|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «ИНФОРМАТИКА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОЕКТИРОВАНИЕ И ТЕХНОЛОГИЯ ПРОИЗВОДСТВА ЭЛЕКТРОННОЙ АППАРАТУРЫ»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

*К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ*

*НА ТЕМУ:*

«Разработка эмулятора ядра MIPS с использованием современных средств и методов проектирования ПО»

Студент группы ИУ4-41М **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кутаев К.С.**

(Подпись, дата)

Руководитель ВКР **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гладких А.А.**

(Подпись, дата)

Нормоконтролер **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сергеева М.Д.**

(Подпись, дата)

2024 г.

АННОТАЦИЯ

Объектом исследования выступают современные средства и методы разработки ПО, а также эмуляторы встраиваемых систем.

Цель работы – исследование современных средств и методов проектирования ПО, а также их применение для реализации эмулятора встраиваемых систем, позволяющего повысить быстродействие эмуляции ПО и уменьшить трудозатраты на реализацию составляющих эмулируемой системы.

В процессе работы произведен обзор существующих эмуляторов ядра MIPS, а также современных средств разработки ПО. Разработана методика исследования быстродействия современных средств проектирования ПО для разработки эмулятора встраиваемых систем. Реализована разработанная методика исследования быстродействия современных средств проектирования ПО в виде программного комплекса. Разработана архитектура эмулятора ядра MIPS.

В результате работы реализован эмулятор ядра MIPS с использованием современного средства проектирования ПО Rust. Проведено испытание быстродействия эмуляторов ядра MIPS. Проведен расчет и сравнение средних значений количества строк исходного кода одной функциональной точки ПО эмуляторов ядра MIPS.

**ABSTRACT**

The object of the research is modern tools and methods of software development, as well as emulators of embedded systems.

The purpose of the work is to study modern tools and methods of software design, as well as their use for the implementation of an embedded systems emulator, which makes it possible to increase the performance of software emulation and reduce labor costs for the implementation of the components of the emulated system.

In the process of work, a review of existing MIPS kernel emulators, as well as modern software development tools, was carried out. A methodology has been developed for studying the performance of modern software design tools for developing an emulator for embedded systems. The developed methodology for studying the performance of modern software design tools in the form of a software package has been implemented. The MIPS kernel emulator architecture has been developed.

As a result of the work, a MIPS kernel emulator was implemented using the modern software design tool Rust. The performance of MIPS kernel emulators has been tested. The calculation and comparison of the average values of the number of lines of source code for one functional point of the MIPS kernel emulator software was carried out.

СОДЕРЖАНИЕ

[АННОТАЦИЯ 2](#_Toc161813047)

[СОДЕРЖАНИЕ 4](#_Toc161813048)

[УСЛОВНЫЕ ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ И ТЕРМИНЫ 5](#_Toc161813049)

[ВВЕДЕНИЕ 6](#_Toc161813050)

[1 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ 10](#_Toc161813051)

[1.1 Описание предметной области 10](#_Toc161813052)

[1.2 Обзор существующих решений 13](#_Toc161813053)

[1.3 Формулирование цели и постановка задач 17](#_Toc161813054)

[1.4 Анализ исполнения программного обеспечения 19](#_Toc161813055)

[1.5 Анализ средств проектирования ПО 23](#_Toc161813056)

[1.6 Метод оценки трудоемкости разработки ПО 28](#_Toc161813057)

[1.7 Разработка методики исследования быстродействия современных средств проектирования ПО 31](#_Toc161813058)

[2 РАЗРАБОТКА ПО ИССЛЕДОВАНИЯ БЫСТРОДЕЙСТВИЯ СРЕДСТВ ПРОЕКТИРОВАНИЯ ПО 34](#_Toc161813059)

[2.1 Реализация разработанной методики исследования быстродействия современных средств проектирования ПО 34](#_Toc161813060)

[2.2 Выбор алгоритма для проведения исследования 37](#_Toc161813061)

[2.3 Анализ результатов исследования 42](#_Toc161813062)

[Выводы 47](#_Toc161813063)

[3 РАЗРАБОТКА ЭМУЛЯТОРА ЯДРА MIPS 48](#_Toc161813064)

[3.1 Обзор выбранных средств реализации 48](#_Toc161813065)

[3.2 Обзор процессорной архитектуры MIPS 50](#_Toc161813066)

[3.3 Разработка архитектуры эмулятора ядра MIPS 57](#_Toc161813067)

[3.4 Разработка эмулятора ядра MIPS 60](#_Toc161813068)

[ЗАКЛЮЧЕНИЕ 61](#_Toc161813069)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 62](#_Toc161813070)

УСЛОВНЫЕ ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ И ТЕРМИНЫ

|  |  |  |
| --- | --- | --- |
| ГОСТ | - | Государственный стандарт, |
| ЕСКД | - | Единая система конструкторской документации, |
| КД | - | Конструкторская документация, |
| МК | - | Микроконтроллер, |
| МС | - | Микросхема, |
| ПК | - | Персональный компьютер, |
| ПО | - | Программное обеспечение, |
| РПЗ | - | Расчётно-пояснительная записка, |
| РТЗ | - | Расширенное техническое задание, |
| САПР | - | Система автоматизированного проектирования, |
| ТЗ | - | Техническое задание, |
| ЭА | - | Электронная аппаратура, |
| ЭС | - | Электронное средство, |
| GDB | - | GNU Debugger (GNU Отладчик) |
| GPIO | - | General Purpose Input-Output (Интерфейс Ввода-Вывода Общего назначения), |
| IoT | - | Internet of Thigs (Интернет Вещей), |
| MIPS | - | Microprocessor without Interlocked Pipeline Stages (Микропроцессор без взаимосвязанных ступеней конвейера) |
| SPI | - | Serial Peripheral Interface (Последовательный Периферийный Интерфейс), |
| SWD | - | Serial Wire Debug (Последовательный Проводной Отладочный интерфейс), |
| USB | - | Universal Serial Bus (Универсальная Последовательная Шина) |

# ВВЕДЕНИЕ

В процессе проектирования электронных устройств часто возникает потребность в тестировании их работоспособности. Отладка разрабатываемого программного обеспечения на самом устройстве в большинстве случаев невозможна в связи с недоступностью устройства или отсутствия на нем необходимых аппаратных частей, которые ещё находятся в процессе проектирования. Тестирование работы программного обеспечения для разрабатываемого устройства на персональном компьютере разработчика в части случаев невозможна из-за различий в аппаратной архитектуре процессора, периферии и окружении. Исходя из этого можно сделать вывод о востребованности решения, которое способно проводить отладку программного обеспечения для устройства без доступа к самому устройству.

Для решения данной проблемы используются эмуляторы - программные средства, которые имитируют функции устройства и его окружения в другой вычислительной системе, чтобы эмулированное поведение максимально близко соответствовало поведению настоящего устройства. Данные средства позволяют на одной компьютерной системе, называемой хостом, запускать программы, написанные для другой системы, называемой гостем.

**Работа посвящена** разработке эмулятора ядра аппаратной архитектуры MIPS. Рассмотрены особенности эмуляции аппаратных платформ одной вычислительной системы в другой. Предложено использование современных средств и методов разработки ПО для реализации эмулятора. Описанный способ имеет преимущество по скорости выполнения ВПО, которое эмулируется, а также по трудозатратам для реализации составляющих эмулируемой системы. Основное внимание уделено сравнению быстродействия современных средств и методов разработки ПО. Показаны возможности аналогов эмулятора аппаратных платформ по скорости выполнения инструкций и трудозатратам для реализации составляющих эмулируемой системы. Основная проблема аналогов заключается либо в использовании устаревших методов проектирования ПО, которые увеличивают трудозатраты на реализацию новых составляющих эмулируемой систем, либо использование средства проектирования ПО с малыми характеристиками быстродействия. Это следствие использования определенных средств и методов разработки ПО. Предложена методика исследования средств и методов проектирования ПО по критериям быстродействия, отличающейся от существующих стабильным окружением, в котором проводится измерение быстродействия, а также иерархическим дескриптором описания конфигурации проводимого исследования. Предложенная методика реализован в виде ПО для тестирования скорости выполнения определенных алгоритмов на различных языках программирования. По результатам тестирования были выбраны методы и средства для разработки эмулятора аппаратных платформ, отвечающее требуемым критериям. Они позволяют эффективно использовать эмуляцию в процессе разработки различных устройств и встраиваемых систем. Эмулятор может быть использован для решения различных задач при отладке в динамическом режиме программных алгоритмов встраиваемого электронного устройства.

**Объектом исследования** выступают современные средства и методы разработки ПО, а также эмуляторы ядра MIPS.

**Предметом исследования** является возможность реализации эмулятора, способного соответствовать повысить быстродействие эмуляции ПО и уменьшить трудозатраты на реализацию составляющих эмулируемой системы.

**Актуальность работы:** сфера применения эмуляторов достаточно обширна. Они используются в процессе разработки как небольших аппаратно-цифровых устройств, так и массивных программно-аппаратных комплексов. Это связано с типичными проблемами аппаратной разработки, такими как отсутствие полностью рабочей аппаратной части или опасность нанесения ущерба в случае неправильного функционирования устройства. Эмулятор позволяет решить данные проблемы за счет полной программной имитации поведения разрабатываемого устройства, позволяя разработчику отлаживаться без наличия аппаратной части в принципе.

​Помимо разработки эмуляторы применяются при проведении исследований и обратной разработки различных встраиваемых устройств. При отсутствии исходного кода возникают трудности при проведении анализа работы внутреннего программного обеспечения устройства. Анализ бинарного кода, собранного под системы персональных компьютеров как правило, не вызывает никаких затруднений на самом раннем его этапе, поскольку объект анализа полностью доступен. В процессе работы устройства часть данных, которые обрабатываются во внутреннем программном обеспечении, генерируются только в процессе его работы. При этом на устройстве не предусмотрено возможности для получения их пользователем. В связи с этим возникает проблема получения некоторой части данных, которые нельзя найти обычным статическим анализом кода. С помощью эмулятора можно запустить работу устройства на другой системе и получить доступ ко всем имеющимся внутри данным, в том числе к тем, которые генерируются в процессе работы, а также к данным регистров и памяти. Стоит отметить, что таким же образом эмулятор применяется для поиска уязвимостей во внутреннем программном обеспечении исследуемого устройства. Для этого к нему добавляется программное средство для проведения фаззинг-тестирования, которое передает на вход устройству неправильные, неожиданные или полностью случайные данные. За счет полного доступ ко всем частям работающего устройства внутри эмулятора можно отслеживать состояния работы его внутреннего программного обеспечения. Все это позволяет фиксировать отказ в обслуживании устройства и находить ранее неизвестные уязвимости.

Автоматизация подобных операций способна существенно сократить необходимые для разработки методов отладки ВПО встраиваемых систем критической инфраструктуры усилия, что критически снижает временную стоимость исследований на безопасность встраиваемых устройств автоматизации критической инфраструктуры.

**Целью работы** является исследование современных средств и методов проектирования ПО, а также их применение для реализации эмулятора встраиваемых систем, позволяющего повысить быстродействие эмуляции ПО и уменьшить трудозатраты на реализацию составляющих эмулируемой системы.

**Структура работы**: Работа состоит из введения и четырех глав. Первая глава представляет собой исследовательскую часть, включающую в себя обзор и анализ существующих решений и аналогов эмулятора аппаратных платформ, обзор и анализ современных средств проектирования ПО, а также разработку методики исследования быстродействия современных средств проектирования ПО для разработки эмулятора встраиваемых систем.

Во второй главе представлена реализация разработанной методики исследования быстродействия современных средств проектирования ПО в виде программного комплекса, а также проведение исследований быстродействия современных средств проектирования ПО с помощью разработанного программного комплекса.

В третьей главе представлена конструкторская часть, которая включает в себя анализ и разработку архитектуры эмулятора ядра MIPS, реализацию эмулятора ядра MIPS с использованием выбранных современных средств проектирования ПО, который позволит повысить быстродействие выполнения эмулируемого ПО и снизить трудозатраты на реализацию составляющих эмулируемой системы.

В четвертой главе представлено проведение испытаний быстродействия эмуляторов ядра MIPS для подтверждения эффективности разработанного решения путем оценки количества выполняемых инструкций в секунду в процессе эмуляции заданного алгоритма, а также проведение расчета и сравнение средних значений количества строк исходного кода одной функциональной точки ПО эмуляторов ядра MIPS..

1 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

## Описание предметной области

Эмулирующая среда (эмулятор) -- программное средство, которое имитирует (эмулирует) работу вычислительной системы и ее окружения в другой вычислительной системе, чтобы имитированное поведение максимально соответствовало поведению настоящей вычислительной системы. Эмуляторы обладают широким спектром применения. Они используются, в частности, для проведения динамического анализа выполнения ПО в рамках проведения сертификационных испытания, для тестирования производительности или для подтверждения корректности поведения устройств или программ. Они могут быть использованы, например, для отладки разрабатываемой программной части программно-аппаратного продукта в случае, если отладка на физическом устройстве затруднительна или невозможна.

История появления эмуляторов связна с развитием компьютеров и появлением новых архитектур. С появлением новой более совершенной архитектуры возникла проблема обратной совместимости программного обеспечения. Разработчикам операционных систем было невыгодно переписывать программные средства под новую архитектуру персонального компьютера. Для решения данной проблемы был разработан эмулятор, который позволял запускать старые программные средства на персональном компьютере с новой архитектурой.

Использование эмуляции как метода тестирования и отладки ПО существует уже довольно давно, начиная с появления первых компьютеров. Оно имеет ряд преимуществ по сравнению с использованием реального оборудования. При запуске эмуляции ПО для тестирования загружается путем записи его непосредственно в эмулируемую память. Эмуляцией ПО называют запуск исполняемого файла в эмулирующей среде. Таким образом, нет необходимости загружать ПО по последовательной линии или сети и запускать мониторинг состояния целевой встраиваемой системы.

Эмулируемая встраиваемая система также просто управляется с помощью передачи команд эмулятору, таким образом нет необходимости физического доступа к целевой встраиваемой системе. За счет того, что эмулятор запускается на автоматизированном рабочем месте (АРМ) разработчика можно создавать сценарии тестирования и автоматизировать их запуск. Поскольку тестирование выполняется путем запуска обычной программы (эмулятора) на АРМ, то запуск наборов тестов, содержащих множество отдельных тестовых случаев, можно распараллелить на нескольких АРМ. Детерминизм эмуляции упрощает проведение регрессионных тестов. Любое изменение результатов выполнения по сравнению с предыдущими выполнениями других версий того же ПО может быть вызвано только различиями в тестируемой программе, а не аппаратными сбоями или другими случайными изменениями.

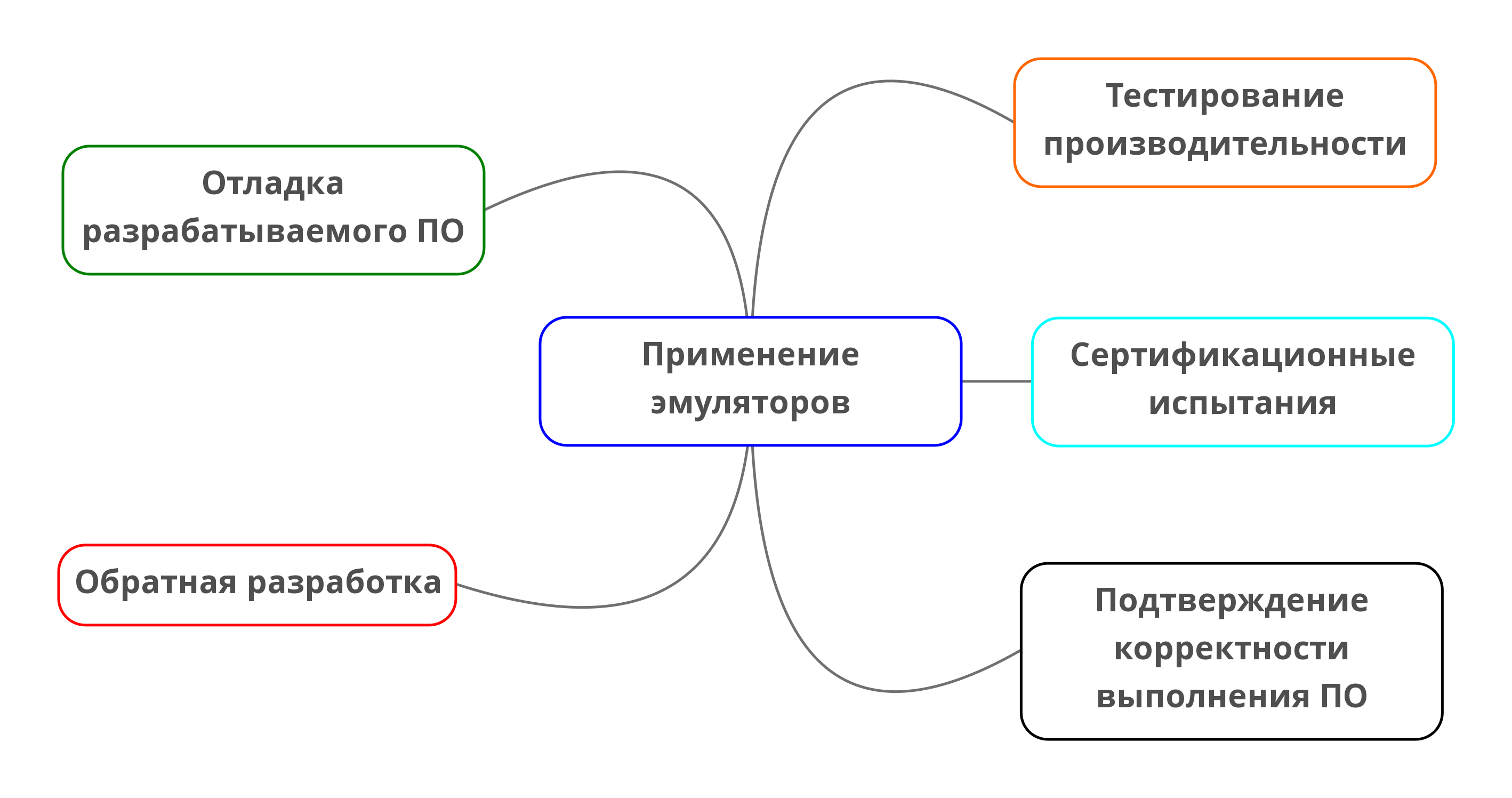


Рисунок – Ментальная карта применения эмуляторов

​При проектировании программного обеспечения всегда встает вопрос о том какой язык программирования выбрать. Для разработки современного программного обеспечения используется большое количество языков программирования. Согласно рейтингу языков программирования на индексе TIOBE в настоящее время около 50-ти языков занимают хотя бы 0.1% доли рынка. Очевидно, что среди них нет наилучшего языка, так как у каждого из них есть своя область применения, своя философия, особенности, различные диалекты и фреймворки. Все это добавляет ещё больше вариантов при выборе языка для написания программного обеспечения. Помимо этого, у языков существует ещё множество других аспектов, по которым можно производить их сравнение. Таким образом, выбор языка программирования для написания современного программного обеспечения является комплексной задачей с большим количеством критериев, по которым необходимо производить оценку.

