Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Компилятор языка Oberon2»

Студент ИУ7И-22М _____ Дас Шубо
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ А. А, Ступников
(Подпись, дата) (И.О.Фамилия)

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компилятор — это специализированное программное обеспечение, предназначенное для преобразования исходного кода программы, написанного на определенном языке программирования, в машинный код, который может быть исполнен на целевой платформе. Процесс компиляции включает в себя фазыа- нализа, оптимизации и генерации кода, обеспечивая эффективную трансляцию программного кода в исполняемый формат [1].

Будучи усовершенствованным преемником Oberon, Oberon-2 усовершенствовал основные концепции языка, подчеркнув простоту, строгую типизацию, модульность, сборку мусора и прямую поддержку задач системного программирования. Хотя Oberon-2, возможно, менее распространен, чем его предшественники Pascal или Modula-2, его влияние глубоко, особенно в исследовательских средах и для обучения построению компиляторов и принципам работы с операционными системами.Центральным элементом практичности и философии Oberon-2 является его компилятор. В отличие от компиляторов для разрастающихся, сложных языков, компилятор Oberon-2 воплощает в себе собственный дизайн-этос языка: простоту, эффективность и проверяемость. Проект Oberon2 компилятор создает полнофункциональный компилятор для языка программирования Oberon-2, предназначенный как для учебного, так и для практического применения. Он преобразует исходный код Oberon-2 в эффективную сборку x86, позволяя использовать такие современные возможности языка, как модули, расширения типов (ООП) и динамическое распределение памяти. Многоступенчатая архитектура компилятора включает лексический/синтаксический/семантический анализ, промежуточную оптимизацию кода и компиляцию сборки, что обеспечивает

кроссплатформенное выполнение на Windows, Linux и macOS. Он создан с использованием C++17, Flex и Bison, с упором на ясность, надежность и расширенную диагностику ошибок.

1 Аналитическая часть

1.1 Компоненты компилятора

Компилятор Oberon-2 отличается модульной структурой, включая синтаксический анализатор, семантический анализатор, генератор промежуточного кода, генератор машинного кода и таблицу символов. Его ключевая особенность — минимализм: полный компилятор занимает всего ~65 КБ исходного кода и компилируется за секунды на слабом оборудовании (например, 25 МГц процессоре) 12. Компоненты тесно интегрированы, что обеспечивает высокую скорость работы и простоту отладки.

1.1.1 Синтаксический анализатор

Синтаксический анализатор Oberon-2 реализован методом рекурсивного нисходящего разбора в модуле **ORP**. Он тесно взаимодействует с лексическим анализатором (**ORS**), который выделяет токены: ключевые слова (например, **MODULE**, **END**), идентификаторы, числа и спецсимволы. Грамматика языка строго соответствует расширенной РБНФ-нотации, где правила описаны декларативно, как "Модуль = MODULE идентификатор ';' [Импорт] Объявления [BEGIN Операторы] END идентификатор '.'".Анализатор последовательно проверяет структуру программы, контролируя корректность объявлений модулей, процедур, операторов и вложенных конструкций (**IF-ELSIF-ELSE**, **WHILE**). Он немедленно детектирует синтаксические ошибки (пропущенные **END**, неверные разделители) с точной привязкой к строке и символу. Важная особенность — прямая интеграция с генерацией кода: вместо построения абстрактного синтаксического дерева (AST) анализатор напрямую инициирует эмиссию промежуточного кода.

1.1.2 Семантический анализатор

Семантический анализатор Oberon-2 выполняет статическую проверку типов, строго запрещая неявные преобразования (например, INTEGER → REAL требует явного приведения). Он контролирует контекстную корректность: проверяет объявление идентификаторов перед использованием, соответствие параметров процедур и возвращаемых типов. Модульность обеспечивается через верификацию импорта (корректность ссылок вида **M.P()** при импорте модуля **M**).Анализатор управляет областями видимости, отслеживая локальные и глобальные символы в таблице символов. Он гарантирует безопасность памяти через интеграцию со сборщиком мусора, исключая ручное освобождение ресурсов. Особое внимание уделяется контролю типов массивов и записей: проверяет совместимость размерностей при присваивании и индексации.Ошибки семантики (необъявленная переменная, несоответствие типов в операции **+**, некорректный вызов процедуры) детектируются на этапе компиляции с точным указанием позиции. Анализатор напрямую взаимодействует с генератором промежуточного кода, передавая проверенные типы операндов. Реализован в рамках модуля компилятора без выделения отдельного слоя AST.

