

Пример диаграммы последовательностей для прецедента «Изменение данных читателя» представлен на рис. А.21.



Рисунок А.21. Пример диаграммы последовательностей

Соответственно, из важных особенностей данной диаграммы можно выделить то, что различная толщина линии может иметь различную семантику, также, как и тип самой линии или стрелки.

А.7. Моделирование статической структуры в виде диаграммы классов

Диаграммы классов, как правило, строятся параллельно с диаграммами сотрудничества, так как в различных ситуациях акцент делается либо на поведении, либо на структуре системы, ввиду чего обе эти диаграммы должны находится в полном соответствии [20].

Основными объектами данной нотации являются классы и отношения между ними.

Для изображения класса на диаграмме используется прямоугольник, разделенный на 3 части: сверху пишется наименование класса, посередине его атрибуты, а снизу – операции. Раздел атрибутов и раздел операций в некоторых случаях могут оставаться пустыми. Пример класса представлен на рис. А.22.



Рисунок А.22. Пример изображения класса

Отношение между классами может быть различных типов, таких как ассоциация, обобщение (наследование), композитная агрегация и коллективная агрегация, и реализация.

Ассоциация показывает, что объекты одной сущности (класса) связаны с объектами другой сущности таким образом, что можно перемещаться от объектов одного класса к другому.

Наследованием является отношение между классами, когда класс-потомок является частным случаем класса-родителя.

Агрегация – вид ассоциации, моделирующий отношения «часть-целое», причем композитная агрегация – более сильная связь, а коллективная агрегация – более слабая [20].

Также, выделяется отношение «Реализация», которая показывает, что определенный класс реализует заданный интерфейс.

Представления всех отношений в данной нотации изображены на рис. А.23.

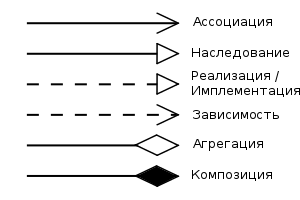


Рисунок А.23. Отношения в диаграмме классов

На концах отношений, как правило, изображается кратность отношения, которое определяет сколько экземпляров класса может быть одновременно связано с другим экземпляром класса. Сами отношения могут также иметь свои наименования.

Стрелки на концах ассоциаций строго определяют её направление. Если стрелка отсутствует, подразумевается, что навигация может проходить в обе стороны.

Частным случаем диаграммы классов является диаграмма понятий, главным отличием которой является отсутствие раздела операций у сущностей.

Пример диаграммы классов для библиотечной информационной системы представлен на рис. А.24.

Таким образом, важной особенностью диаграммы классов также является различие в семантике стрелок, в зависимости от их изображения на диаграмме. Более того, на атрибуты и операции класса накладывается ограничение – они должны начинаться со специального символа, определяющего их видимость.



Рисунок А.24. Пример диаграммы классов

А.8. Построение моделей реализации в нотации диаграммы компонентов

Для построения моделей реализации в UML используется 2 типа диаграмм – диаграммы компонентов и диаграммы развертывания. Далее будет рассмотрен только первый тип, так как диаграмма развертывания имеет только 2 элемента: узел и отношение.

Основным элементом диаграммы компонентов является компонент, для изображения которого используется прямоугольник с двумя накладками слева (рис. А.25).



Рисунок А.25. Пример компонента

Однако компонент может реализовывать определенный интерфейс, который изображается в виде отрезка с окружностью на конце, который располагается перпендикулярно самому компоненту (рис. А.26).



Рисунок А.26. Пример реализации интерфейса

Очевидно, что сами компоненты и интерфейсы должны иметь свое название, однако компоненты также имеют и стереотипы, записываемые в кавычках над именем компонента.

Пунктирной стрелкой показывается зависимость одного компонента от другого, причем расположение стрелки важно, если связывается интерфейс – в таком случае стрелка должна быть инцидента именно окружности на конце отрезка.

В данной нотации могут использоваться и пиктограммы, которые более точно отражают суть компонента, чем его стандартная нотация. Также, могут использоваться и заметки для пояснения сути компонентов.

Пример диаграммы компонентов представлен на рис. А.27. Как видно из рисунка, некоторые компоненты могут объединяться в группы, что делается для повышения наглядности и для отражения их сути.



Рисунок А.27. Пример диаграммы компонентов

Главной особенностью данной нотации является то, что в качестве объектов могут выступать любые изображения, которые пользователь посчитает более наглядными, чем стандартная нотация, ввиду чего при моделировании у пользователя должна быть возможность импорта своих изображений.

А.9. Выводы

Моделирование и проектирование информационных систем происходит с использованием множества различных диаграмм, каждая из которых имеет свои особенности. Разрабатываемый редактор визуальных моделей должен позволять учитывать тонкости нотаций различных подходов к моделированию.