​Правильный выбор средств и инструментов при разработке оказывает существенное влияние на процесс выполнения, качество результата и возможность дальнейшей поддержки любого проекта, особенно для различных критических систем. Для уменьшения числа вариантов при выборе языка программирования необходимо отталкиваться от программного продукта, который разработчик хочет получить в конечно итоге. Для этого необходимо оценить набор критериев, которыми должен обладать разрабатываемый продукт.

​Хотя большинство языков являются и универсальными, но некоторые из них являются более подходящими для конкретных целей, нежели другие. Например, язык JavaScript отлично подходит для разработки web-приложений, так как имеет готовую экосистему для этого и превосходную переносимость, которая в контексте web-приложений категорически необходима, ведь весь код JavaScript исполняется на персональном компьютере пользователя. С другой стороны, он совсем не подходит для решения задач системного программирования из-за того, что является интерпретируемым языком и обладает недостаточными для системного программирования быстродействием.

​Таким образом языки можно условно разделить по распространенным задачам, для решения которых они подходят лучше всего. При условии, что задача, которую должен решать разрабатываемый программный продукт не является распространенной, у разработчика нет готовых решений, средств и абстракций, которые бы упростили разработку программного продукта. В связи с этим возникает необходимость в проведении сравнительного анализа языков программирования, для выявления наиболее подходящего для поставленной задачи.

## 1.2 Обзор существующих решений

Проведем обзор и анализ имеющихся аналогов разрабатываемого ПО.

**Qemu (Quick Emulator)** - инструмент с открытым исходным кодом, который используется для эмуляции различных архитектур. Qemu имеет большое сообщество разработчиков и, как следствие, хорошую поддержку сообщества разработчиков. Поддерживает два режима эмуляции: пользовательский режим, в котором происходит полная эмуляция системы, включая процессор и периферию, и системный режим в котором происходит только трансляция инструкций и системных вызовов эмулируемой системы. На рисунке 2 приведен пример работы Qemu в системном режиме, где происходит эмуляция операционной системы ReactOS.

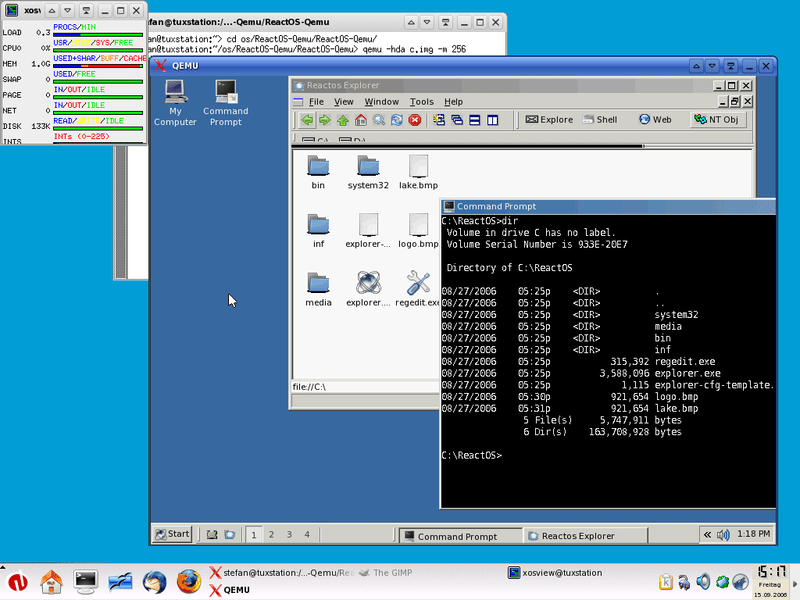


Рисунок 2 - Эмуляция операционной системы ReactOS в эмуляторе Qemu

Qemu обладает такими положительными характеристиками, как наличие интерфейса RSP GDB, моделирование на основе логического соединения шин и высокое быстродействие, так как написан на языке С. С другой стороны это создает большой недостаток в виде высокой сложности написания новых модулей (архитектур, периферии и т.д.).

**KOPYCAT** – отечественный эмулятор аппаратных платформ, позволяет осуществлять низкоуровневую программную эмуляцию произвольных аппаратных систем и их отладку через стандартный интерфейс GDB, поддерживаемый большинством средств разработки. Реализован на языке программирования Kotlin (Java), главным преимуществом которого является эффективность разработки новых решений на нем. На рисунке представлена архитектура эмулятора для эмуляции МК серии STM32.

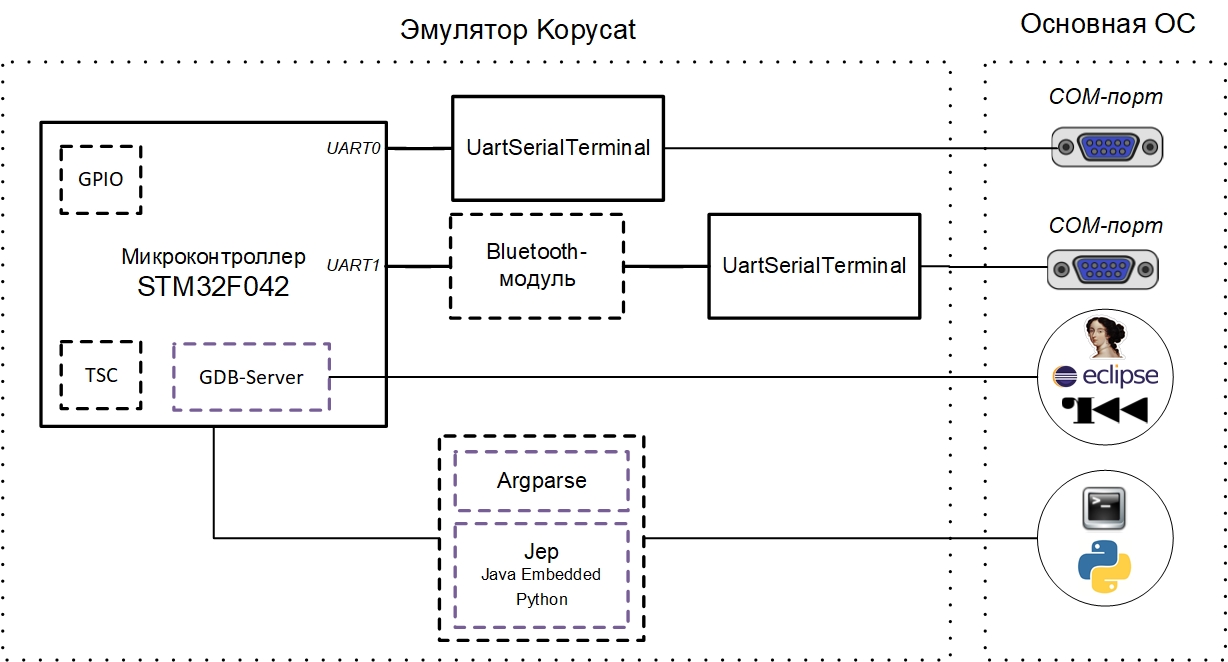


Рисунок – Архитектура эмулятора KOPYCAT для эмуляции МК серии STM32

Одним из конкурентных преимуществ KOPYCAT является наличие удобного SDK-разработчика и документации для создания новых вычислительных ядер и периферийных устройств. При этом возможно два подхода для создания новых модулей: первый — это структурный подход, второй — поведенческое описание. При структурном описании в файле в формате JSON указываются установленные модули и соединения между ними. Также структурное описание допустимо проводить и с помощью языка программирования Kotlin/Java. При поведенческом же описании в модуль могут быть добавлены обработчики записи или чтения с определенных адресов (к которому будет подключен модуль). Внутри обработчика при этом задается, каким образом модуль должен отрабатывать данное событие. Поведенческое описание может быть задано только на языке программирования Kotlin/Java.

**Unicorn** - это легковесный, мультиплатформенный и мультиархитектурный эмулятор процессора. Это не стандартный эмулятор. Он не эмулирует работу всей программы или целой ОС. Он не поддерживает системные команды (такие как открытие файла, вывод символа в консоль и т. д.). Разработчику придется самостоятельно заниматься разметкой памяти и загрузкой данных в нее, а далее просто запускается выполнение с какого-то конкретного адреса. На рисунке представлен графический интерфейс эмуляции BIOS в Unicorn.

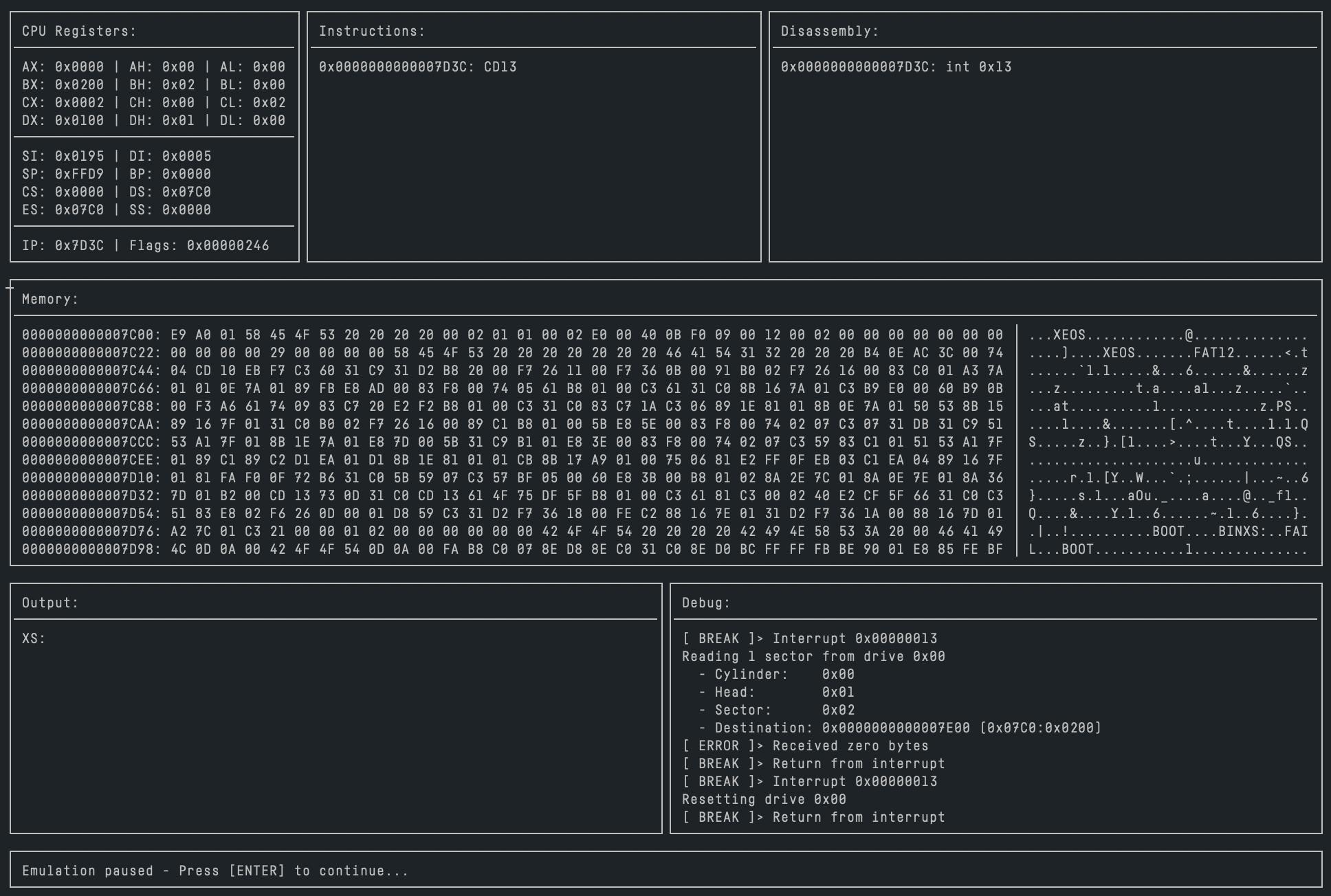


Рисунок – Эмуляция BIOS в Unicorn

К недостаткам данного эмулятора относится отсутствие встроенных абстракций и инструментов по работе ПО, настройка и использование могут быть сложными для новичков или неопытных пользователей, отсутствие системных команд. Таким образом для работы с ним требуется высокая квалификация специалиста.

**Proteus** - пакет программ для автоматизированного проектирования электронных схем, отличительной чертой которого является возможность моделирования работы программируемых устройств (рисунок 3). Эмуляция в Proteus, в отличие от Qemu происходит на уровне электрических сигналов, что значительно замедляет ее скорость.

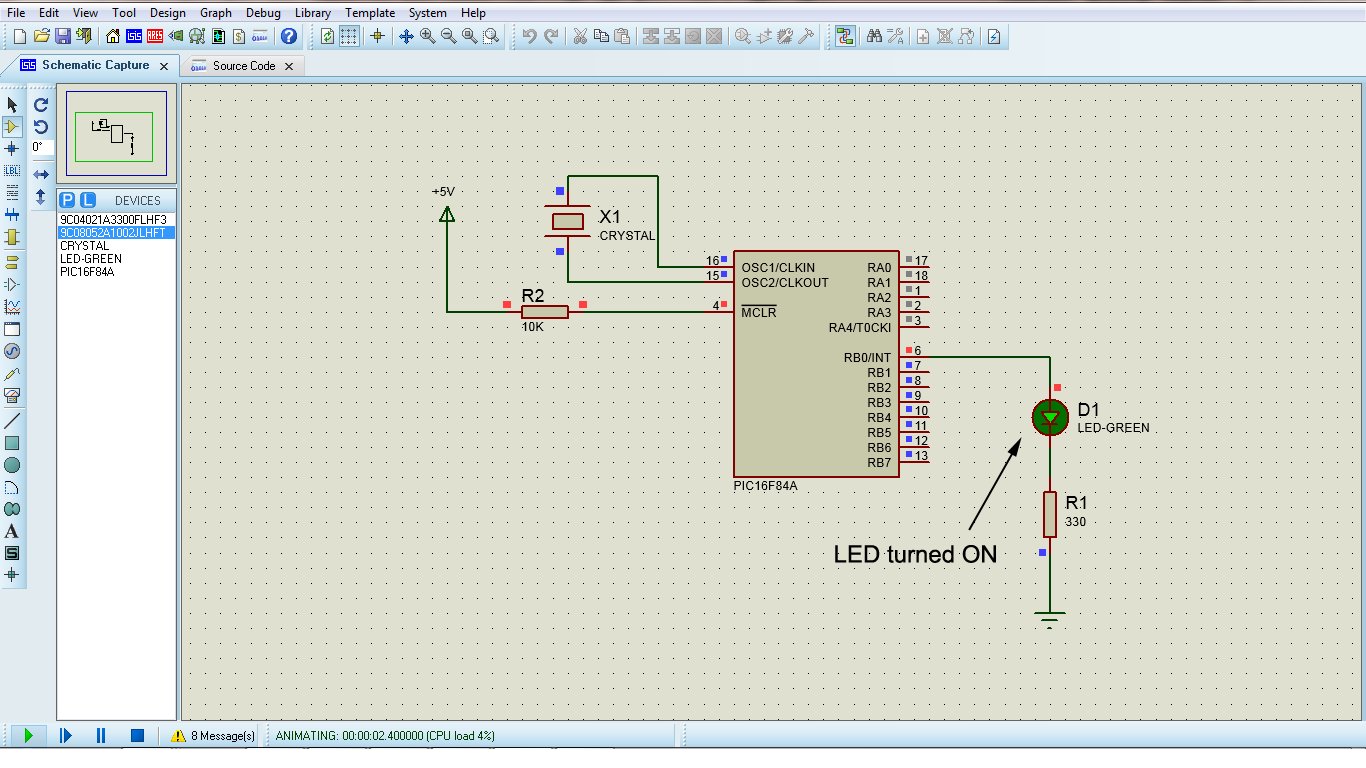


Рисунок - Эмуляция работы микроконтроллера семейства PIC16 в Proteus

Является проприетарным программным обеспечением, но присутствует возможность написания собственных модулей на C++. С этим связаны различные проблемы, такие как отсутствие подробной документации, слабая поддержка пользователей в сообществе разработчиков и специфика языка С++.

**Эмуляторы, встроенные в IDE** - некоторые IDE, например ARM Keil имеют встроенный эмулятор аппаратных платформ (рисунок 4). Такие эмуляторы имеют слабую поддержку сообщества разработчиков, медленны в работе и практически неприменимы в задачах разработки и реверс-инжиниринга встроенных систем.