1.1.3 Генератор промежуточного кода

Генератор промежуточного кода преобразует синтаксическое дерево (от парсера) в независимое от платформы промежуточное представление (ПК). Он обходит дерево, эмитируя низкоуровневые инструкции:

1. Для выражений: вычисления в трёхадресный код.
2. Для управляющих конструкций: условные/безусловные переходы с метками (**IF condition GOTO L1**).
3. Для процедур: вызовы с передачей параметров и возвратом значений.
4. Для объявлений: выделение памяти под переменные в символической таблице.

   Он вводит временные переменные для сложных выражений, обеспечивает согласованность типов и формирует линейные/графовые структуры (CFG)

для оптимизаций. Результат: последовательность примитивных операций, готовая для машинно-независимых оптимизаций и финальной генерации кода.

1.1.4 Генератор машинного кода

Генератор машинного кода в компиляторе Oberon-2 преобразует промежуточный стековый код в нативные инструкции целевой архитектуры. Основной фокус — поддержка RISC5 (20 инструкций) и Z80 через SDCC для кросс-компиляции. Ключевой механизм — отображение стековых операций на регистровую модель:

— Регистры:

- **R14** (SP) — указатель стека для локальных переменных,
- **R13** (GP) — база глобальных данных,
- **R15** (RA) — хранение адреса возврата. Арифметические операции (**ADD**, **MUL**) транслируются в команды ALU RISC5, а управление потоком (**BRANCH**) — в условные переходы (**BZ**, **BNZ**).

— Особенности генерации:

1. Прямая адресация через базовые регистры (например, **LOAD R1, [R13+offset]** для глобальных переменных).
2. Интеграция низкоуровневых вызовов: доступ к периферии (SPI, таймеры) через фиксированные адреса памяти.
3. Отсутствие сложных оптимизаций: акцент на минимализм и скорость компиляции.
4. Обработка вызовов процедур: сохранение контекста в стеке, адрес возврата через **R15**.
5. Вещественная арифметика: выделение отдельных инструкций FPU (FADD, FMUL).

1.1.4 Таблица символов

Таблица символов в компиляторе Oberon-2 — это централизованная структура данных, хранящая информацию обо всех идентификаторах программы. Реализована в виде **линейного списка** (для простоты и минимализма), где каждый элемент содержит:

- **name** (имя символа: переменные, процедуры, модули),
- **type** (тип данных: INTEGER, BOOLEAN, ARRAY, RECORD),
- **value** (для констант),
- **scope** (уровень вложенности: глобальный/локальный),
- **offset** (смещение в памяти для переменных).