Прежде всего, сами объекты могут иметь различное представление в зависимости от их семантики, причем под объектами понимаются не только блоки, в виде активностей, сущностей, функций и т.д., но и связи между ними. Связи могут иметь как различные «наконечники», так и различный тип линий.

Объекты могут иметь свои наименования, которые могут быть расположены как внутри самого изображения объекта, так и рядом с ним. В связи с этим, на объекты могут накладываться различные ограничения – например, необходимость заполнения текстового поля у объекта или, наоборот, невозможность объекта владеть каким-либо текстовым полем.

Количество некоторых объектов на диаграмме может быть строго ограничено – также, как, например, и количество входных или выходных стрелок из определенного блока. В некоторых случаях также важно учитывать, могут ли связи пересекаться друг с другом.

Место инцидентности блока и связи имеет высокое значение, так как это местоположение также может иметь свою семантику. Более того, в определенных нотациях это место строго определяется и не может быть каким-либо иным.

Некоторые нотации структурного подхода подразумевают возможность декомпозиции, причем линии на родительском уровне, связанные с детализируемым процессом, должны соответствовать линиям на следующем уровне иерархии.

Разные модели описывают процессы и объекты под различным углом, ввиду чего представление одного процесса или объекта может быть выражено сразу несколькими диаграммами различных нотаций, соответственно, и декомпозиция определенного объекта или процесса может быть представлена в виде нескольких диаграмм.

1. Описание бизнес-процесса выдачи книги



Рисунок B.1. Описание бизнес-процесса в виде диаграммы UML

1. Техническое задание

**Редактор визуальных моделей dsm-платформы**

**Техническое задание**

Инв. № подл.

Подпись и дата

Взам. инв. №

Инв. № дубл.

Подпись и дата

|  |  |  |
| --- | --- | --- |
|  |  | Руководитель разработки |
|  |  |  |
|  |  | \_\_\_\_\_\_ Лядова Л.Н. |
|  |  | “24” марта 2020 |
|  |  |  |
|  |  | Ответственный исполнитель |
|  |  |  |
|  |  | \_\_\_\_\_\_ Суворов Н.М. |
|  |  | “24” марта 2020 |

1. **Введение**

Наименование программы – «Редактор визуальных моделей DSM-платформы». Программа является языковым инструментарием, позволяющим производить разработку собственных визуальных нотаций и построение моделей.

1. **Основание для разработки**

Основанием для разработки является учебный план кафедры информационных технологий в бизнесе (ИТБ) по направлению «Программная инженерия» на 3 курс, утвержденный заведующим кафедрой.

1. **Назначение разработки**

Система предназначена для разработки визуальных предметно-ориентированных языков.

Функциональным назначением программы является предоставление пользователю возможности создания и редактирования визуальных языков, а также предоставление возможности использования языков моделирования для описания предметных областей и различных процессов.

Программа предназначена для эксплуатации в процессе моделирования систем. Пользователями программы могут быть как разработчики и IT-специалисты, так и эксперты предметных областей.

1. Требования к программе или программному изделию
   1. **Требования к функциональным характеристикам**

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

1. Взаимодействия с моделями и метамоделями:
   1. Функции создания новой метамодели.
   2. Функции описания элементов метамодели, включая их атрибуты и ограничения.
   3. Функции создания новой модели с использованием созданного языка.
   4. Функции изменения (редактирования) метамодели, путем добавления, удаления или изменения сущностей, отношений, атрибутов и ограничений.
   5. Функции изменения (редактирования) модели, путем добавления, удаления или изменения её элементов.
   6. Функции перехода на следующий или предыдущий уровень иерархии моделей и метамоделей.
   7. Функции трансформации моделей.
   8. Функции отображения свойств элементов модели и метамодели.
   9. Функции представления декомпозиции элементов и связей сразу несколькими моделями.
2. Взаимодействия с визуальным редактором:
   1. Функции добавления, удаления и изменения элементов на полотне.
   2. Функции определения положения точек, через которые связываются элементы.
   3. Функции настройки модели, включая название, признак возможности пересечения связей, признак наличия более 1 связи у сущности, и функции настройки представления модели, включая цвет фона, видимость сетки.
   4. Функции импорта изображений, предназначенных для использования в качестве объектов модели.
   5. Функции сохранения и поиска построенных моделей в репозитории.
   6. Функции представления модели в различных нотациях.
3. Функции импорта и экспорта построенных моделей и метамоделей.