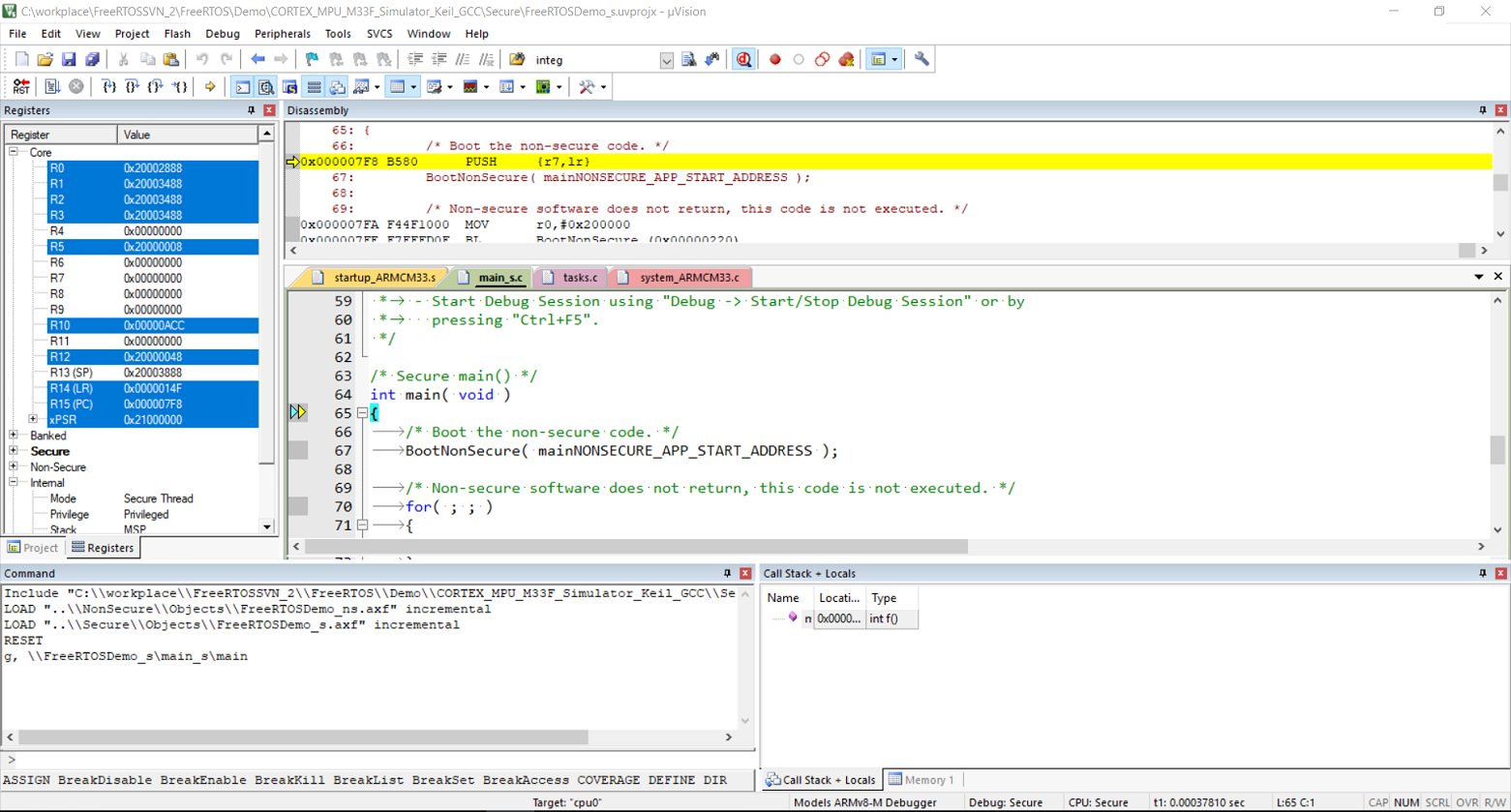


Рисунок - Эмуляция работы ядра ARM Cortex-M33 в IDE Keil uVision

Это связано с отсутствием в таких эмуляторах возможности эмулировать периферийные устройства, которые являются обязательной составляющей любой встраиваемой системы.

Сравнение существующих решений представлено в таблице 1.

Таблица 1. Сравнение инструментов эмуляции аппаратных платформ и моделирования электрических схем

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Программное средство** | **Быстродействие** | **Сложность реализации**  **новых модулей** | **Интерфейс GDB** | **Моделирование электрических сигналов** |
| Qemu | Высокое | Высокая | Есть | Нет |
| KOPYCAT | Среднее | Низкая | Есть | Нет |
| Unicorn | Высокое | Высокая | Есть | Нет |
| Proteus | Низкое | Высокая | Нет | Есть |
| Keil IDE | Низкое | Отсутствует | Есть | Нет |

Трудозатраты на реализацию составляющих эмулируемой системы должны быть минимизированы в связи с постоянной потребностью в эмуляции новых вычислительных систем. Эмулятор должен обладать характеристикой быстродействия в связи с наличием временных затрат на трансляцию процессорных инструкций эмулируемой архитектуры в инструкции хост-системы. Применяемые в существующих решениях средства разработки ПО не позволяют добиться высокого быстродействия эмуляции и низких трудозатрат на реализацию составляющих эмулируемой системы

## Формулирование цели и постановка задач

В итоге рассмотрения данного вопроса можно сказать, что существующие решения для проведения эмуляции аппаратных платформ не обладают характеристикой быстродействия и возможностью простой реализации новых модулей одновременно. При высокой скорости работы таких языков как C/C++ они обладают малым количеством встроенных средств и абстракций, слабо развиты и часто имеют не интуитивное поведение. Таким образом возникает потребность в разработке эмулятора, который будет обладать высокими характеристиками быстродействия и удобством реализации новых модулей.

Основной составным элементом программного обеспечения, от которого зависят данные характеристики является язык программирования, на котором ведут разработку. Также важной составляющей является архитектурная реализация эмулятора.

Производить оценку и сравнение архитектурных реализаций крайне сложная и многокритериальная задача, так как она зависит от множества входных данных, которые будут подаваться на вход эмулятора, и оценить все их нецелесообразно.

Объективно оценить языки программирования по критериям поддерживаемых абстракций, специальных средств и удобства в целом крайне затруднительно. Здесь также влияют огромное количество параметров, таких как специфика языка, его целевая область применения, субъективные предпочтения разработчика и т.д. Таким образом, выбор языков программирования по данному пункту стоит производить на основе мнения сообщества разработчиков о возможностях различных языков.

Стоит отметить, что в отличии от предыдущих критериев, объективно оценить характеристику быстродействия языка программирования возможно. Для этого используются синтетические тесты, написанные на всех тестируемых языках программирования. Они позволяют оценить производительность языка по скорости выполнения определенных задач.

Для проведения оценки скорости исполнения различных языков программирования необходимо аналитическим методом выбрать те, который потенциально больше всего подходят для решения задачи разработки эмулятора аппаратных архитектур. Выбор был произведен на основе мнения сообщества разработчиков о скорости исполнения и наличии поддерживаемых абстракций, специальных средств и удобства разработки на языке программирования в целом. Также были приняты во внимание такие критерии, как переносимость, среда исполнения, масштабируемость, качество экосистемы для разработки, а также поддержка языка среди сообщества разработчиков.

## 1.4 Анализ исполнения программного обеспечения

Программное обеспечение с точки зрения процессора – это последовательность инструкций, описывающих действия, которые необходимо совершить над набором данных. Данные процессорные инструкции могут быть записаны в виде мнемонических кодов на языке Ассемблера. Все языки Ассемблера являются машинно-зависимыми, то есть ориентированы на определенный тип процессорной архитектуры. Примеры мнемоник на языке Ассемблера архитектуры ARM (Advanced RISC Machine) представлены на рисунке 8.

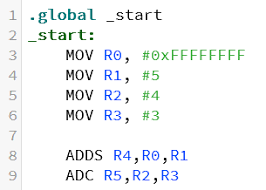


Рисунок 8– Мнемоники инструкций на языке Ассемблера архитектуры ARM

С точки зрения разработчика ПО, это синтаксическая единица, которая соответствует правилам определённого языка программирования, состоящая из определений, операторов или инструкций, необходимых для определённой функции, задачи или решения проблемы. Представлена она в виде текстового файла с набором исходного кода. Пример исходного кода на языке C представлен на рисунке 9.

Чтобы процессор мог выполнить действия, указанные в исходном коде ПО, их необходимо преобразовать в процессорные инструкции. Способов преобразования существует множество. Рассмотрим самые основные и распространенные из них.

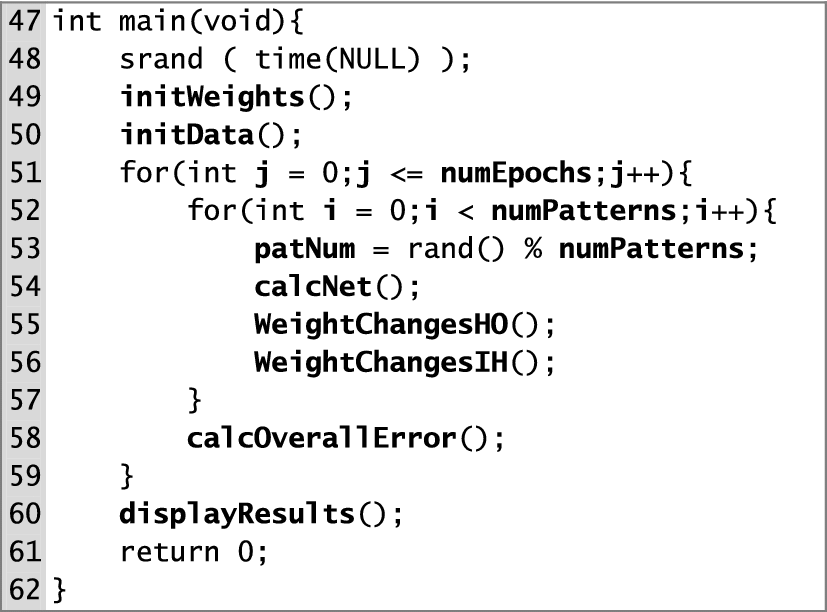


Рисунок 9 – Исходный код на языке программирования C

Компиляция – это процесс преобразования исходного кода ПО в машинный код (набор процессорных инструкций), который может быть непосредственно выполнен процессором. Во время компиляции компилятор анализирует весь исходный код и создает исполняемый файл, который содержит машинный код ПО. Исполняемый файл является машинно-зависимым, таким образом, для каждой процессорной архитектуры необходимо проводить отдельный процесс компиляции. Языки программирования, которые используют компиляцию как способ преобразования исходного кода в процессорные инструкции, называют компилируемыми языками.

Одним из главных преимуществ компилируемых языков является их способность генерировать оптимизированный по критериям времени и памяти машинный код. Большинство компилируемых языков программирования требуют явного объявления размеров и типов данных переменных. Это позволяет более эффективно использовать ресурсы вычислительной системы и на ранних этапах разработки обнаруживать ошибки типизации, что способствует повышению надежности и безопасности ПО.

Разработка ПО на компилируемых языках может быть более трудоемкой, поскольку такие языки обычно требуют более строгого синтаксиса и языковых конструкций. Ошибка в исходном коде может привести к сложно обнаружимым ошибкам компиляции, что требует дополнительного времени на отладку, поиск и исправление причины ошибки. Компилируемые языки программирования требуют повторной компиляции программного кода при каждом изменении, что может быть трудоемким и затратным процессом при работе с большим ПО. Это приводит к снижению гибкости и скорости разработки ПО. К распространенным компилируемым языкам программирования относятся C, C++, Rust, Go.

Интерпретация – это процесс выполнения исходного кода путем его анализа и последовательной интерпретации отдельным ПО интерпретатором во время исполнения ПО. В отличие от компилированных языков программирования, где исходный код предварительно преобразуется в процессорные инструкции и сохраняется в виде исполняемого файла, при интерпретации ПО каждая синтаксическая единица исходного кода анализируется и выполняется непосредственно в процессе работы ПО. Языки программирования, которые используют интерпретацию как способ трансляции исходного кода в машинный код, называют интерпретируемыми языками.

Одним из основных преимуществ интерпретации программ является ее платформенная независимость. Интерпретируемые языки программирования в большинстве случаев могут быть выполнены на различных платформах без необходимости модификации исходного кода. При интерпретации ПО проще отслеживать и исправлять ошибки, так как интерпретатор выполняет код построчно. Интерпретируемые языки программирования обычно поддерживают динамическую типизацию, что означает, что типы переменных могут быть определены во время выполнения ПО.

Однако, интерпретация ПО также имеет свои минусы. Один из основных недостатков – это низкая производительность по сравнению с компилируемыми языками программирования. Поскольку интерпретатор производит анализ и выполнение исходного кода, время выполнения может быть дольше, чем у ПО, скомпилированного в машинный код. Это особенно заметно при обработке больших объемов данных или выполнении высоконагруженных вычислений. К распространенным интерпретируемым языкам программирования относятся Python, JavaScript, PHP, Ruby, SQL, bash.

Существуют также байт-код языки программирования, которые используют как компиляцию, так и интерпретацию для преобразования исходного кода в машинный код. Байт-код языки программирования представляют исходный код программы в промежуточной форме. Компилятор преобразует исходный код в последовательность инструкций, которые являются низкоуровневыми и промежуточными для выполнения на виртуальной машине или интерпретаторе. Этот промежуточный код называется байт-кодом, поскольку он представлен в виде последовательности байтов.

Одним из основных преимуществ использования байт-кода является платформенная независимость. Построение виртуальной машины или интерпретатора, который выполняет байт-код, позволяет запускать ПО на разных платформах без изменения исходного кода. Байт-код также часто используется для улучшения производительности ПО. Он может быть скомпилирован наиболее эффективным образом для конкретной виртуальной машины или интерпретатора, что приводит к повышению эффективности выполнению ПО. К распространенным байт-код языкам программирования относятся Java, C#, Kotlin и Scala.

С использованием результатов проведенного анализа была составлена ментальная карта классификации языков программирования по критерию преобразования исходного кода в процессорные инструкции, представленная на рисунке 10.



Рисунок – Ментальная карта классификации языков программирования

По результатам проведенного анализа можно сделать вывод о том, что каждый из представленных типов языков программирования обладает своими преимуществами и недостатками. Как следствие, каждый из них находит применение в различных областях разработки ПО. Выбор типа языка программирования определяется требованиями к задаче, которую он должен решать.

## 1.5 Анализ средств проектирования ПО

Для проведения исследования средств разработки ПО необходимо выбрать из множества существующих те, которые вероятнее всего обладают характеристиками быстродействия и эффективности разработки с использованием их. Под средством разработки в данном исследовании понимаются языки программирования, на которых может быть реализован эмулятор аппаратных платформ.

Для выбора языков программирования для исследования составим критерии, по которым будем оценивать языки программирования:

- скорость -- насколько быстро язык исполняет требуемую задачу;

- актуальность -- насколько язык современен и какой он имеет потенциал в будущем для дальнейшего перспективного развития продукта;

- распространенность – популярность языка среди сообщества разработчиков, и как следствие количество доступной информации и документации в сети интернет;

- количество интегрированных абстракций и синтаксиса -- набор конструкций, абстракций, парадигм и специальных средств, которые повышают эффективность разработки с помощью языка;

- экосистема -- наличие качественных IDE с автоматизацией рутинных процессов, наличие компилятора или интерпретатора без программных ошибок, наличие эффективной системы отладки разрабатываемого продукта.

Проведем обзор и анализ выбранных языков программирования для исследования их быстродействия:

**Python** - интерпретируемый скриптовый язык программирования, разработанный в конце 1980-х годов Гвидо ван Россумом. Он обладает простым и элегантным синтаксисом и является самым популярным языком программирования на данный момент по версии индекса TIOBE. Одним из ключевых преимуществ Python является его простота использования. Python предлагает обширные встроенные абстракции и имеет простой и понятный синтаксис, что делает его легким в изучении. Python является интерпретируемым языком, что делает его гораздо медленнее по сравнению с нативными языками, но существуют разные способы оптимизации по критерию быстродействия. Python известен своим множеством библиотек и фреймворков, которые делают его очень мощным для различных задач.

**Cython** - язык программирования, который представляет собой надстройку Python с возможностью явного определения типов данных. Он позволяет разработчикам писать высокопроизводительный код, близкий к языку C, который можно легко интегрировать с существующим кодом Python. Одним из главных преимуществ Cython является его производительность. Благодаря явному определению типов данных и использованию статической типизации, Cython может генерировать оптимизированный машинный код. Cython активно используется в реальных проектах, где требуется компромисс между простотой разработки и производительностью. Cython не так распространен, как Python, но все же имеет свою аудиторию и растет в популярности. Cython предоставляет ту же интеграцию и синтаксис, что и Python, добавляя возможность использования статической типизации.

**JavaScript** - язык программирования, разработанный Netscape Communications Corporation в 1995 году. Он широко используется для создания интерактивных веб-страниц и веб-приложений, и наряду с HTML и CSS, является одним из основных языков для разработки фронтенда. JavaScript также может быть использован для разработки серверной части веб-приложений с использованием платформы Node.js. Это позволяет разрабатывать полноценные приложения, которые могут работать как на стороне клиента, так и на стороне сервера.

JavaScript является интерпретируемым языком, и его производительность зависит от интерпретатора, но множество оптимизаций делают его достаточно быстрым. JavaScript является одним из самых распространенных языков программирования и имеет огромную аудиторию разработчиков. JavaScript предлагает множество интегрированных абстракций и имеет легкий для понимания синтаксис.

**C#** - был разработан Microsoft в 2000 году и является ключевым языком для платформы .NET. C# сочетает в себе сильные стороны C++ и удобство Java, что делает его очень популярным среди разработчиков. Одним из основных преимуществ C# является его объектно-ориентированность. Он предоставляет механизмы наследования, инкапсуляции и полиморфизма для построения модульных и расширяемых программ. Кроме того, C# имеет сборку мусора для автоматического управления памятью, что облегчает разработку и устраняет множество ошибок, связанных с утечками памяти.

C# медленнее относительно нативных языков, но современные оптимизации делают его довольно эффективным. C# очень распространен, особенно для разработки приложений для Windows и игр на платформе Unity. C# предлагает широкий набор интегрированных абстракций и имеет синтаксис, являющийся комбинацией C++ и Java. C# сопровождается обширной экосистемой, включая интегрированную среду разработки Visual Studio и множество библиотек для различных целей.

**Kotlin** - язык программирования, разработанный JetBrains в 2011 году, специально для разработки приложений на платформе Java. Kotlin является статически типизированным языком, который обладает простым и понятным синтаксисом и представляет собой расширение возможностей Java. Одним из основных преимуществ Kotlin является его совместимость с Java. Это позволяет разработчикам легко интегрировать Kotlin в существующие проекты на Java и переиспользовать код. Кроме того, Kotlin предоставляет дополнительные возможности, такие как расширения функций и поддержка нулевых значений, что упрощает написание безопасного и эффективного кода.

Является байткод языком, компилируется в JVM байткод, и проявляет хорошую производительность, но все ещё недостающую до уровня нативных языков. Kotlin не так широко распространен, как Java, но стремительно растет. Kotlin предоставляет современную синтаксическую конструкцию и имеет множество усовершенствований по сравнению с Java.

**Rust** - относительно новый системный язык программирования, разработанный Mozilla Research в 2010 году. Прежде всего, Rust известен своими возможностями по повышению безопасности и предотвращению множества типичных ошибок программирования. Одним из ключевых преимуществ Rust является его система контроля заимствования, которая позволяет предотвратить ошибки, связанные с несогласованным доступом к памяти. Это делает Rust особенно безопасным для разработки системного программного обеспечения, где низкоуровневые ошибки могут привести к серьезным проблемам.

Rust также обладает мощной системой типов и статическим анализом, что позволяет выявлять ошибки во время компиляции. Это сильно упрощает отладку и повышает качество разрабатываемого кода. Кроме того, Rust предоставляет механизм сборки мусора для управления памятью. Rust является компилируемым нативным языком программирования, который предлагает отличную производительность. Rust все еще развивается и не так широко распространен, как другие языки, но найдет свое место в специализированных областях. Имеет растущую экосистему с библиотеками и инструментами для разработки.

