|  |  |
| --- | --- |
| 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 являются эмуляторы аппаратных платформ.

Цель работы – разработка эмулятора ядра MIPS с использованием современных средств и методов проектирования ПО.

В процессе работы произведен обзор существующих эмуляторов аппаратных платформ, поддерживающих эмуляцию ядра архитектуры MIPS, классификация и описание современных средств и методов проектирования ПО.

В результате проведённого исследования произведена классификация современных средств и методов проектирования ПО по критериям быстродействия и эффективности разработки. По результатам проведённого исследования был выбран язык программирования, который соответствует обоим критериям. На базе выбранного языка программирования разработан эмулятор ядра MIPS.

**ABSTRACT**

The object of the study is emulators of hardware platforms.

The goal of the work is to develop a MIPS core emulator using modern tools and software design methods.

In the process of work, a review of existing emulators of hardware platforms that support emulation of the MIPS architecture core, a classification and description of modern software design tools and methods were carried out.

As a result of the research, a classification of modern software design tools and methods was made according to the criteria of speed and development efficiency. Based on the results of the study, a programming language was selected that meets both criteria. A MIPS core emulator has been developed based on the selected programming language.

СОДЕРЖАНИЕ

[АННОТАЦИЯ 2](#_Toc154092056)

[СОДЕРЖАНИЕ 3](#_Toc154092057)

[УСЛОВНЫЕ ОБОЗНАЧЕНИЯ, СОКРАЩЕНИЯ И ТЕРМИНЫ 4](#_Toc154092058)

[ВВЕДЕНИЕ 5](#_Toc154092059)

[1 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ 9](#_Toc154092060)

[1.1 Описание предметной области 9](#_Toc154092061)

[1.2 Обзор существующих решений 11](#_Toc154092062)

[1.3 Формулирование цели и постановка задач 16](#_Toc154092063)

[1.4 Выбор алгоритма для проведения исследования 17](#_Toc154092064)

[1.5 Выбор средств разработки ПО для проведения исследования 20](#_Toc154092065)

[1.6 Разработка метода исследования средств разработки ПО 24](#_Toc154092066)

[1.7 Обоснование выбора языка программирования для разработки тестового окружения 24](#_Toc154092067)

[1.8 Результаты проведения исследования 30](#_Toc154092068)

[Выводы 33](#_Toc154092069)

[2 КОНСТРУКТОРСКАЯ ЧАСТЬ 34](#_Toc154092070)

[2.1 Обзор выбранных средств реализации 34](#_Toc154092071)

[2.2 Обзор процессорной архитектуры MIPS 37](#_Toc154092072)

[2.3 Разработка архитектуры эмулятора ядра MIPS 43](#_Toc154092073)

[ЗАКЛЮЧЕНИЕ 46](#_Toc154092074)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 47](#_Toc154092075)

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 аппаратных платформ и реализации в нем 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 эффективности разработки и реализации новых архитектур и устройств. Основное внимание уделено сравнению возможностей современных средств и методов разработки ПО по скорости выполнения алгоритмов. Показаны возможности аналогов эмулятора аппаратных платформ по скорости выполнения инструкций и эффективности реализации новых архитектур и устройств. Основная проблема аналогов заключается в несоответствии одновременно обоим исследуемым критериям. Это следствие использования определенных средств и методов разработки ПО, в частности языков программирования. Предложена стратегия исследования средств и методов проектирования ПО по критериям быстродействия и эффективности разработки новых решений. Предложенный метод 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 критериям быстродействия и эффективности реализации новых решений. 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, в частности популярные языки программирования, а также скорость исполнения инструкций в аналогах разрабатываемого продукта.

**Предметом исследования** является возможность реализации эмулятора, способного соответствовать одновременно критериям быстродействия и эффективности разработки новых решений в нем.

**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 реверс-инжиниринга работы 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 повышение эффективности разработки встраиваемых систем за счет разработки эмулятора, обладающего критериями быстродействия и эффективности реализации новых устройств и архитектур в нем.

**Структура работы**: Работа состоит из введения и трех глав. Первая глава представляет собой исследовательскую часть, включающую в себя анализ существующих решений и аналогов эмулятора аппаратных платформ, разработку тестового окружения для проведения исследования скорости выполнения алгоритмов различными языками программирования и анализ результатов проведенного исследования. Во второй главе представлена конструкторская часть, которая включает в себя анализ и проектирование программного обеспечения эмулятора аппаратных платформ. В третьей главе представлена технологическая часть, включающая в себя разработку программного обеспечения эмулятора и его тестирование, а также исследование работы разработанного эмулятора с существующими решениями и аналогами по скорости исполнения определенных алгоритмов.