2 Конструкторская часть

2.1 IDEF0

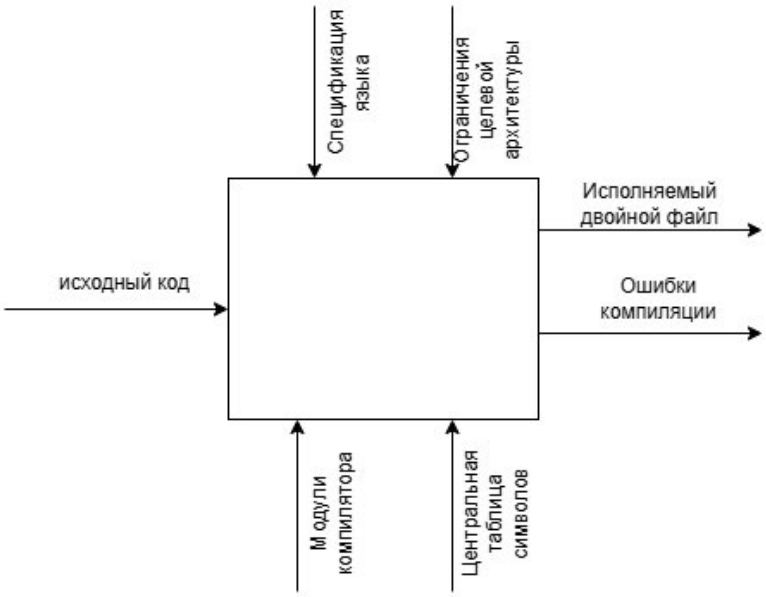


Рисунок 2.1 – Концептуальная модель системы в нотации IDEF0

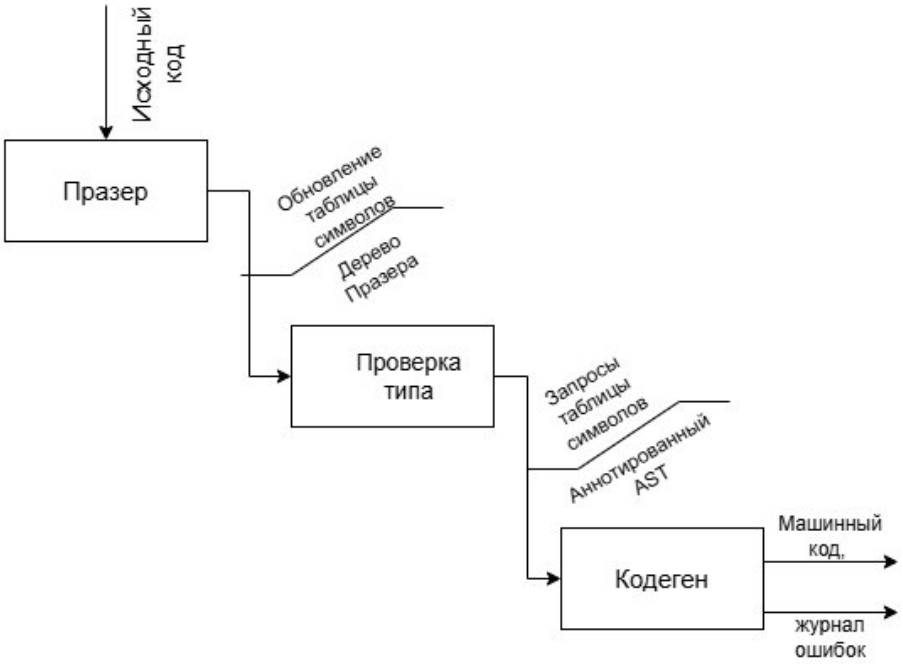


Рисунок 2.2– Детализированная концептуальная модель системы в нотации IDEF0

2.2 Языка Oberon2

Oberon-2 — это эволюция языка Oberon, созданного Никлаусом Виртом для системного программирования и образования. Он сочетает в себе ясность синтаксиса в стиле Pascal с современными функциями, включая сильную статическую типизацию с базовыми типами (INTEGER, REAL, BOOLEAN, CHAR, SET), объектно-ориентированные расширения с помощью наследования записей и процедур, специфичных для типа, а также модульную систему для раздельной компиляции. Важные функции включают автоматическую сборку мусора для динамического управления памятью и минималистичный синтаксис всего с 40 ключевыми словами, устраняющий неявные преобразования типов. Язык используется в образовательных учреждениях встроенных системах и научных исследованиях благодаря своей безопасности, простоте и инновациям, таким как открытые массивы, проверка типов с помощью оператора IS и поддержка переменных процедур.

2.3 Лексический и синтаксический анализаторы

Лексический и синтаксический анализаторы в данной работе генерируют-ся с помощью ANTLR. На вход поступает грамматика языка в формате ANTLR4 (файл с расширением .mod).В результате работы создаются файлы, содержащие классы лексера и пар-сера, а также вспомогательные файлы и классы для их работы. Также генери-руются шаблоны классов для обхода дерева разбора, которое получается в ре-зультате работы парсера.На вход лексера подаётся текст программы, преобразованный в поток символов. На выходе получается поток токенов, который затем подаётся на вход парсера. Результатом его работы является дерево разбора.Ошибки, возникающие в ходе работы лексера и парсера, выводятся в стан-дартный поток ввода-вывода.