**C** - разработанный в начале 1970-х годов, является одним из наиболее широко используемых языков в программной индустрии. Он известен своей мощностью и эффективностью, а также широкой поддержкой различных архитектур и операционных систем. Одним из главных преимуществ C является его низкоуровневое программирование, что позволяет разработчикам иметь полный контроль над памятью и процессором. Это делает C особенно подходящим для разработки системного и встроенного программного обеспечения, операционных систем и драйверов устройств.

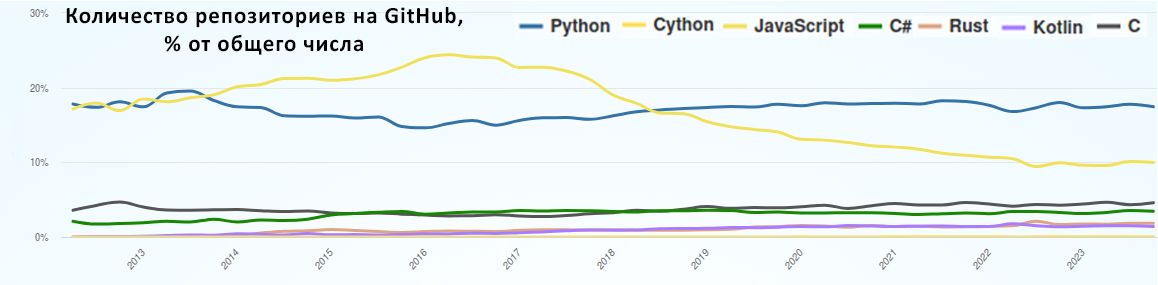


Рисунок – График количества репозиториев на GitHub по годам для исследуемых средств разработки ПО

В данном исследовании C используется как эталон языка программирования с наибольшим быстродействием и распространенностью. Таким образом для разработки эмулятора аппаратных платформ он не рассматривается в связи с отсутствием поддержки объектно-ориентированного программирования и отсутствием интегрированных абстракций.

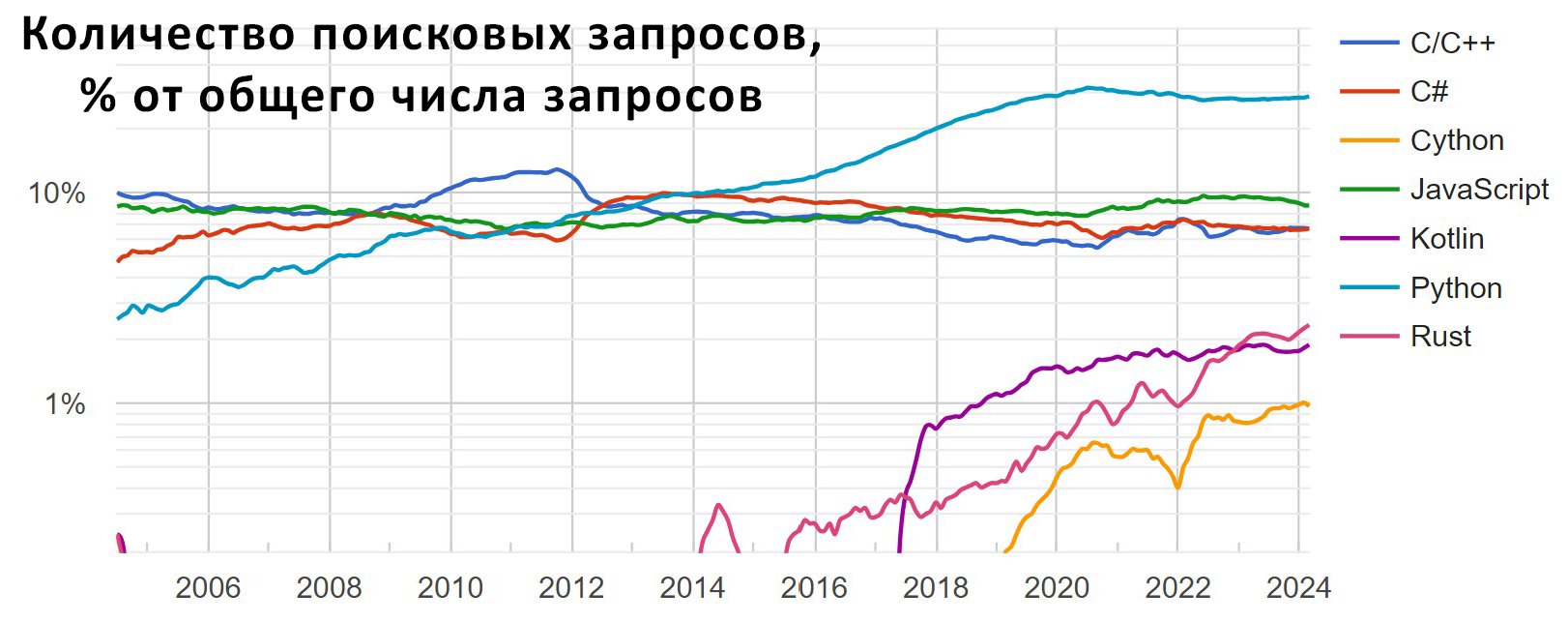


Рисунок – График количества поисковых запросов по годам для исследуемых средств разработки ПО

В результате анализа для исследования были выбраны 7 языков программирования, на которых будет производиться реализация исследуемых алгоритмов и сравнительный анализ получившихся результатов их выполнения. Стоит отметить, что на некоторых из выбранных языков программирования реализованы эмуляторы встраиваемых систем, рассмотренные выше. Таким образом, в дальнейших исследованиях можно будет провести сравнительный анализ по критерию быстродействия разработанного эмулятора встраиваемых систем на выбранном по результатам данного исследования языке программирования и существующих решений эмуляторов встраиваемых систем.

## 1.6 Метод оценки трудоемкости разработки ПО

Анализ функциональных точек — стандартный метод измерения размера программного продукта с точки зрения пользователей системы. Метод разработан Аланом Альбрехтом (Alan Albrecht) в середине 70-х. Метод был впервые опубликован в 1979 году. В 1986 году была сформирована Международная Ассоциация Пользователей Функциональных Точек (International Function Point User Group — IFPUG), которая опубликовала несколько ревизий метода.

Метод предназначен для оценки на основе логической модели объема программного продукта количеством функционала, востребованного заказчиком и поставляемого разработчиком. Несомненным достоинством метода является то, что измерения не зависят от технологической платформы, на которой будет разрабатываться продукт, и он обеспечивает единообразный подход к оценке всех проектов в компании.

При анализе методом функциональных точек надо выполнить следующую последовательность шагов:

* Определение типа оценки.
* Определение области оценки и границ продукта.
* Подсчет функциональных точек, связанных с данными.
* Подсчет функциональных точек, связанных с транзакциями.
* Определение суммарного количества не выровненных функциональных точек (UFP).
* Определение значения фактора выравнивания (FAV).
* Расчет количества выровненных функциональных точек (AFP).

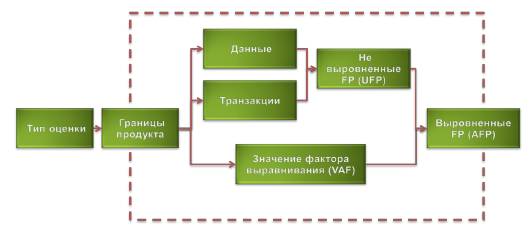


Рисунок - Процедура анализа по методу функциональных точек

Первое, что необходимо сделать, это определить тип выполняемой оценки. Метод предусматривает оценки трех типов:

* Проект разработки. Оценивается количество функциональности поставляемой пользователям в первом релизе продукта.
* Проект развития. Оценивается в функциональных точках проект доработки: добавление, изменение и удаление функционала.
* Продукт. Оценивается объем уже существующего и установленного продукта.

Второй шаг — это определение области оценки и границ продукта. В зависимости от типа область оценки может включать:

* Все разрабатываемые функции (для проекта разработки)
* Все добавляемые, изменяемые и удаляемые функции (для проектов поддержки)
* Только функции, реально используемые, или все функции (при оценке продукта и/или продуктов).

Третий шаг. Границы продукта определяют:

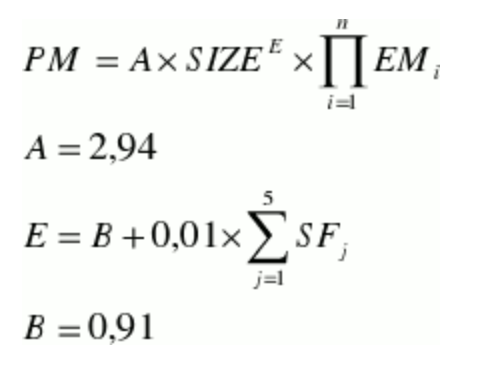
* Что является «внешним» по отношению к оцениваемому продукту.
* Где располагается «граница системы», через которую проходят транзакции передаваемые или принимаемые продуктом, с точки зрения пользователя.
* Какие данные поддерживаются приложением, а какие — внешние.

Метод анализа функциональных точек ничего не говорит о трудоемкости разработки оцененного продукта. Вопрос решается просто, если компания разработчик имеет собственную статистику трудозатрат на реализацию функциональных точек. Если такой статистики нет, то для оценки трудоемкости и сроков проекта можно использовать метод COCOMO II.

Методика COCOMO позволяет оценить трудоемкость и время разработки программного продукта. Впервые была опубликована Бари Боэмом в 1981 году в виде результат анализа 63 проектов компании «TRW Aerospace». В 1997 методика была усовершенствована и получила название COCOMO II. Калибровка параметров производилась по 161 проекту разработки. В модели используется формула регрессии с параметрами, определяемыми на основе отраслевых данных и характеристик конкретного проекта.

Различаются две стадии оценки проекта: предварительная оценка на начальной фазе и детальная оценка после проработки архитектуры.

Формула оценки трудоемкости проекта в чел.\*мес. имеет вид:



где

* SIZE — размер продукта в KSLOC
* EMi — множители трудоемкости
* SFj — факторы масштаба
* n=7 — для предварительной оценки
* n=17 — для детальной оценки

Главной особенностью методики является то, что для того, чтобы оценить трудоемкость, необходимо знать размер программного продукта в тысячах строках исходного кода (KSLOC, Kilo Source Lines Of Code). Размер программного продукта может быть, например, оценен экспертами с применением метода PERT.

Если мы провели анализ продукта методом функциональных точек, то его размер может быть рассчитан с использованием собственных статистических данных или с использованием статистики по отрасли

Оценка трудоемкости должна быть вероятностным утверждением. Это означает, что для нее существует некоторое распределение вероятности, которое может быть очень широким, если неопределенность высокая, или достаточно узким, если неопределенность низкая.

## 1.7 Разработка методики исследования быстродействия современных средств проектирования ПО

Метод для проведения исследования был разработан с целью создания стабильного окружения, в котором можно было бы оценить влияние языка программирования на скорость исполнения алгоритмов. В качестве основы использовались стандартные сценарии тестирования языков программирования на быстродействие.

Тестовое окружение представляет собой ПО, в которое можно расширять за счет добавления новых конфигураций и тестов на различных языках программирования. Python является одним из самых популярных языков программирования в мире. Он широко используется в различных областях, таких как научные исследования, веб-разработка, анализ данных и машинное обучение. Благодаря популярности Python имеет большую поддержку и активное сообщество разработчиков, что обеспечивает наличие множества ресурсов, документации, библиотек и инструментов для разработки.

Python известен своей простотой и лаконичностью синтаксиса, что делает код на этом языке легко читаемым и понятным. Это позволяет разработчикам быстро разрабатывать и поддерживать код, а также улучшает его читаемость для других разработчиков, облегчая дальнейшую работу над проектом тестового окружения. Кроме того, простота Python способствует ускорению процесса разработки и уменьшению вероятности возникновения ошибок.

Python обладает обширной экосистемой библиотек и инструментов, которые предоставляют различные возможности для написания и оптимизации кода. Например, библиотека NumPy предоставляет высокопроизводительные структуры данных и функции для научных вычислений, а библиотека Pandas позволяет эффективно работать с данными. Благодаря наличию таких библиотек в Python возможно легко реализовать и сравнить алгоритмы на различных языках программирования.

Python является кросс-платформенным языком программирования, что позволяет запускать код на различных операционных системах, таких как Windows, macOS и Linux. Это облегчает тестирование и сравнение быстродействия нескольких языков программирования на разных платформах.

Python легко интегрируется с другими языками программирования и системами. Например, можно легко вызывать код на C/C++ из Python с использованием специальных библиотек. Такая интеграция позволяет сравнить производительность разных языков программирования с использованием одних и тех же алгоритмов.

В итоге, выбор языка программирования Python для написания тестового окружения для сравнения быстродействия нескольких языков программирования обосновывается его популярностью, простотой и читаемостью кода, наличием обширной экосистемы библиотек и инструментов, кросс-платформенностью и удобством интеграции с другими языками программирования. Все эти факторы позволяют достичь эффективности и точности при сравнении быстродействия различных языков программирования с использованием заданных алгоритмов.

Ключевым методом исследования является проведение тестирования быстродействия выполнения заданных алгоритмов различными языками программирования и последующее сравнение результатов проведенных экспериментов.

Для проведения тестирования быстродействия языков программирования было разработано программное средство «FPLB» (Flexible Programming Language Benchmarking) с необходимым для тестирования окружением, в проектировании которого были использованы исследования, представленные в работе «Cross-Language Compiler Benchmarking». Блок-схема алгоритма работы программного средства «FPLB» представлена на рисунке .

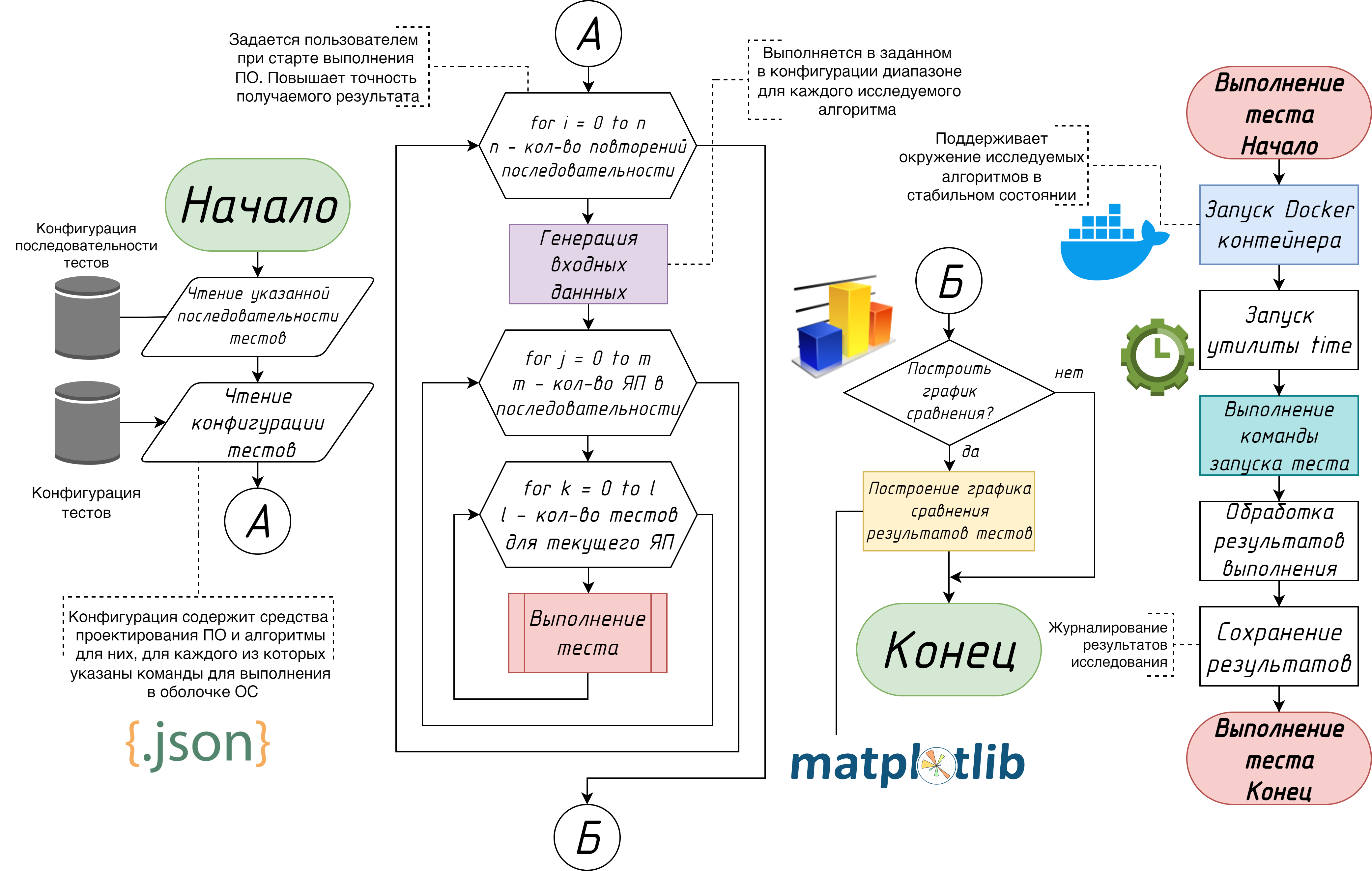


Рисунок – Алгоритм исследования быстродействия современных средств проектирования ПО для разработки эмулятора встраиваемых систем

Для оценки быстродействия выполнения тестируемого ПО был использован метод, предложенный в исследовании «Сравнительное тестирование языков программирования». Значение времени исполнения тестируемого ПО на определенном языке программирования измеряется с помощью ПО «time». Это стандартная unix-утилита, которая выводит статистическое сообщение о времени, затраченном на выполнение переданного на вход утилите ПО.

**2 РАЗРАБОТКА ПО ИССЛЕДОВАНИЯ БЫСТРОДЕЙСТВИЯ СРЕДСТВ ПРОЕКТИРОВАНИЯ ПО**

## 2.1 Реализация разработанной методики исследования быстродействия современных средств проектирования ПО

Программное средство «FPLB» позволяет в автоматизированном режиме производить запуск различных заранее подготовленных алгоритмов для заданных языков программирования. Оно также позволяет вести журналирование результатов тестирования, строить гистограмму на основе результатов тестирования, генерировать необходимые входные данные для проведения тестирования, а также запускать последовательность одинаковых тестов и вычислять среднее значение результатов проведенных тестов. Вывод справочных данных об использовании программного средства «FPLB» представлен на рисунке .