Для программы ввод и вывод данных осуществляется прежде всего с использованием пользовательского интерфейса (Windows Forms/WPF), однако для программы предусмотрена возможность загрузки и сохранения моделей и метамоделей, для чего входные и выходные данные программы должны быть организованы в виде отдельных бинарных файлов или файлов в формате .xml. Для импорта изображений входные данные должны быть организованы в виде файлов графических форматов (.png, .jpg, .jpeg, .bmp). Файлы указанного формата должны размещаться на локальных или съемных носителях, отформатированных согласно требованиям операционной системы.

Выходные данные также должны быть реализованы в виде файлов регистрации в формате .log в качестве средства обеспечения журналирования, которые позволят вести хронологическую запись сведений о происходящих в системе событиях.

Требования к временным характеристикам программы не предъявляются.

* 1. **Требования к надежности**

Надежность ИС в целом зависит от надежности используемых технических средств и операционной системы.

Устойчивое функционирование программы должно быть обеспечено выполнением совокупности организационно-технических мероприятий, перечень которых приведен ниже:

1. организацией бесперебойного питания технических средств;
2. использованием лицензионного программного обеспечения;
3. регулярным выполнением рекомендаций Министерства труда и социального развития РФ, изложенных в Постановлении от 23 июля 1998 г. «Об утверждении межотраслевых типовых норм времени на работы по сервисному обслуживанию ПЭВМ и оргтехники и сопровождению программных средств»;
4. регулярным выполнением требований ГОСТ 51188-98 «Защита информации. Испытания программных средств на наличие компьютерных вирусов»;
5. резервным копированием данных программы, а также возможностью создания контрольных точек.

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать времени, необходимого на перезагрузку операционной системы и запуск программы, при условии соблюдения условий эксплуатации технических и программных средств. При использовании рекомендуемой конфигурации, определенной в требованиях к составу и параметрам технических средств, время восстановления не должно превышать 15 минут.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

Корректность входной и выходной информации отслеживается, а в случае ошибки ИС оповещает пользователя, после чего пользователю повторно предлагается совершить ввод или вывод информации. Ошибки, касающиеся входной и выходной информации, не должны вызывать отказ программы.

Отказы программы возможны вследствие некорректных действий пользователя (оператора) при взаимодействии с операционной системой. Во избежание возникновения отказов программы по указанной выше причине следует обеспечить работу пользователя без предоставления ему административных привилегий. Отказы программы вследствие некорректных действий оператора при взаимодействии с программой через пользовательский интерфейс недопустимы.

Отказы как самой системы, так и ее отдельных функций, могут привести к аварийному завершению работы программы, однако при перезапуске программы ее функциональность не должна пострадать. Среди возможных последствий вышеперечисленных отказов можно выделить нарушение целостности файлов модели и метамодели, и частичную потерю информации, вносившейся непосредственно перед отказом, однако у оператора должна сохраняться возможность вернуться к созданной резервной копии или контрольной точке.

**4.3. Условия эксплуатации**

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

Для работы с данной программой необходимо 2 штатные единицы: системный администратор для поддержания работоспособности технических средств, и сам пользователь программы (оператор).

Системный администратор должен иметь высшее профильное образование, тогда как оператор должен как минимум опыт работы с персональным компьютером и опыт в построении моделей. Пользователь программы (оператор) также должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы.

Программа не требует проведения каких-либо видов обслуживания кроме тех, что описаны в требованиях к надежности.

**4.4. Требования к составу и параметрам технических средств**

Для осуществления пользования данной программой необходимо использование персонального компьютера. Для разработки предполагается, прежде всего, использование Visual Studio 2017, ввиду чего минимальные требования к оборудованию для проведения разработки определяются минимальными требованиями данного продукта:

* процессор с тактовой частотой не ниже 1,8 GHz, при количестве ядер не менее 2;
* ОЗУ не менее 4 ГБ;
* не менее 20 Гб свободного места на жестком диске;
* графическая карта с поддержкой DirectX 9;
* видеоадаптер с минимальным разрешением 720p (1280 на 720 пикселей);
* ОС *MS Windows* 10 версии 1703 и выше;
* клавиатура;
* мышь.

Для использования разработанного программного продукта необходим установленный .*NET Framework* версии не ранее 4.7, однако он поддерживается только ОС *Windows* 10, из-за чего к системе предъявляются совокупные требования данных продуктов:

* процессор с тактовой частотой не ниже 1 GHz, при количестве ядер не менее 2;
* ОЗУ не менее 2 ГБ;
* не менее 25 Гб свободного места на жестком диске;
* графическая карта с поддержкой DirectX 9;
* видеоадаптер с минимальным разрешением 720p (1280 на 720 пикселей);
* ОС MS Windows 10 версии 1703 и выше;
* клавиатура;
* мышь.
  1. **Требования к информационной и программной совместимости**

Информационная структура файла как для входных, так и для выходных данных должна включать в себя текст, содержащий разметку, предусмотренную спецификацией формата .xml. Система может быть расширена установкой конвертеров для разных типов файлов для расширения возможностей импорта и экспорта.