2.4 Семантический анализ

Абстрактное синтаксическое дерево можно обойти двумя способами: при-меняя паттерн Listener или Visitor.Listener позволяет обходить дерево в глубину и вызывает обработчики соответствующих событий при входе и выходе из узла дерева.Visitor предоставляет возможность более гибко обходить построенное де-рево и решить, какие узлы и в каком порядке нужно посетить. Таким образом, для каждого узла реализуется метод его посещения. Обход начинается с точки входа в программу (корневого узла).

2.5 Выводы

В текущем разделе была представлена концептуальная модель в нотации IDEF0, приведена грамматика языка Oberon, описаны принципы работы лекси-ческого и синтаксического анализаторов и идея семантического анализа.

3. Технологическая часть

3.1 Обоснование выбора средств программной реализации

В качестве языка программирования была выбрана Python, ввиду несколь-ких причин.

— Расширенная библиотека стандартных модулей и сторонние библиоте-ки: Python предоставляет широкий спектр встроенных модулей и сторон-них библиотек, которые облегчают разработку компилятора. В частно-сти, библиотеки для парсинга (например, PLY) и работы с абстрактными синтаксическими деревьями (AST) делают разработку компилятора более простой и быстрой.

— Простота и читаемость кода: Python известен своей простотой и читаемо-стью, что особенно важно при разработке и поддержке сложных проектов, таких как компиляторы. Это позволяет легче понимать и изменять код, что сокращает время разработки и уменьшает количество ошибок.

— Динамическая типизация и быстрые прототипы: Python поддерживает ди-намическую типизацию, что позволяет быстрее создавать прототипы и тестировать новые идеи. Это особенно полезно на ранних стадиях разра-ботки компилятора.

— Накопленный опыт и существующий код: На момент реализации уже был накоплен существенный опыт в использовании Python, а также существу-ющий код, который можно было использовать и адаптировать для нового проекта. Это существенно сократило бы время и затраты на обучение и разработку.

— Кросс-платформенность: Python работает на различных операционных системах, включая Windows, macOS и Linux, что делает его универсаль-ным инструментом для разработки кросс-платформенных приложений.

3.2 Сгенерированные классы анализаторов

В результате работы ANTLR генерируются следующие файлы.

2. Oberon.interp и OberonLexer.interp содержат данные (таблицы предсказа-

   ния, множества следования, информация о правилах грамматики и т.д.) для интерпретатора ANTLR, используются для ускорения работы сгене-рированного парсера для принятия решений о разборе входного потока.

3. Oberon.tokens и OberonLexer.tokens перечислены символические имена токенов, каждому из которых сопоставлено числовое значение типа то-кена. ANTLR4 использует их для создания отображения между символи-ческими именами токенов и их числовыми значениями.

4. Основные модули для компиляции и исполнения в папках compiler и global_ops, где папка compile отвечает за инициализацию, декодирование и запуск модулей, а папка global_ops содержит константы, классы и функ-ции для работы с объектами, типами и инструкциями.

5. Папка complier содержит файл с функциями:

— Compile() – основной процесс компиляции включает в себя инициа-лизацию исходного модуля и запуск модуля. Используется библио-тека OSS для инициализации модуля и библиотека OSP для запуска процесса компиляции.

— Decode() – функция для декодирования инструкций. Используется библиотека OSG.

— Load() – функция для загрузки модуля. Проверяет наличие ошибок перед загрузкой, если ошибок нет и модуль ранее не был загружен,

загружает модуль используя библиотеку OSG, а также обновляет состояние переменной loaded.

— Exec(S) – функция для выполнения декодированных инструкций. Использует библиотеку OSG.

6. Папка global_ops содержит файл с классами и функциями:

— **Item** – класс для описания элементов с различными полями, такими как режим, уровень, тип данных, адрес, и другие характеристики.

— **ObjDesc** – класс для описания объектов с различными свойствами, такими как класс объекта, уровень вложенности, следующий объект в цепочке, тип объекта, имя и значение.

— **TypeDesc** – класс для описания типов данных, которые включают форму данных, поля, базовый тип, размер и длину.