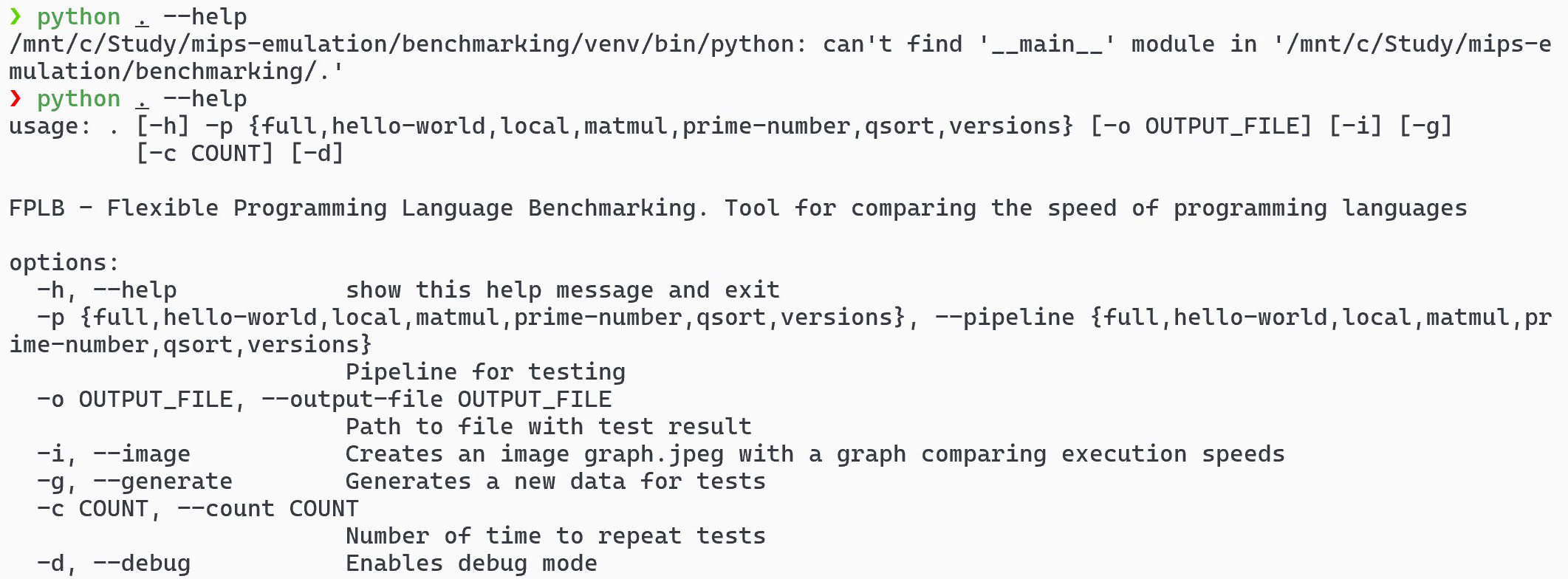


Рисунок 7 - Вывод справочных данных о использовании программного средства «FPLB»

Для разработки программного средства «FPLB» был выбран язык программирования Python, так как на нем легко разрабатывать, отлаживать и поддерживать ПО. Он имеет большое и активное сообщество разработчиков, что обеспечивает наличие множества ресурсов, документации, библиотек и инструментов для разработки. В качестве окружения для исполнения тестов была выбрана контейнерная платформа Docker. Это распространенный легкий и автономный выполняемый контейнер, который включает в себя всё необходимое окружение для запуска ПО, в том числе библиотеки, системные инструменты и среду выполнения. Он позволяет поддерживать окружение тестируемых алгоритмов в одном и том же стабильном состоянии, что позволяет снизить влияние различных процессов, выполняющихся в вычислительной системе, на которой проходит тестирование.

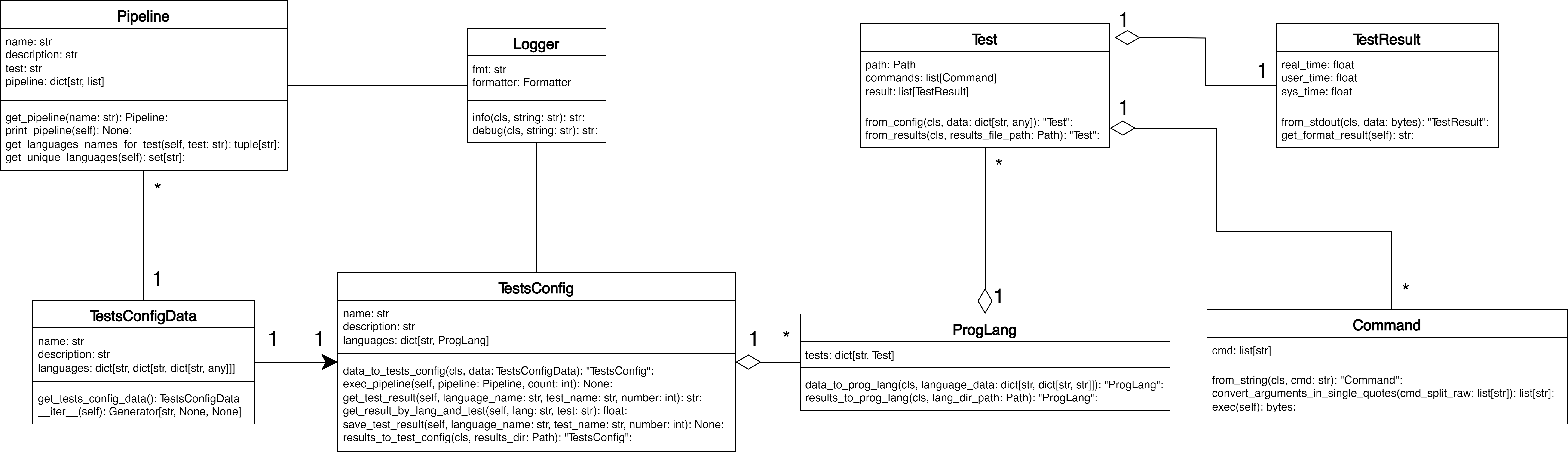


Рисунок – Диаграмма классов разработанного ПО «FPLB»

Выбор алгоритмов, скорость выполнения которых будет тестироваться, был основан на работе «Исследование скоростей выполнения базовых математических задач популярных языков программирования» [24]. Так как целью работы является исследование характеристики быстродействия языков программирования в выполнении алгоритмов, наиболее часто используемых в процессе эмуляции встраиваемых систем, то необходимо выбрать те алгоритмы, которые содержат в себе операции, чаще всего используемые в работе эмулятора. Этими операциями являются чтение и запись данных в память, а также базовые арифметические операции. Это связано со спецификой работы современных процессорных архитектур [25]. Таким образом, требуются алгоритмы, содержащие в себе множество подобных операций.

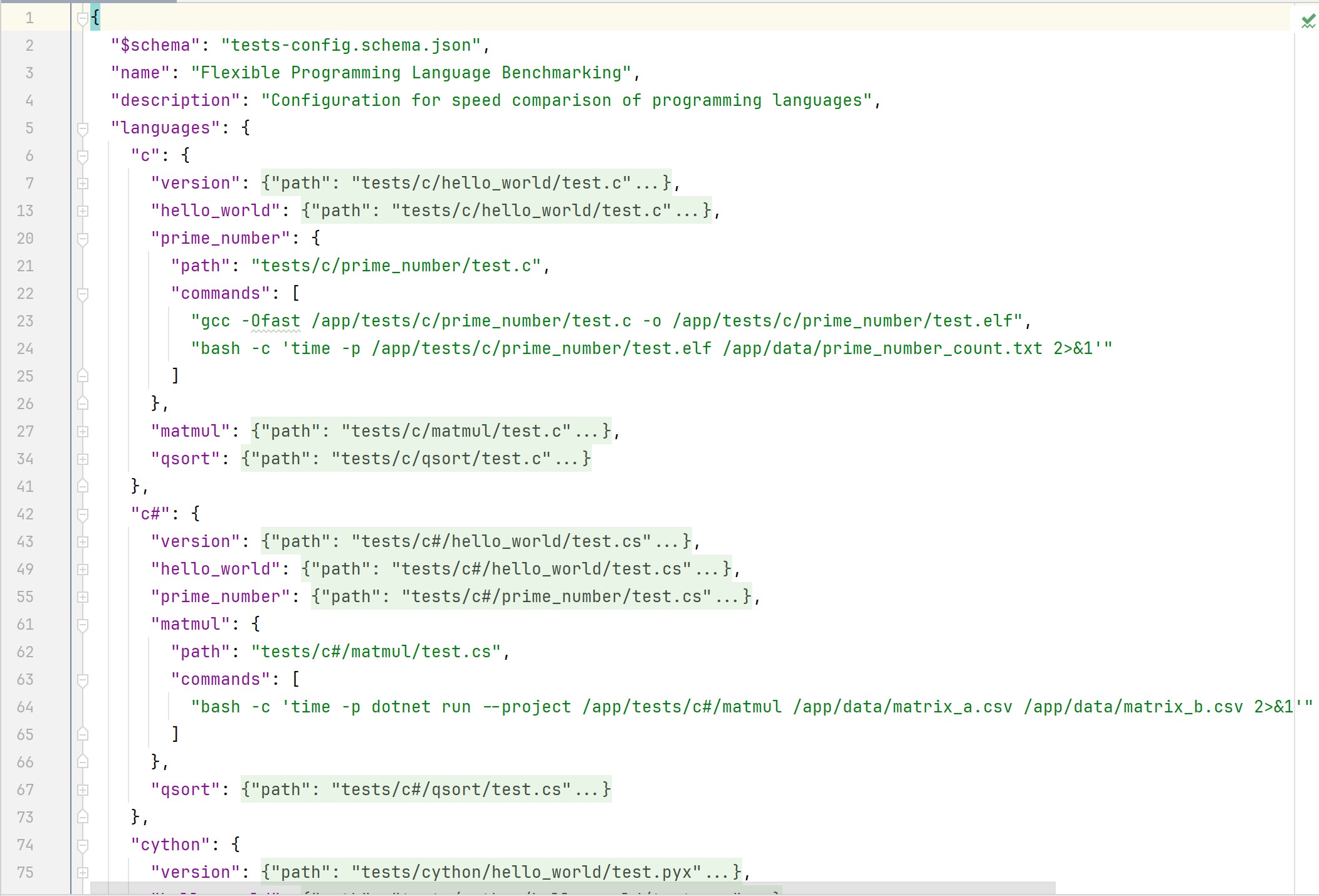


Рисунок - Четырехуровневый иерархический дескриптор конфигурации «FPLB»

Визуализация результатов была осуществлена с использованием метода, применяемого в исследовании «Cross-Language Compiler Benchmarking» [15]. Для этого была использована библиотека для языка Python «MatPlotLib». Она позволяет проводить визуализацию данных двумерной и трёхмерной графикой. Таким образом можно наглядно оценить соотношение скоростей выполнения алгоритмов различными языками программирования.

Программное средство для проведения тестирования установлено на виртуальную машину с дистрибутивом ОС семейства Linux Ubuntu 22.04. Параметры виртуальной машины:

- Процессор AMD Ryzen 5 5600U with Radeon Graphics 2 ядра;

- Оперативная память 4 Гб;

- Постоянная память 60 Гб;

- Видеоадаптер не используется.

Тестирование проводится в Docker контейнере, созданном на основе ОС семейства Linux Ubuntu 22.04. Он позволяет поддерживать окружение в одном и том же стабильном состоянии, что позволяет снизить влияние различных процессов, происходящих на вычислительной системе, на которой происходит тестирование.

В программном средстве «FPLB» для тестирования предусмотрена конфигурация последовательности тестов на определенных языках программирования и определенных алгоритмов. Таким образом, созданы 3 конфигурации на каждый из исследуемых алгоритмов. Программное средство «FPLB» также имеет функциональность запуска последовательности заданного количества одних и тех же тестов. Это позволяет провести множество одинаковых тестов и посчитать среднее арифметическое полученных результатов. Полученный результат округляется до 2х знаков после запятой. Для каждого теста проводится генерация нового набора входных данных одинаковой заданной размерности. Тестирование проводилось при 20ти циклах повторов тестов. Исходя из статистических методов, данный метод тестирования позволит получить результат, приближенный к действительности.

Для алгоритма поиска простых чисел методом перебора программное средство «FPLB» генерирует значения количества искомых простых чисел в диапазоне от 10000 до 15000. Для алгоритма перемножения матриц генерируются матрицы размером 2000 на 2000 элементов с значениями в диапазоне от 0 до 65536. Для алгоритма быстрой сортировки генерируются массивы данных размером 1000000 элементов с числами в диапазоне от 0 до 4294967294.

## 2.2 Выбор алгоритма для проведения исследования

При написании тестов на языках программирования, которые необходимо сравнить, следует использовать только базовые синтаксические конструкции языка, для того чтобы максимально объективно производить оценку производительности. Это связано с тем, что в рассматриваемых языках могут быть свои уникальные синтаксические абстракции, принцип работы которых во внутреннем устройстве языка неизвестен, и не может быть точно скопирован в других языках программирования.

Как уже было изложено ранее, одним из наиболее важных и объективных критериев сравнения языков программирования является их быстродействие. В соответствие нашей цели важно, чтобы результаты тестов показывали скорость выполнения тех алгоритмов, которые чаще всего используются в работе эмулятора архитектур. Под это условие подходят различные математические операции. Также стоит протестировать языки на выполнение какого-либо универсального алгоритма, часто выполняющегося для многих задач в разных приложениях. Сам программный код на разных языках должен быть написан максимально идентично с точки зрения вычислительной сложности, иначе данные будут необъективными, так как, даже исходя из теоретического расчёта, код на одном языке будет «проигрывать» коду на другом. Также очевидно, что тестирование должно проводиться на одной и той же платформе, так как производительность конкретной вычислительной машины напрямую влияет на результат.

В открытых источниках можно найти различные сравнения производительности языков, однако данные исследования не удовлетворяют всем приведённым выше требованиям. Например, в [1] проверяется только один алгоритм, присутствуют не все нужные нам языки; в [2] нет математических алгоритмов, малая выборка языков; в [3] присутствуют не все нужные языки, алгоритмы выполнения операций на разных языках отличаются. Таким образом, нам потребовалось выбрать алгоритмы и реализовать тесты самим.

Для проведения исследования быстродействия языков программирования были выбраны 3 алгоритма. Операции, выполняемые в них, являются самыми распространенными операциями, которые выполняет процессор. Следовательно, эти операции будут выполнятся чаще всего в эмуляторе встраиваемых систем. Необходимо, чтобы ПО, разработанное на языке программирования, на котором будет реализован эмулятор, максимально быстро выполняло именно эти операции. В приведенных алгоритмах не рассматриваются вспомогательные операции, такие как выделение памяти, чтение данных, вывод результатов. Рассмотрим выбранные алгоритмы подробнее.

**Алгоритм перемножения матриц** — это процесс, который состоит в умножении двух матриц для получения третьей матрицы, называемой матрицей произведения. Алгоритм реализован с использованием циклов, где каждому элементу матрицы произведения ставится в соответствие сумма произведений элементов соответствующих строк и столбцов. Отличие в реализации заключается в том, что вторая матрица транспонируется для более эффективного использования памяти. Блок-схема алгоритма перемножения матриц представлена на рисунке .

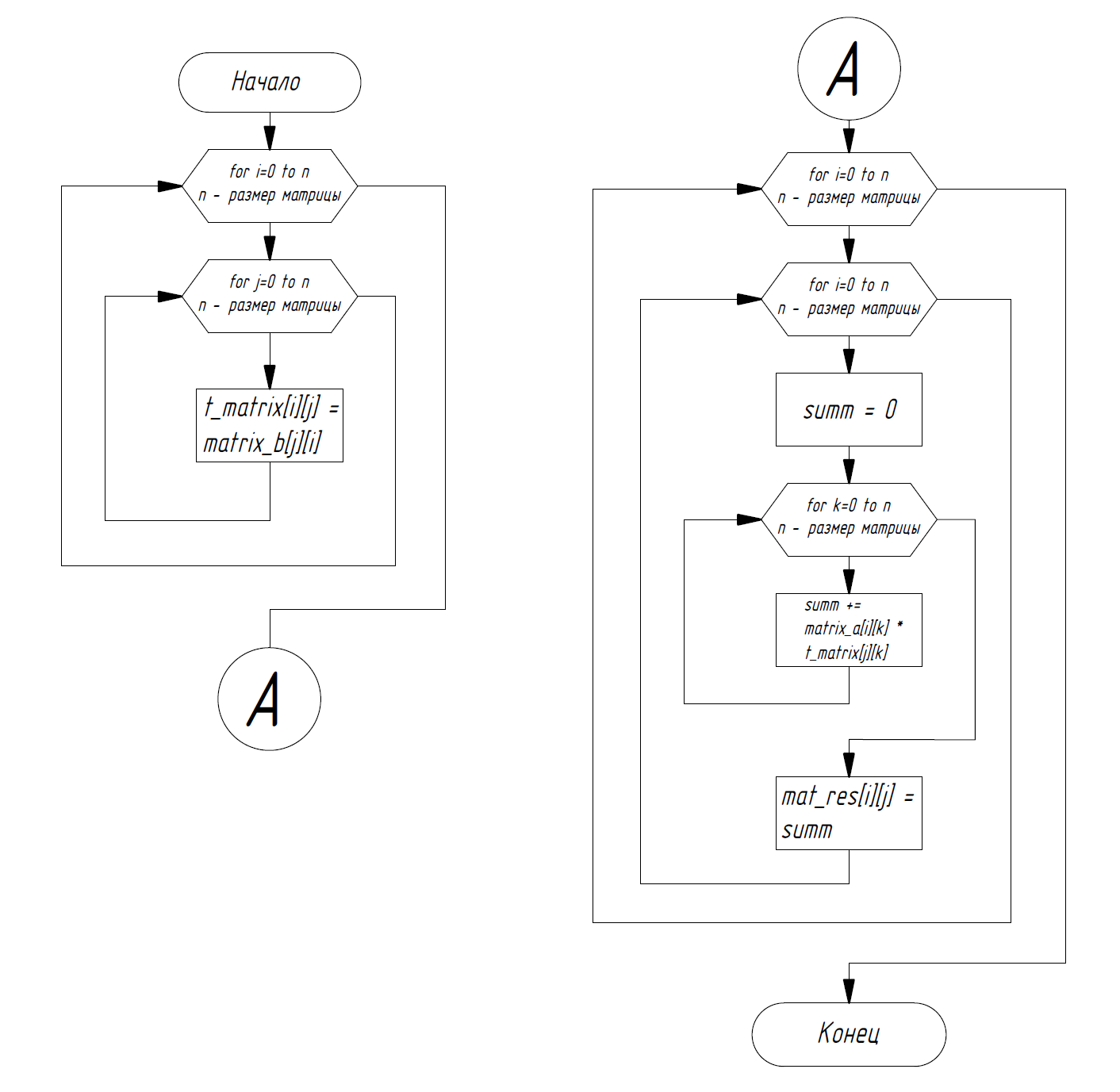


Рисунок – Блок-схема алгоритма перемножения матриц

Алгоритм перемножения матриц имеет сложность O(n^3), где n – размер стороны матрицы, так как содержит три вложенных цикла. Временная сложность этого алгоритма зависит от размеров входных матриц и может быть существенной для больших матриц.