Исходные коды программы должны быть реализованы на языке программирования *C*#. В качестве интегрированной среды разработки программы должна быть использована среда *Microsoft Visual Studio 2017* (локализованная, русская версия).

Данная ИС разрабатывается под операционную систему *MS Windows* 10 (64*x*) с установленным пакетом .*NET Framework* 4.7.

Для защиты информации и программ должна быть реализована система аутентификации пользователя для предотвращения доступа злоумышленника к программе. Защита информации и программ также частично обеспечивается организационно-технических мероприятиями, описанными в требованиях к надежности.

* 1. **Требования к маркировке и упаковке**

Система поставляется в виде открытого кода, доступного для скачивания через сервис GitHub, поэтому требования к маркировке и упаковке отсутствуют.

* 1. **Требования к транспортированию и хранению**

Программа распространяется с использованием веб-сервиса для хостинга *IT*-проектов *GitHub* (https://github.com). Транспортировка и хранение также могут осуществляться на физических носителях: в таком случае, требования к транспортировке и хранению соответствуют аналогичным требованиям для самого физического носителя.

1. **Требования к программной документации**

Требования к программной документации изложены в табл. *C*.1.

***Таблица C.1. Требования к программной документации с содержанием и предназначением***

| **Название документа** | **Краткое содержание** |
| --- | --- |
| Текст программы | Сам код программы с необходимыми комментариями |
| Программа и методика испытаний | Требования, подлежащие проверке при испытании программы, а также порядок и методы их контроля |
| Техническое задание | Назначение и область применения программы, технические, технико-экономические и специальные требования, предъявляемые к программе, необходимые стадии и сроки разработки, виды испытаний |
| Руководство программиста | Объединенное: руководство системного программиста + руководство программиста. Содержит: сведения для проверки, обеспечения функционирования и настройки программы на условия конкретного применения, а также сведения для эксплуатации программы |
| Руководство оператора | Сведения для обеспечения процедуры общения оператора с вычислительной системой в процессе выполнения программы |

1. **Технико-экономические показатели**

Разрабатываемая программа является исследовательским прототипом, разрабатываемым в исследовательских и научных целях в рамках образовательного процесса, ввиду чего расчёт экономической эффективности и годовой потребности не может быть произведен. Тем не менее, разрабатываемая система имеет преимущество перед аналогами, заключающееся в выразительной мощности выбранного математического аппарата, что делает редактор более гибким в работе.

1. **Стадии и этапы разработки**

Стадии и этапы разработки представлены в табл. *C*.2.

***Таблица C.2. Содержание и сроки этапов разработки***

| **№** | **Содержание работы** | **Срок** | **Исполнитель этапа** |
| --- | --- | --- | --- |
| 1 | Постановка задачи | 01.11.2019 | Суворов Н.М. |
| 2 | Определение требований к программе | 15.02.2020 | Суворов Н.М. |
| 3 | Определение и уточнение требований к программным средствам | 07.03.2020 | Суворов Н.М. |
| 4 | Определение стадий, этапов и сроков разработки программы и документации | 17.03.2020 | Суворов Н.М. |
| 5 | Выбор языков программирования | 17.03.2020 | Суворов Н.М. |
| 6 | Описание программы и методики испытаний | 18.03.2020 | Суворов Н.М. |
| 7 | Согласование и утверждение ТЗ | 20.03.2020 | Суворов Н.М. |
| 8 | Проектирование приложения | 01.09.2020 | Суворов Н.М. |
| 9 | Программирование приложения и отладка | 01.02.2021 | Суворов Н.М. |
| 10 | Разработка ПМИ, руководства программиста и оператора | 15.02.2021 | Суворов Н.М. |
| 11 | Проведение приемо-сдаточных испытаний | 01.03.2021 | Суворов Н.М. |
| 12 | Корректировка программы и программной документации по итогам испытаний | 07.03.2021 | Суворов Н.М. |

1. **Порядок контроля и приемки**

Испытание разрабатываемой информационной системы и контроль качества ее работы будет проводиться на базе компьютера, удовлетворяющего рекомендуемым требованиям, изложенным в требованиях к составу и параметрам технических средств. Во время испытаний будут проведены:

Функциональные тестирование.

Тестирование стабильности работы.

Тестирование удобства эксплуатации.

Тестирование защиты и надежности.

Тестирование процедур, выполняемых оператором.

Тестирование производительности системы.