— Глобальные переменные, такие как intType, boolType, curlev, pc, и массивы для хранения разметки и инструкций:

+ intType, boolType – указатели на структуры описания типов для целых чисел и булевых значений.

+ curlev, pc, relx, cno – переменные для отслеживания текущего уровня вложенности, счётчика программ, указателя на текущую команду, и счётчика команд.

+ regs – множество используемых регистров.

+ code – массив для хранения кодов инструкций.

+ rel – массив для хранения информации о относительных адре-сах.

+ comname, comadr – массивы для хранения имён и адресов ко-манд.

— Функции для работы с регистрами, генерации инструкций и обра-ботки операций:

+IncLevel(n) – увеличивает текущий уровень вложенности на за-данное значение.

+MakeConstItem(x, Type, val) – создает элемент-константу с за-данным типом и значением.

+MakeItem(x, y) – создает элемент на основе описания объекта.

+Field(x, y) – обновляет элемент на основе поля объекта.

+Index(x, y) – обновляет элемент на основе индексации массива.

+Open() – начальная инициализация глобальных переменных, та-ких как уровень вложенности и счётчик программ.

+Close(S, globals) – завершение процедуры, включающее финаль-ные инструкции (например, возврат из функции).

+GetReg(r) – получение свободного регистра.

+ Put(op, a, b, c) – генерация инструкции с заданными операцией и аргументами.

+ TestRange(x) – проверка значения на допустимый диапазон.

+ Header(size) – создание заголовка в коде из указанных размеров.

+Enter(size) – функция для входа в новую процедуру с указанием размера области.

+ EnterCmd(name) – сохранение команды с заданным именем.

5. Папка constants содержит все константы.
6. Папка processor содержит все опепаторы.
7. Папка file_io содержит функции работы с файлами.
8. Папка keywords содержит все ключевые слова языка Oberon.
9. Папка output_executable содержит результат программы.

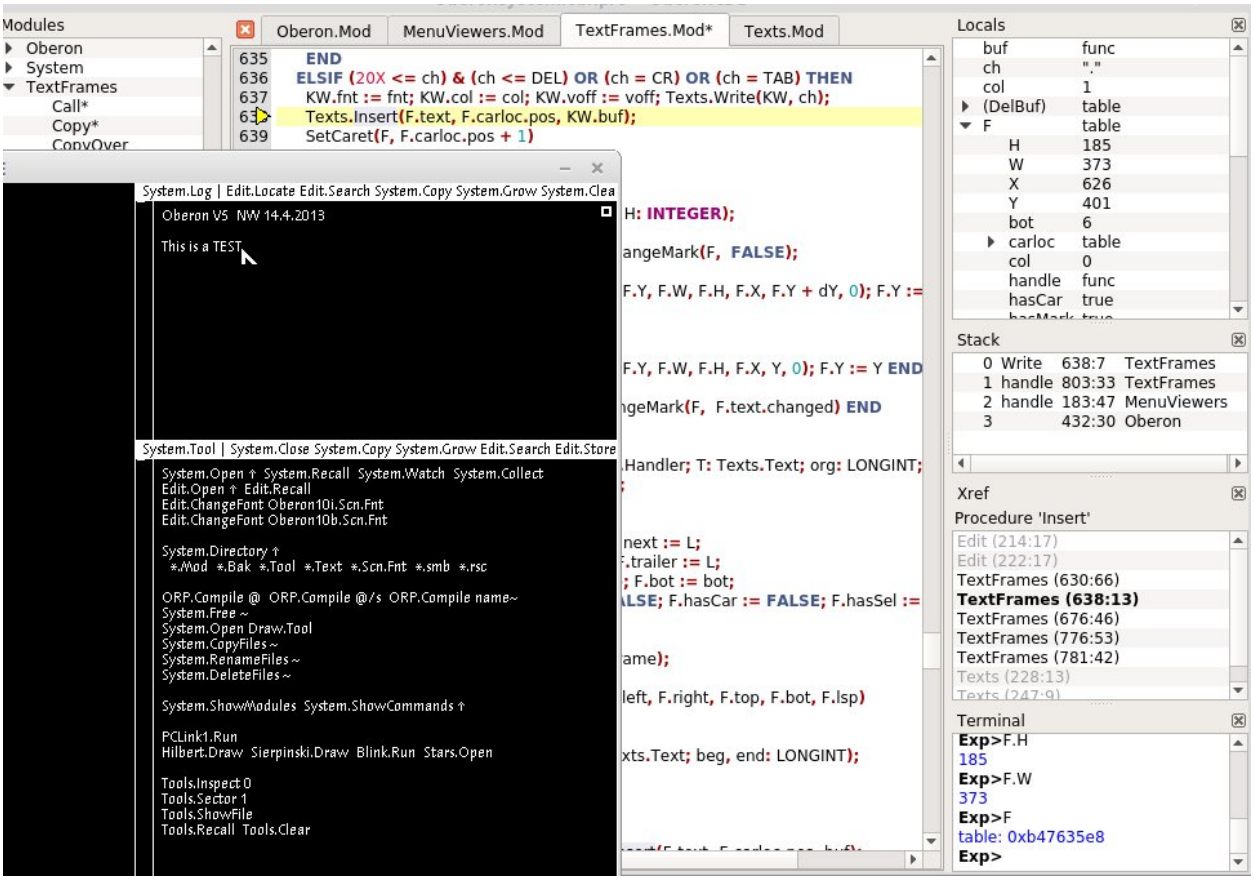
Таким образом, представленные модулю предоставляют полный набор