**Быстрая сортировка** представляет собой один из наиболее эффективных методов сортировки массивов данных. Алгоритм быстрой сортировки реализуется путем выбора опорного элемента из исходного массива и последующего разделения массива на две подгруппы: элементы, меньшие опорного, и элементы, большие опорного. Эти подгруппы рекурсивно подвергаются процессу сортировки с использованием быстрой сортировки, пока не будет достигнуто конечное упорядочивание. Этот процесс называется "разбиением".

Алгоритм быстрой сортировки обладает высокой эффективностью, в среднем имеет временную сложность , где - количество элементов в массиве. Однако, в худшем случае, его временная сложность может достигать , что происходит при неудачном выборе опорного элемента или в случае, когда массив уже упорядочен по возрастанию или убыванию.

Блок-схема алгоритма быстрой сортировки представлена на рисунке .

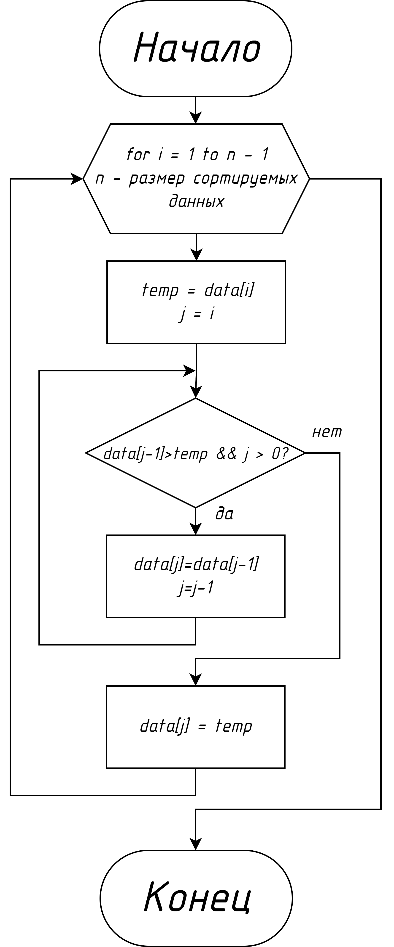


Рисунок – Блок-схема алгоритма быстрой сортировки

**Алгоритм поиска простых чисел** решает задачу нахождения всех простых чисел до заданного целого числа . Простыми числами называются натуральные числа, которые делятся нацело лишь на себя и на единицу. Остальные натуральные числа называются составными. Единственное исключение — единицу — не относят ни к простым, ни к составным. Решение данной задачи находит широкое применение в области криптографии, анализа данных и других прикладных задач.

Существует множество оптимизированных по критерию быстродействия алгоритмов поиска простых чисел: Решето Эратосфена, Решето Сундарама, Решето Аткина, Колёсная факторизация, Сегментированная факторизация с базой 2. Для данного исследования был выбран самый простой алгоритм поиска простых чисел – перебор всех чисел до заданного и проверка их на простоту. В данном алгоритме отсутствуют очевидные оптимизации по критерию быстродействия, например, такие как отбрасывание всех четных чисел после числа «2». Алгоритм имеет временную сложность , где - количество искомых простых чисел.

Обоснованием данного выбора является то, что для проверки быстродействия ПО, разработанного на определенном языке программирования, необходимо выполнить задачу нагрузки вычислительной системы по выполнению определенных математических операций, а не задачу поиска простых чисел. По этой причине разработанный алгоритм не сохраняет найденные числа, кроме последнего. Блок-схема разработанного алгоритма поиска простых чисел методом перебора представлена на рисунке .

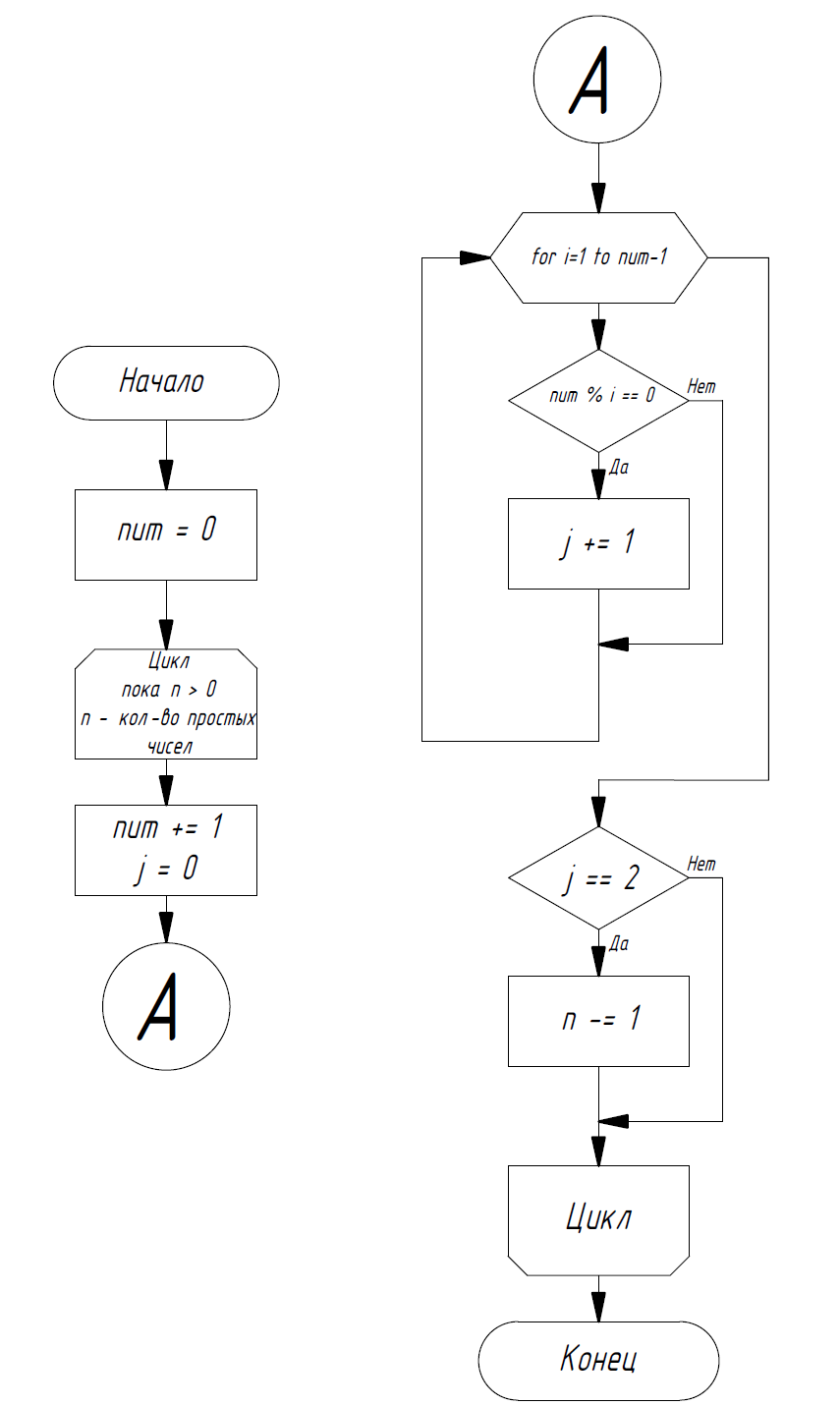


Рисунок – Блок-схема алгоритма поиска простых чисел методом перебора

В результате проведенного анализа были выбраны 3 алгоритма, в которых присутствуют операции, наиболее часто выполняющиеся в эмуляторе встраиваемых систем. Для каждого из алгоритмов представлена блок схема алгоритма и его временная сложность. Данные алгоритмы были реализованы на рассматриваемых выше языках программирования и интегрированы в программное средство «FPLB» для проведения тестирования быстродействия их выполнения.

## 2.3 Анализ результатов исследования

Исследование было проведено для сравнения быстродействия нескольких языков программирования с использованием заданных алгоритмов. Целью исследования было определить наиболее эффективные языки программирования и сделать выводы о предпочтительности их использования в конкретных условиях.

Для этого было разработано специальное тестовое окружение на языке программирования Python, которое позволяло провести сравнение производительности различных языков программирования. В ходе исследования были реализованы одинаковые алгоритмы на разных языках программирования и проанализированы результаты выполнения.

В данной главе представлены результаты исследования, включая сравнительный анализ производительности языков программирования. Важное внимание уделяется аспектам, таким как время выполнения, использование оперативной памяти и другие показатели производительности. Были использованы стандартные алгоритмы и популярные приложения, чтобы максимально охватить области, в которых часто используются языки программирования.

Каждый алгоритм был реализован на разных языках программирования, и производительность каждой реализации была оценена в соответствии с заданными параметрами. Выявленные различия в производительности могут служить основой для принятия решений о выборе языка программирования в конкретных ситуациях и для оптимизации процессов разработки.

Исследование предоставляет ценные результаты для разработчиков и решает вопросы эффективного использования языков программирования. Результаты документируются и предлагаются в этой главе с всей необходимой информацией, анализом и выводами. Более того, в данной главе рассматриваются возможные проблемы и ограничения исследования, что позволяет учесть их в дальнейших исследованиях и разработке программного обеспечения.

Результатом данного исследования является набор столбчатых диаграмм, на вертикальной оси которых отображается время в секундах, а на горизонтальной оси представлены языки программирования. Получившиеся столбчатые диаграммы из программного средства «FPLB» для каждого из исследуемых алгоритмов представлены на рисунках .

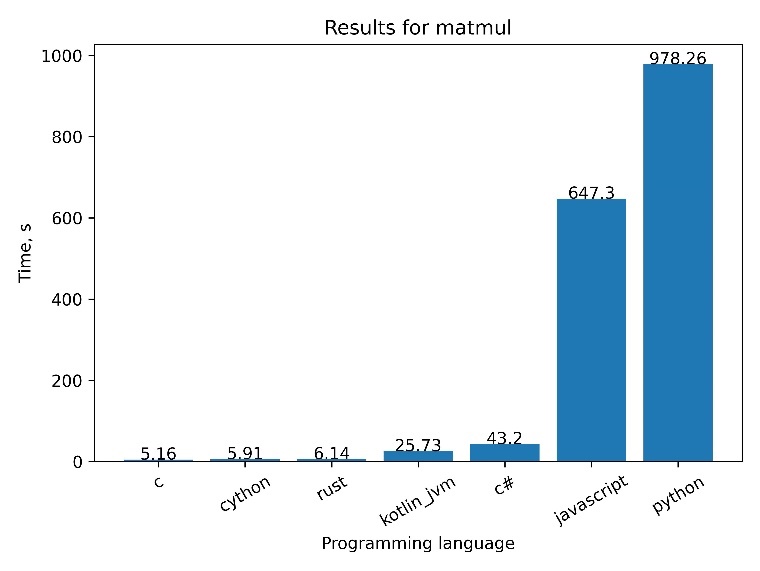


Рисунок – Результаты исследования алгоритма умножения матриц

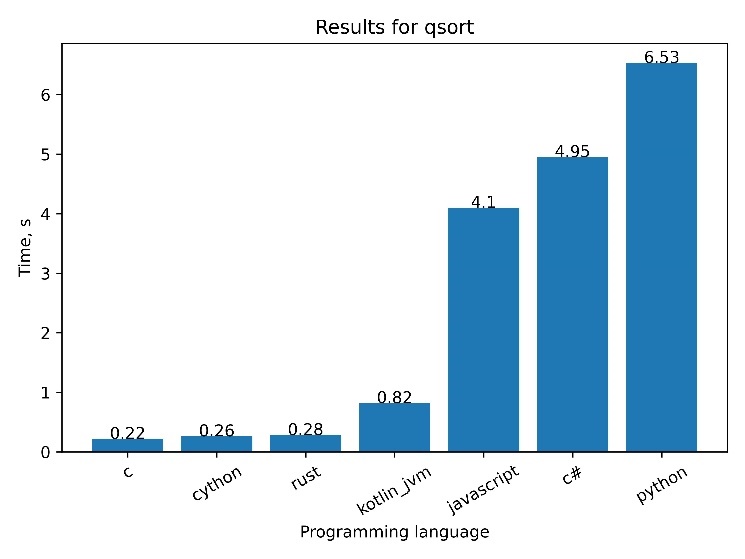


Рисунок – Результаты исследования алгоритма быстрой сортировки

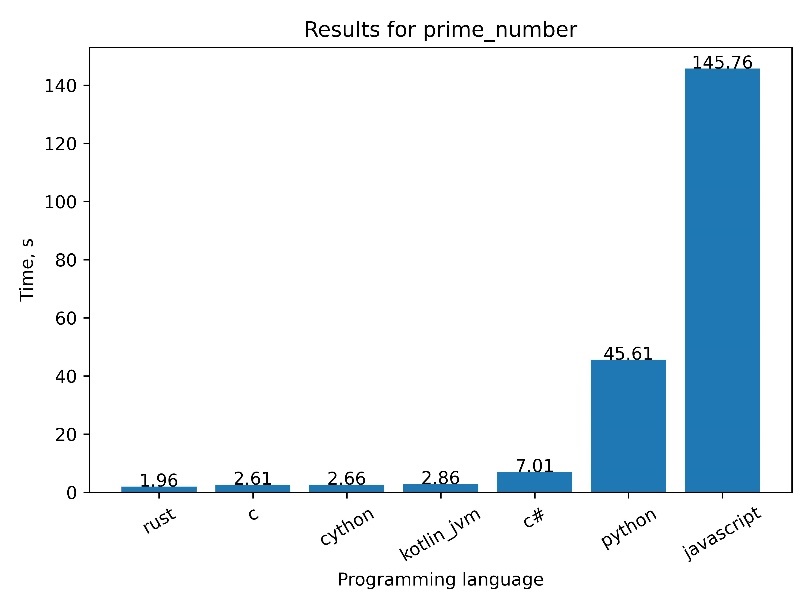


Рисунок – Результаты исследования алгоритма поиска простых чисел методом перебора

Результаты исследования коррелируют с результатами анализа языков программирования и их классификации по критерию преобразования исходного кода в процессорные инструкции. Компилируемые языки C и Rust показывают наилучшее быстродействие в проведенном исследовании. Их результаты примерно равны с учетом случайной погрешности. Язык программирования Cython также показывает высокое быстродействие, так как имеет механизм компиляции, аналогичный тому, что применяется в языке программирования C.

Байт-код языки программирования Kotlin и C# показывают средние значения быстродействия в проведенном исследовании. В исследовании алгоритма поиска простых чисел языку программирования Kotlin даже удалось получить результат, сравнимый по скорости с компилируемыми языками программирования. С другой стороны, в исследовании алгоритма быстрой сортировки язык программирования C# уступил с небольшим отставанием интерпретируемому языку программирования JavaScript. На основании этого можно сделать вывод о том, что байт-код языки программирования в зависимости от выполняемой задачи могут показывать результаты быстродействия в широком диапазоне.

Интерпретируемые языки программирования Python и JavaScript показали худшие результаты быстродействия в данном исследовании. Результаты других языков программирования на порядок выше, чем у интерпретируемых. Стоит отметить, что в исследовании алгоритма поиска простых чисел методом перебора JavaScript показал результат почти в 3 раза медленнее по сравнению с Python. При том, что в остальных исследованиях алгоритмов JavaScript показывал результат лучше Python. Данный результат предположительно связан с использованием в реализованных алгоритмах типа данных «BigInt», который позволяет хранить численные значения больше, чем , в отличие от стандартного типа данных Int. Это связано с тем, что в исследуемых алгоритмах могут получаться числовые значение больше тех, что помещаются в стандартный тип данных Int. Предположительно использование типа данных BigInt требует больших вычислительных ресурсов и, как следствие, уменьшает быстродействие ПО, разработанного на языке программирования JavaScript с применением такого типа данных. Стоит также отметить, что в таком случае применимость языка программирования JavaScript для разработки эмулятора встраиваемых систем является спорным вопросом, так как в современных процессорных архитектурах используются машинные слова размером 64 бита, а также существуют регистры размера 128 бит и более.

Таблица – результаты проведенного исследования быстродействия средств проектирования ПО

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Средство** | **Алгоритм умножения матриц** | **Алгоритм «Быстрая сортировка»** | **Алгоритм поиска простых чисел** | **Среднее значение** |
| **Python** | 978.26 | 6.53 | 45.61 | 343.46 |
| **Cython** | 5.91 | 0.26 | 2.66 | 2.94 |
| **JavaScript** | 647.30 | 4.10 | 145.76 | 265.72 |
| **C#** | 43.20 | 4.95 | 7.01 | 18.38 |
| **Rust** | 6.14 | 0.28 | 1.96 | 2.79 |
| **Kotlin** | 25.73 | 0.82 | 2.86 | 9.80 |
| **C** | 5.16 | 0.22 | 2.61 | 2.66 |

Анализ результатов исследования позволил подтвердить классификацию языков программирования по критерию преобразования исходного кода в процессорные инструкции. Также были выявлены особенности поведения ПО, разработанного на некоторых языках программирования. Из результатов исследования следует, что скорость работы языков программирования может значительно различаться в зависимости от их особенностей и оптимизаций. Однако, следует отметить, что скорость работы не является единственным критерием при выборе языка программирования, учитывать также следует удобство использования, доступность разработчиков и другие факторы.

На основании результатов данного исследования было выявлено, что выбор языка программирования для разработки ПО должен быть осуществлен с учетом конкретных требований и задач, которые решает разрабатываемый программный продукт. Однако, необходимо отметить, что результаты данного исследования являются относительными и могут зависеть от множества факторов, включая конкретную реализацию языка программирования, оптимизацию кода, аппаратное обеспечение и другие условия.

## Выводы

# 3 РАЗРАБОТКА ЭМУЛЯТОРА ЯДРА MIPS

## 3.1 Обзор выбранных средств реализации

В данном разделе будет рассмотрено выбор средств реализации для эмулятора аппаратных платформ с учетом уровня моделирования состояния системы, интерфейса отладки, способа взаимодействия с пользователем и способа конфигурации. В качестве языка средства разработки на основе проведенного исследования был выбран язык программирования Rust.

Rust - это современный системный язык программирования, разработанный в Mozilla Research. Он призван объединить высокую производительность и безопасность в единое целое. Rust обладает инновационными особенностями, позволяющими разработчикам создавать надежные, эффективные и безопасные приложения. В данной статье будет представлен подробный обзор языка программирования Rust с акцентом на его основные особенности и преимущества.

Одной из ключевых особенностей Rust является его система безопасности, которая позволяет предотвратить большинство ошибок, связанных с памятью и разыменованием нулевого указателя. Rust использует механизм владения (ownership), который предотвращает одновременный доступ к данным несколькими владельцами и избегает таких проблем, как сбой в управлении памятью и гонки данных. Благодаря этому, Rust обеспечивает высокий уровень безопасности и защиты при разработке критически важных систем.

Rust обеспечивает высокую совместимость с языками C и C++. Это позволяет эффективно использовать имеющийся код на C/C++ в проектах, написанных на Rust. Rust также предоставляет возможности для взаимодействия с кодом на C/C++ и реализации бинарных интерфейсов при необходимости.

Rust предоставляет удобные средства для разработки многопоточных приложений. Он предлагает понятие безопасного многопоточного программирования без гонок данных (data race). Благодаря особенностям языка, таким как строгость с конкуренцией и проверка времени компиляции, Rust позволяет программисту уверенно работать с параллельными потоками исполнения, минимизируя возможные ошибки при разработке и обеспечивая высокую производительность.

Rust обладает активным сообществом разработчиков и развивающейся экосистемой инструментов. Современные разработчики могут воспользоваться множеством библиотек, фреймворков и инструментов, созданных для Rust. Наличие пакетного менеджера Cargo позволяет удобно управлять зависимостями и развертывать проекты. Большое количество документации, обучающих материалов и руководств делает процесс изучения и применения Rust доступным и эффективным.

Rust обеспечивает высокую производительность благодаря своему низкоуровневому управлению памятью и возможности контроля над расходами ресурсов. Благодаря отсутствию сборки мусора и низкому уровню абстракции, Rust позволяет разработчику точно управлять памятью и приводить код к максимальной эффективности выполнения.

Rust предлагает разработчикам современный, безопасный и высокопроизводительный язык программирования для создания надежных и эффективных приложений. Сочетание его особенностей - безопасности, совместимости, многопоточности, экосистемы разработки и производительности делает его привлекательным инструментом для различных проектов, от маленьких скриптов до крупных системного уровня. Учитывая растущую популярность и поддержку сообщества, Rust становится одним из ключевых языков программирования для современной разработки ПО.

При выборе средства реализации эмулятора аппаратных платформ следует учитывать уровень моделирования состояния системы. Оптимальным выбором является использование высокоуровневых средств, таких как языки программирования высокого уровня, так как они обеспечивают более абстрактное представление системы и позволяют эффективно моделировать и реализовывать сложное аппаратное и программное взаимодействие.

Важным аспектом при выборе средств реализации является наличие удобного и функционального интерфейса отладки. Предпочтительными являются средства с интегрированными возможностями отладки, такие как интегрированные среды разработки (IDE) или отдельные средства отладки, позволяющие контролировать процесс исполнения и анализировать состояние системы во время выполнения.

Для удобства и эффективности работы с эмулятором аппаратных платформ важно обеспечить удобный и интуитивно понятный способ взаимодействия с пользователем. Рекомендуется выбрать средства, которые предоставляют графический интерфейс пользователя (GUI) или мощные средства командной строки для ввода и вывода данных. Также необходимо учитывать возможность предоставления дополнительных функций, таких как поддержка настраиваемых скриптов или плагинов для расширения функциональности.

При выборе средств реализации эмулятора аппаратных платформ следует обратить внимание на способ конфигурации системы. Необходимо выбрать средства, которые обеспечивают гибкую и настраиваемую конфигурацию, позволяющую задавать различные параметры и характеристики системы, такие как процессор, оперативная память, периферийные устройства и т.д. Кроме того, полезными могут быть инструменты для импорта и экспорта конфигурационных файлов, а также возможность сохранения и загрузки конфигурации для повторного использования.

При выборе средств реализации для эмулятора аппаратных платформ необходимо учитывать уровень моделирования состояния системы, интерфейс отладки, способ взаимодействия с пользователем и способ конфигурации. Рекомендуется использовать высокоуровневые языки программирования с интегрированными средствами отладки, обеспечивать удобный GUI или средства командной строки для взаимодействия с пользователем, а также предоставлять настраиваемые параметры конфигурации для гибкости и переиспользования.

## 3.2 Обзор процессорной архитектуры MIPS

Архитектура MIPS была одной из первых RISC-архитектур, получившей признание со стороны промышленности. Она была анонсирована в 1986 году. Первоначально это была полностью 32-битовая архитектура, которая включала 32 регистра общего назначения, 16 регистров плавающей точки и специальную пару регистров для хранения результатов выполнения операций целочисленного умножения и деления. Размер команд составлял 32 бит, в ней поддерживался всего один метод адресации, и пользовательское адресное пространство также определялось 32 битами. Выполнение арифметических операций регламентировалось стандартом IEEE 754.

В компьютерной промышленности широкую популярность приобрели 32-битовые процессоры R2000 и R3000, которые в течение достаточно длительного времени служили основой для построения рабочих станций и серверов. Затем на смену микропроцессорам семейства R3000 пришли новые 64-битовые микропроцессоры R4000 и R4400. (MIPS Technology была певой компанией выпустившей процессоры с 64-битовой архитектурой). Набор команд этих процессоров (спецификация MIPS II) был расширен командами загрузки и записи 64-разрядных чисел с плавающей точкой, командами вычисления квадратного корня с одинарной и двойной точностью, командами условных прерываний, а также атомарными операциями, необходимыми для поддержки мультипроцессорных конфигураций.

Процессоры R2000 и R3000 имели стандартные пятиступенчатые конвейеры команд. В процессорах R4000 и R4400 применяются более длинные конвейеры (иногда их называют суперконвейерами). Количество ступеней в процессорах R4000 и R4400 увеличилось до восьми, что объясняется прежде всего увеличением тактовой частоты и необходимостью распределения логики для обеспечения заданной пропускной способности конвейера. Процессоры R2000 и R3000 имели стандартные пятиступенчатые конвейеры команд. В процессорах R4000 и R4400 применяются более длинные конвейеры (иногда их называют суперконвейерами). Количество ступеней в процессорах R4000 и R4400 увеличилось до восьми, что объясняется прежде всего увеличением тактовой частоты и необходимостью распределения логики для обеспечения заданной пропускной способности конвейера. На рисунке представлена блок схема процессора R10000

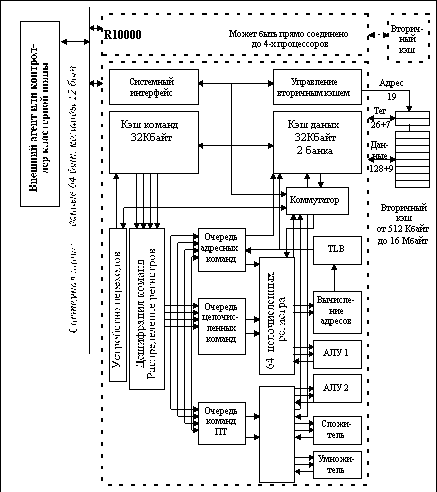


Рисунок – Блок схема процессора R10000

Кэш-память данных первого уровня процессора R10000 имеет емкость 32 Кбайт и организована в виде двух одинаковых банков емкостью по 16 Кбайт, что обеспечивает двухкратное расслоение при выполнении обращений к этой кэш-памяти. Интерфейс кэш-памяти второго уровня процессора R10000 поддерживает 128-битовую магистраль данных, которая может работать с тактовой частотой до 200 МГц, обеспечивая скорость обмена до 3.2 Гбайт/с (для снижения требований к быстродействию микросхем памяти предусмотрена также возможность деления частоты с коэффициентами 1.5, 2, 2.5 и 3). Объем внутренней двухканальной множественно-ассоциативной кэш-памяти команд составляет 32 Кбайт. В процессе ее загрузки команды частично декодируются. При этом к каждой команде добавляются 4 дополнительных бит, которые указывают исполнительное устройство, в котором она будет выполняться. Таким образом, в кэш-памяти команды хранятся в 36-битовом формате. Размер строки кэш-памяти команд составляет 64 байта.

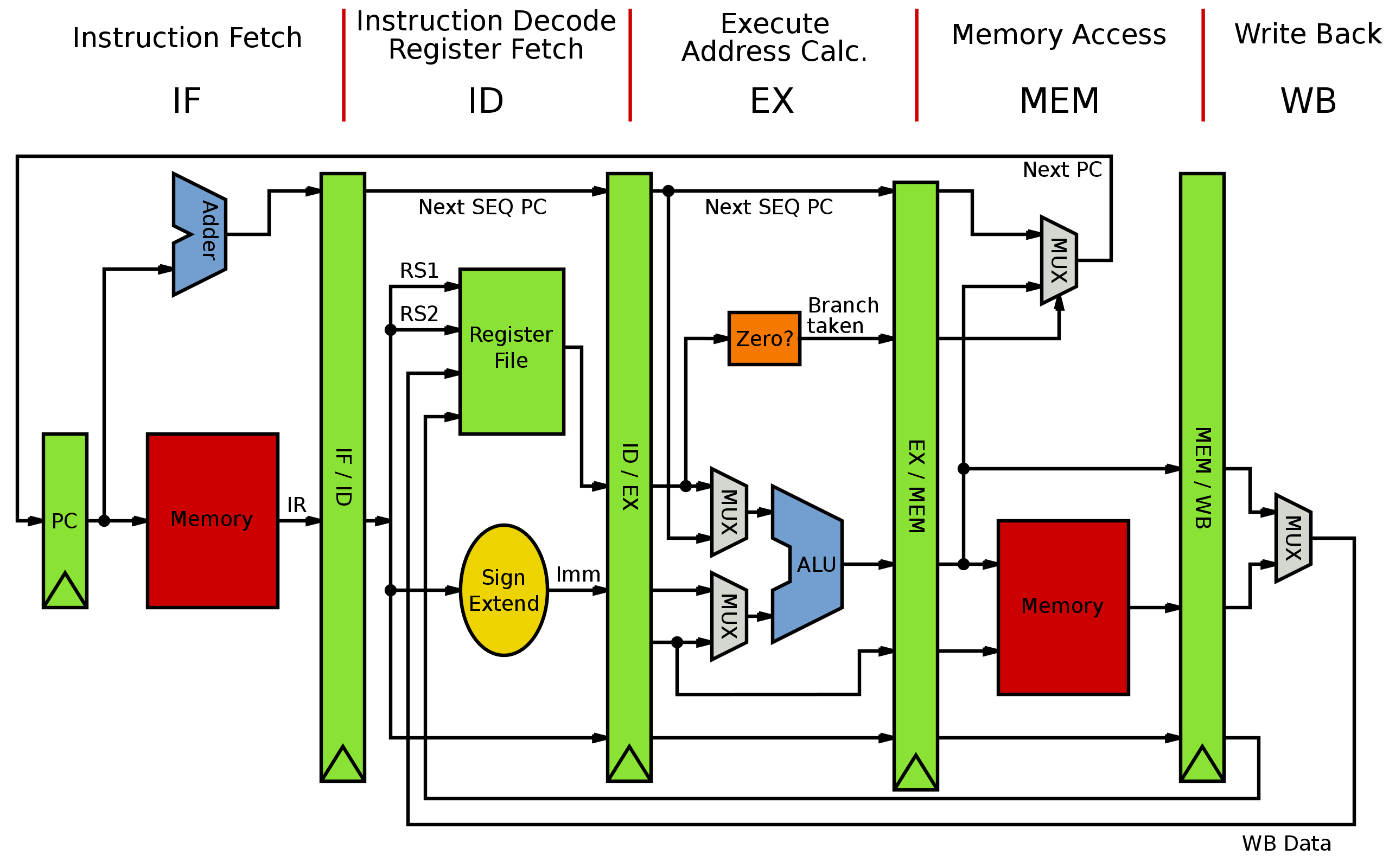


Рисунок - Конвейер MIPS, проходящий пять стадий (получение инструкции, декодирование, исполнение, доступ к памяти и вывод)

В процессоре R10000 имеются пять полностью независимых исполнительных устройств: два целочисленных АЛУ, два основных устройства плавающей точки с двумя вторичными устройствами плавающей точки, которые работают с длинными операциями деления и вычисления квадратного корня, а также устройство загрузки/записи. В микропроцессоре R10000 имеются два целочисленных АЛУ: АЛУ1 и АЛУ2. Время выполнения всех целочисленных операций АЛУ (за исключением операций умножения и деления) и частота повторений составляют один такт.

Устройство загрузки/записи содержит очередь адресов, устройство вычисления адреса, устройство преобразования виртуальных адресов в физические (TLB), стек адресов, буфер записи и кэш-память данных первого уровня. Устройство загрузки/записи выполняет команды загрузки, записи, предварительной выборки, а также команды работы с кэш-памятью.

Выполнение всех команд загрузки и записи начинается с трехтактной последовательности, во время которой осуществляется выдача команды, вычисление виртуального адреса и его преобразование в физический. Преобразование адреса осуществляется во время выполнения команды только однажды. Производится обращение к кэш-памяти данных, и пересылка требуемых данных завершается при наличии данных в кэш-памяти первого уровня.

Одна основных частей MIPS-процессора – это регистры. В стандартном MIPS-процессоре имеется 32 основных регистра и ещё 32 в первом сопроцессоре – модуле, который используется для вычислений с плавающей запятой. Каждый регистр имеет размер 32 бита, соответственно в него целиком помещается одно значение типа int. Для хранения переменной типа long необходимо использовать сразу два регистра. К каждому регистру можно обратиться по его порядковому названию и по его общему названию.

Имеются следующие регистры:

- $zero ($0) – регистр, всегда содержащий значение 0 и доступный только для чтения;

- $at ($1) – временный регистр процессора;

- $v0-$v1 ($2-$3) – для результатов, возвращаемых функциями;

- $a0-$a3 ($4-$7) – для аргументов функций;

- $t0-$t9 ($8-$15, $24-$25) – для временных данных, можно использовать как угодно;

- $s0-$s8 ($16-$23, $30) – для постоянных данных, можно использовать как угодно;

- $k0-$k1 ($26-$27) – зарезервировано для ядра операционной системы;

- $gp ($28) – поинтер для глобальных переменных, практически не используется;

- $sp ($29) – поинтер стека, его значение всегда равно верхнему адресу стека;

- $ra ($31) – бог солнца адрес инструкции, из которой была вызвана функция;

- $f0 – для результатов, возвращаемых функцями, с плавающей запятой;

- $f4, $f6, $f8, $f10, $f16, $f18 – для временных данных с плавающей запятой;

- $f12, $f14 – для параметров функций с плавающей запятой

Существует три основных типа инструкций MIPS-ассемблера:

Команды типа R-type имеют 3 операнда, все операнды - адреса регистров, 2 регистра-источника, 1 регистр - назначение.

Рассмотрим 2 команды этого типа - add и sub.

Команда add (add - сложить) очевидно складывает значения 2х регистров и кладет результат в 3й регистр.

синтаксис на ассемблере:

add rd, rs, rt

rd, rs, rt - адреса (имена) регистров - команда add должна взять значения из регистров rs и rt, вычислить сумму rs+rt и записать результат в регистр rd.

например:

add $s0, $s1, $s2

на практике должно быть выполнено как $s0=$s1+$s2 - вычислить сумму значений, хранящихся в регистрах $s1 и $s2, и записать результат в регистр $s0.

Команда sub (subtract - вычесть) очевидно вычисляет разность значений 2х регистров и кладет результат в 3й регистр.

синтаксис на ассемблере:

sub rd, rs, rt

rd, rs, rt - адреса (имена) регистров - команда sub должна взять значения из регистров rs и rt, вычислить разность rs-rt и записать результат в регистр rd

например:

sub $s0, $s1, $s2

на практике должно быть выполнено как $s0=$s1-$s2 - вычислить разность значений регистров $s1 и $s2 и записать результат в регистр $s0.

Команды типа I-type в отличие от команд типа R-type умеют работать с числовыми константами (immediates - надо понимать что-то типа "незамедлительные" значения, т.е. их не нужно ниоткуда получать), которые встроены прямо в код команды.

Команда lw (load word - загрузить слово) загружает значение из памяти данных в регистр.

синтаксис на ассемблере:

lw rt, imm (rs)

rt - адрес регистра-назначения

imm - константа - адрес загружаемого значения в памяти данных

rs - адрес регистра, содержащего значение сдвига для адреса загрузки

например:

lw $s0, 4 ($0)

сначала вычислит адрес загружаемого значения как значение imm + значение, содержащееся в регистре rs (4+0=4), затем считает значение по вычисленному адресу (4) из памяти данных и запишет его в регистр rt ($s0).

Команда sw (store word - сохранить слово) записывает (сохраняет) значение из регистра в память данных.

синтаксис на ассемблере:

sw rt, imm (rs)

rt - адрес регистра-источника

imm - константа - адрес сохранения значения в памяти данных

rs - адрес регистра, содержащего значение сдвига для адреса сохранения

например:

sw $s0, 4 ($0)

сначала вычислит адрес сохранения значения как значение imm + значение, содержащееся в регистре rs (4+0=4), затем считает значение из регистра rt ($s0) и сохранит его в память данных по вычисленному адресу (4).

Команда addi (add immediate - сложить с константой) складывает значение регистра с константой и записывает результат в регистр.

синтаксис на ассемблере:

addi rt, rs, imm

rt - адрес регистра-назначения

rs - адрес регистра, содержащего 1е складываемое значение

imm - константа - 2е складываемое значение

команда addi должна взять значения из регистра rs, вычислить сумму rs+imm и записать результат в регистр rt.

например:

addi $s0, $s1, 4

на практике должно быть выполнено как $s0=$s1+4 - вычислить сумму значения, хранящегося в регистре $s1 со значением константы 4, и записать результат в регистр $s0.

## 3.3 Разработка архитектуры эмулятора ядра MIPS

В данной главе будет рассмотрено проектирование программной архитектуры разрабатываемого эмулятора архитектуры MIPS, а также описаны составные программные части, включая регистры процессора, инструкции, арифметико-логическое устройство, блок управления памятью, исключения, вычисления с плавающей запятой и вычислительный конвейер. Диаграмма составных частей архитектуры ядра MIPS представлена на рисунке.