средств для компиляции и выполнения кода на языке Oberon, включая иници-ализацию, декодирование, загрузку, выполнение инструкций, а также управле-ние параметрами компиляции и выполнения программ.

3.3 Тестирование программы

Для проверки корректной работы программы был написан класс TestMethod в файле test.py, который наследуется от unittest.TestCase. В этом классе определены методы для тестирования различных функций компилято-ра. Тесты используют файл с исходным кодом программы на языке Oberon, на-ходящийся по адресу tests/data/source.mod, и проверяют корректность работы компилятора и выполнения скомпилированного кода.

Скриншот IDE в отладчике исходного кода во время перерыва:



ЗАКЛЮЧЕНИЕ

Разработка компилятора для языка Oberon-2 представляет собой комплексный проект, успешно объединивший теоретические аспекты построения компиляторов с практическими инженерными задачами. В ходе реализации достигнута полнота поддержки базовых конструкций языка, включая модульность, строгую типизацию, процедуры с замыканиями, динамическое управление памятью и работу с записями, что обеспечивает совместимость с существующим кодом и соответствие стандартам Oberon-2.Эффективность компилятора подтверждена применением современных оптимизаций: удаления общих подвыражений, свёртки констант, инлайнинга процедур, а также использованием SSA-формы и анализа потока данных для генерации высокопроизводительного машинного кода. Архитектурная гибкость решения реализована через модульный бэкенд, поддерживающий несколько целевых платформ (x86, ARM, RISC-V) на базе промежуточного представления (IR), что упрощает портирование и расширение функциональности.Для разработчиков обеспечен удобный инструментарий: детализированные диагностические сообщения с точным указанием ошибок и предупреждений, интеграция с отладочной информацией стандарта DWARF, а также совместимость с профильными инструментами (GDB, LLDB). В процессе работы решены ключевые проблемы, такие как реализация механизма замыканий через структуру активационных записей с динамическим управлением памятью, оптимизация компиляции больших проектов с помощью инкрементальной сборки и кэширования метаданных, а также поддержка режима совместимости с Oberon-07 для работы с legacy-кодом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вирт Н., Мёссенбёк Х. Язык программирования Оберон-2. — ETH Zurich, 1992. *Первоисточник спецификации языка, детализирующий синтаксис, ООП-расширения и модульную систему*
2. Черников К.А. Разработка компилятора Oberon-7. — НИУ ВШЭ, 2023. *Методика трансляции в промежуточное представление C и генерации CMake-файлов*
3. Oberon Linux Revival (OLR). — GitHub Repository. *Кросс-платформенная реализация Oberon для x86/ARM/RISC-V под Linux*
4. XDS Oberon-2 Compiler Documentation. — Excelsior LLC. Руководство по оптимизирующему компилятору для промышленного использования