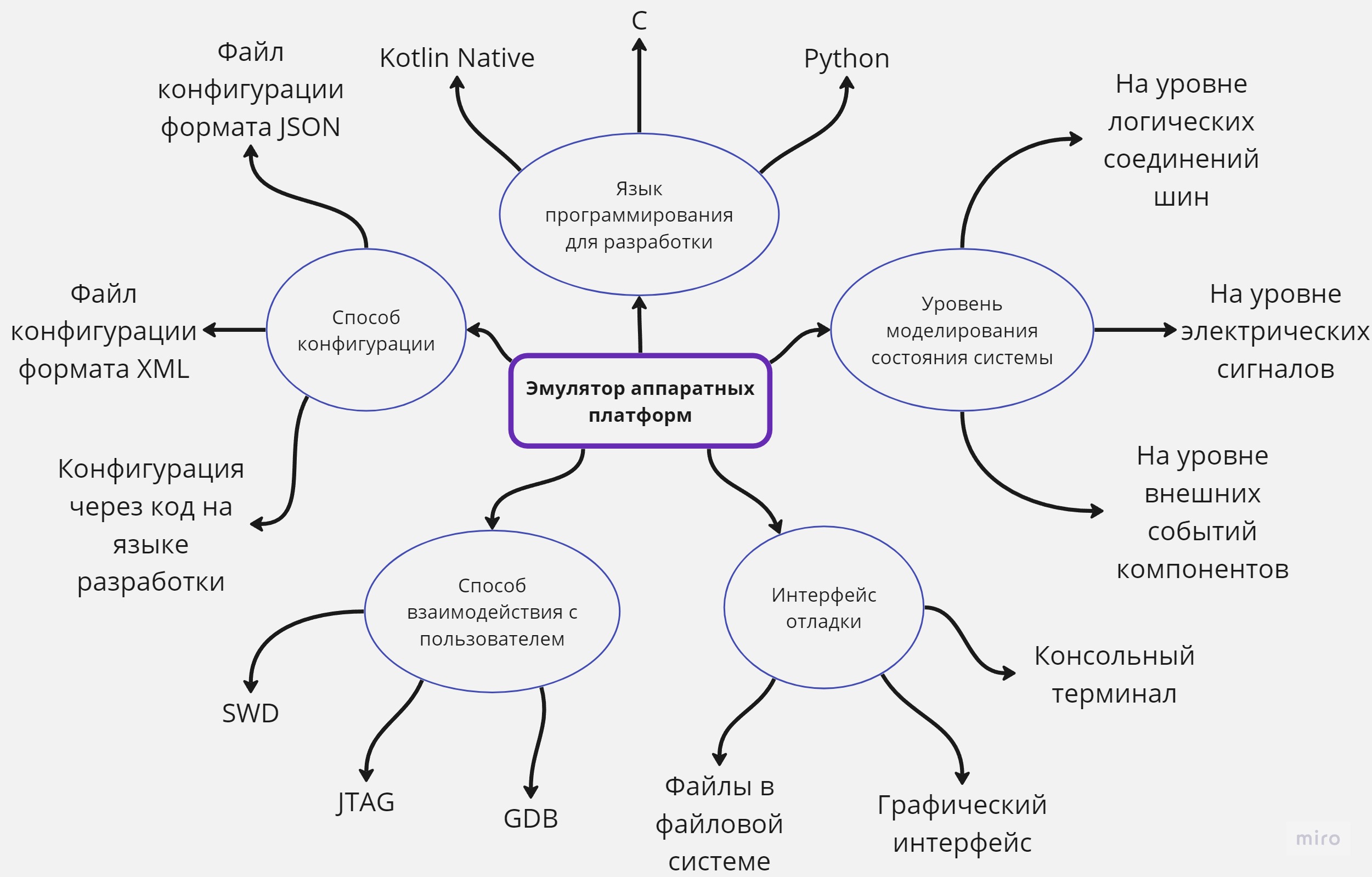


Рисунок – Выбор средств разработки эмулятора

Программная архитектура разрабатываемого эмулятора MIPS должна содержать модель регистров процессора. Регистры должны быть организованы таким образом, чтобы сохранять и обновлять значения регистров в соответствии с работой процессора. Необходимо предусмотреть различные типы регистров, такие как общего назначения, специальные (например, регистр программного счетчика) и регистры сопроцессоров.

Программная архитектура должна включать модель инструкций архитектуры MIPS. Инструкции должны быть организованы таким образом, чтобы моделировать выполнение команд на процессоре MIPS. Необходимо учесть различные типы инструкций, такие как арифметические, логические, переходы и обращения к памяти, и обеспечить возможность декодирования и исполнения каждой инструкции.

Архитектура эмулятора должна включать модель арифметико-логического устройства, которое обеспечивает выполнение арифметических и логических операций. Необходимо предусмотреть различные операции, такие как сложение, вычитание, умножение, деление и логические операции, и обеспечить возможность эмуляции данных операций в соответствии с архитектурой MIPS.

В рамках программной архитектуры необходимо разработать блок управления памятью, который обеспечивает доступ к памяти в соответствии с архитектурой MIPS. Это включает в себя моделирование работы кэш-памяти, виртуальной памяти, а также обращения к различным типам памяти, таким как память данных, кода и стек.

Программная архитектура эмулятора должна поддерживать моделирование исключений, таких как прерывания, системные вызовы и другие обработчики исключительных ситуаций. Важно предусмотреть соответствующие механизмы для обработки и передачи исключений в эмуляторе с целью верного воспроизведения поведения архитектуры MIPS.

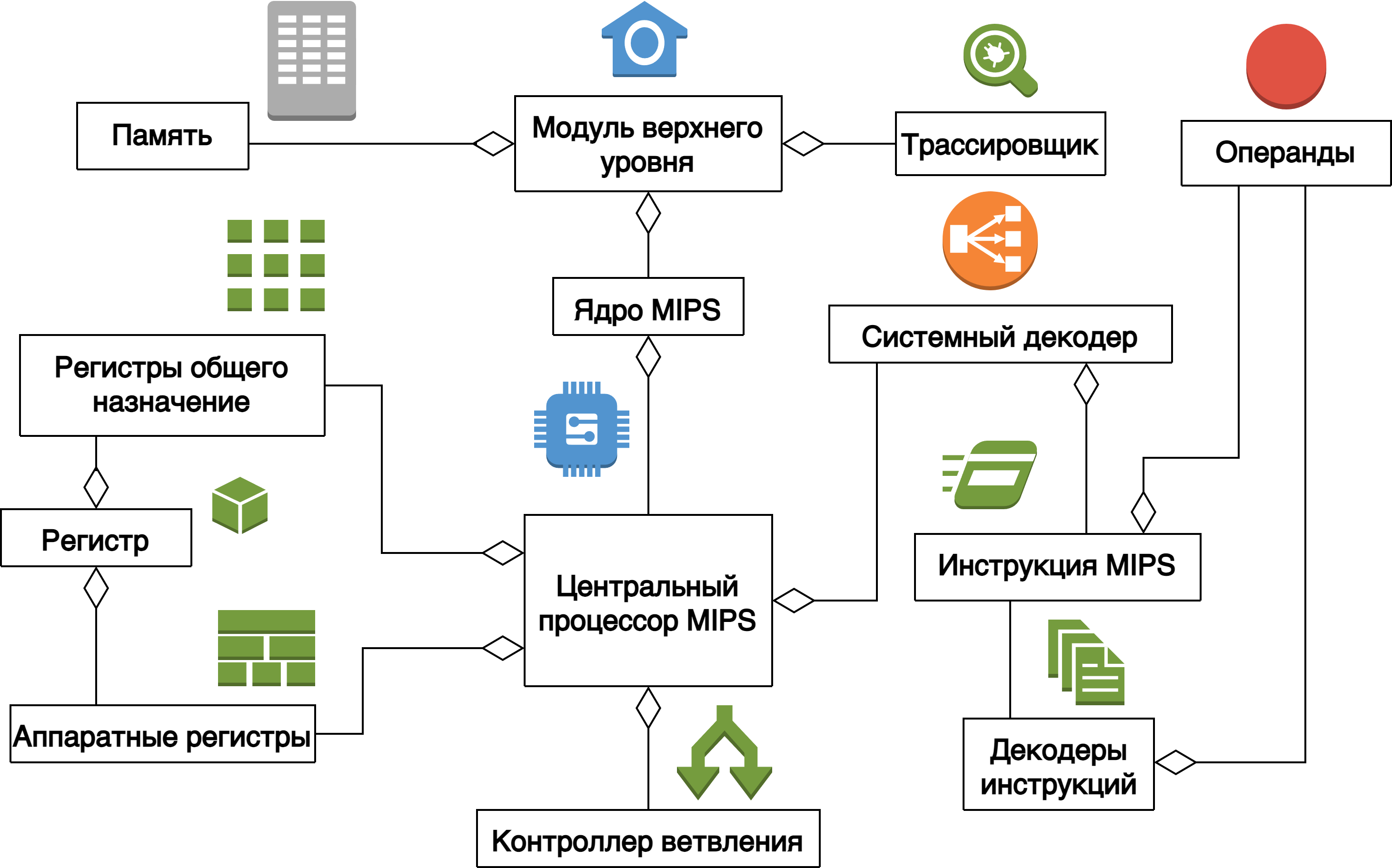


Рисунок – Диаграмма составных частей архитектуры ядра MIPS

При проектировании программной архитектуры следует учесть возможность моделирования операций с плавающей запятой в соответствии с архитектурой MIPS. Необходимо предусмотреть поддержку различных типов данных с плавающей запятой, таких как одинарная и двойная точность, и обеспечить возможность выполнения соответствующих операций, таких как сложение, умножение, деление и сравнение.

При проектировании архитектуры эмулятора можно учесть концепцию вычислительного конвейера для повышения производительности эмулируемой системы. Можно разработать модель вычислительного конвейера, которая разделяет выполнение инструкций на несколько стадий (например, выборка, декодирование, исполнение) и обеспечивает эффективное использование ресурсов процессора.

Для проектирования программной архитектуры разрабатываемого эмулятора архитектуры MIPS необходимо учесть различные составные программные части, включая регистры процессора, инструкции, арифметико-логическое устройство, блок управления памятью, исключения, вычисления с плавающей запятой и вычислительный конвейер. Это позволит создать эффективный и достоверный эмулятор, который точно воспроизводит поведение машины MIPS.

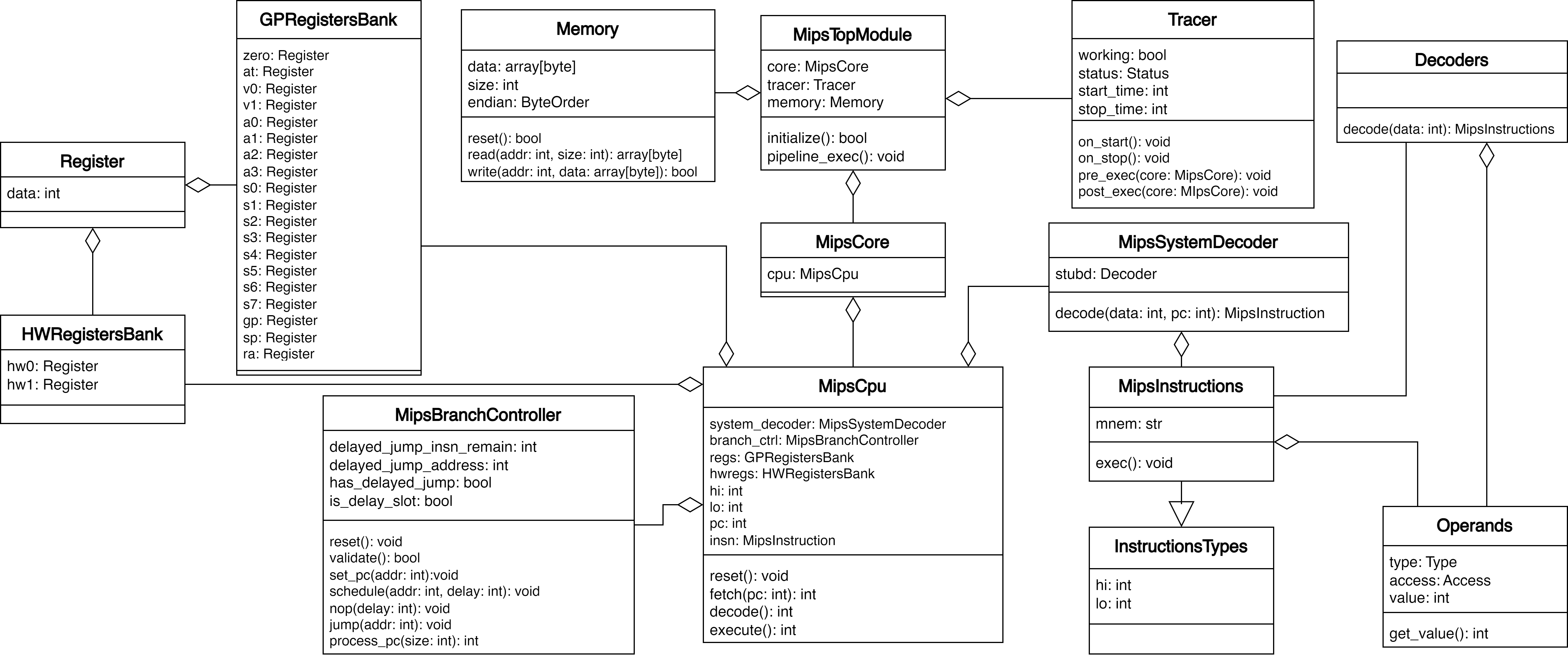


Рисунок – Диаграмма классов разрабатываемого эмулятора ядра MIPS

## 3.4 Разработка эмулятора ядра MIPS

Эмулятор реализован с использованием языка программирования Rust. Это относительно новый язык системного программирования, выпущенный в 2015 году, который обеспечивает множество гарантий безопасности с использованием памяти, выполняется без сборщика мусора и обеспечивает параллельную производительность приложений C/C++.

Rust имеет чрезвычайно выразительную систему типов, имеющую множество параллелей с семейством языков ML. В эмуляторе это использовано для упрощения сложных структур, таких как парсер AST, полностью типобезопасным образом. Эмулятор таким образом гарантированно не имеет уязвимостей, связанных с небезопасностью памяти, обычно связанной с языками нижнего уровня, поскольку он не использует какой-либо небезопасный код, а компилятор Rust может гарантировать безопасность памяти с помощью своей проверки заимствований. Rust также имеет очень развитые возможности обработки ошибок, позволяющие легко распространять исключения вверх за счет использования своего ? оператор. Это позволяет эмулятору кодировать множество различных вариантов ошибок с помощью алгебраических типов данных Rust, как описано в следующем разделе, и гарантирует, что ошибки случайно не пропадут или не распространятся вверх, приводя к завершению процесса.

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Карпунин А.А., Власов А.И. Обработка данных с распределенным реестром в концепции "Индустрия 4.0" // Сборник «Энергосбережение и эффективность в технических системах». Материалы V Международной научно-технической конференции студентов, молодых учёных и специалистов. 2018. С. 120-121.

2. Автоматизация хранения SMD компонентов. Электронный ресурс. Адрес доступа: <https://www.iss-group.ru/electronic-components>. Дата обращения 02.05.2022.

3. Шахнов В.А., Резчикова Е.В. ТРИЗ в техническом университете // Saarbrucken, 2015.

4. Обзор 98 одноплатных компьютеров. Электронный ресурс. Адрес доступа: https://habr.com/ru/post/405023/. Дата обращения 23.04.2022.

5. Автоматизированная система хранения и учета электронных компонентов. Электронный ресурс. Адрес доступа: https://usamodelkina.ru/s/19920-avtomatizirovannaja-sistema-hranenija-i-ucheta-jelektronnyh-komponentov.html. Дата обращения 05.05.2022.

6. Официальный сайт компании Odoo. Электронный ресурс. Адрес доступа: https://www.odoo.com/ru\_RU. Дата обращения 14.03.2022.

7. ГОСТ 2.701-84 «Схемы. Виды и типы. Общие требования к выполнению».

8. ГОСТ 2.702-2011 «Правила выполнения электрических схем».

9. Микроконтроллер STM32F103RDT6. Официальный сайт «STMicroelectronics». Электронный ресурс. Адрес доступа: https://www.st.com/en/microcontrollers-microprocessors/stm32f103rd.html. Дата обращения 24.05.2022.

10. Документация на транзистор BC182LB. Электронный ресурс. Адрес доступа: https://static.chipdip.ru/lib/318/DOC000318843.pdf. Дата обращения 18.05.2022.

11. Кутаев К.С., Рыжов Ф.С. Визуализированная цифровая система хранения и автоматизированной выдачи электронных компонентов на базе собственного сервера с удаленным доступом // KPO-Science. 2021.

12. Кутаев К.С., Латыпова В.А., Кольцова В.С., Кольцов И.Р. Обеспечение безопасности клиент-серверных приложений от инъекций // KPO-Science. 2022.

13. ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения».

14. Кутаев К.С., Латыпова В.А. Модернизация распределения нагрузки на клиент-серверной архитектуре поверх сетевого протокола передачи данных // KPO-Science. 2022.

15. ГОСТ Р 53429-2009 «Платы печатные. Основные параметры конструкции».

16. ГОСТ 10316-78 «Гетинакс и стеклотекстолит фольгированные. Технические условия».

17. ГОСТ 33366.1-2015 «Пластмассы. Условные обозначения и сокращения. Основные полимеры и их специальные характеристики».

18. ГОСТ 21931-76 «Припои оловянно-свинцовые в изделиях. Технические условия».

19. РД 50-690-89 «Методы оценки показателей надежности»

20. Конструкторско-технологические проектирование электронной аппаратуры: Учебник для вузов / К.И. Билибин, А.И, Власов, Л.В. Журавлева и др. Под общ.ред. В.А. Шахнова – М.: Изд-во МГТУ им. Н.Э. Баумана, 2002.

21. ГОСТ Р ИСО 7045-2013 «Винты со скругленной головкой и крестообразным шлицем типа Н или типа Z»

22. Документация на светодиодную ленту WS2812B. Электронный ресурс. Адрес доступа: https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf. Дата обращения 13.05.2022.

23. Документация на RFID-считыватель MFRC522. Электронный ресурс. Адрес доступа: https://www.elecrow.com/download/MFRC522%Datasheet.pdf. Дата обращения 13.05.2022.