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.

​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.

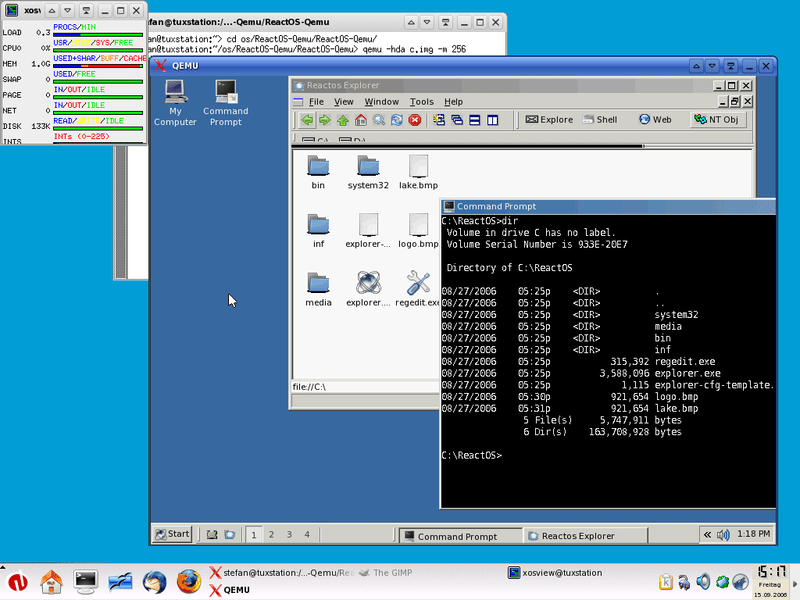


Рисунок 2 - Эмуляция операционной системы ReactOS в эмуляторе Qemu

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.

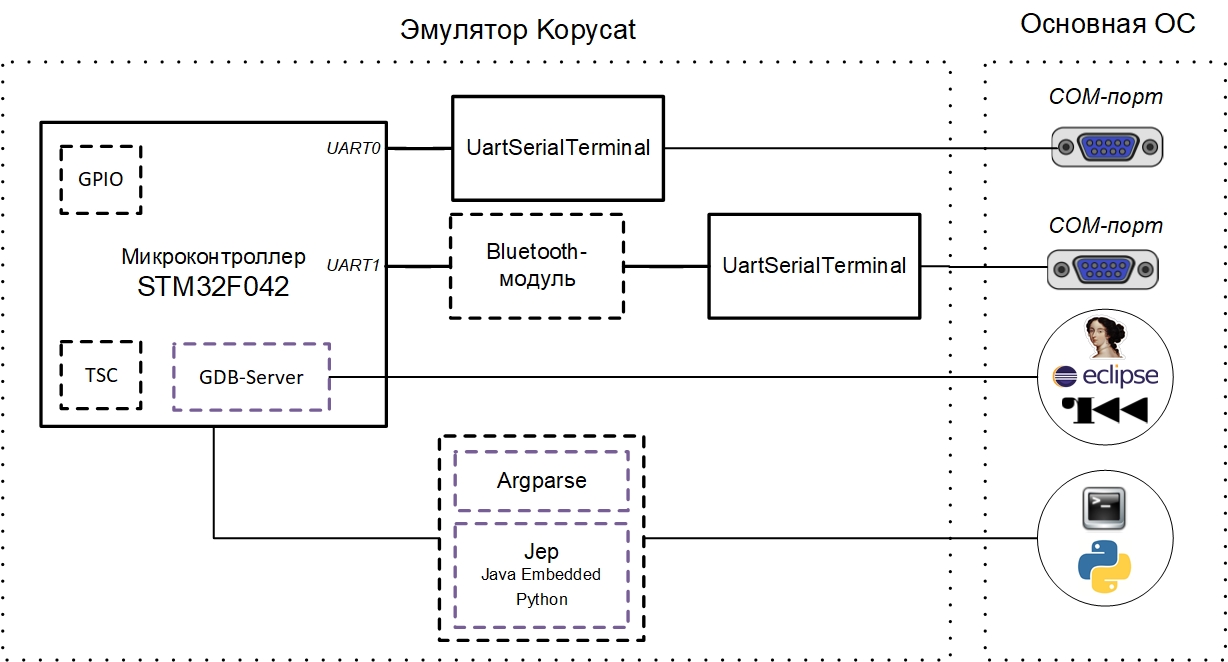


Рисунок – Архитектура эмулятора KOPYCAT для эмуляции МК серии STM32

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.

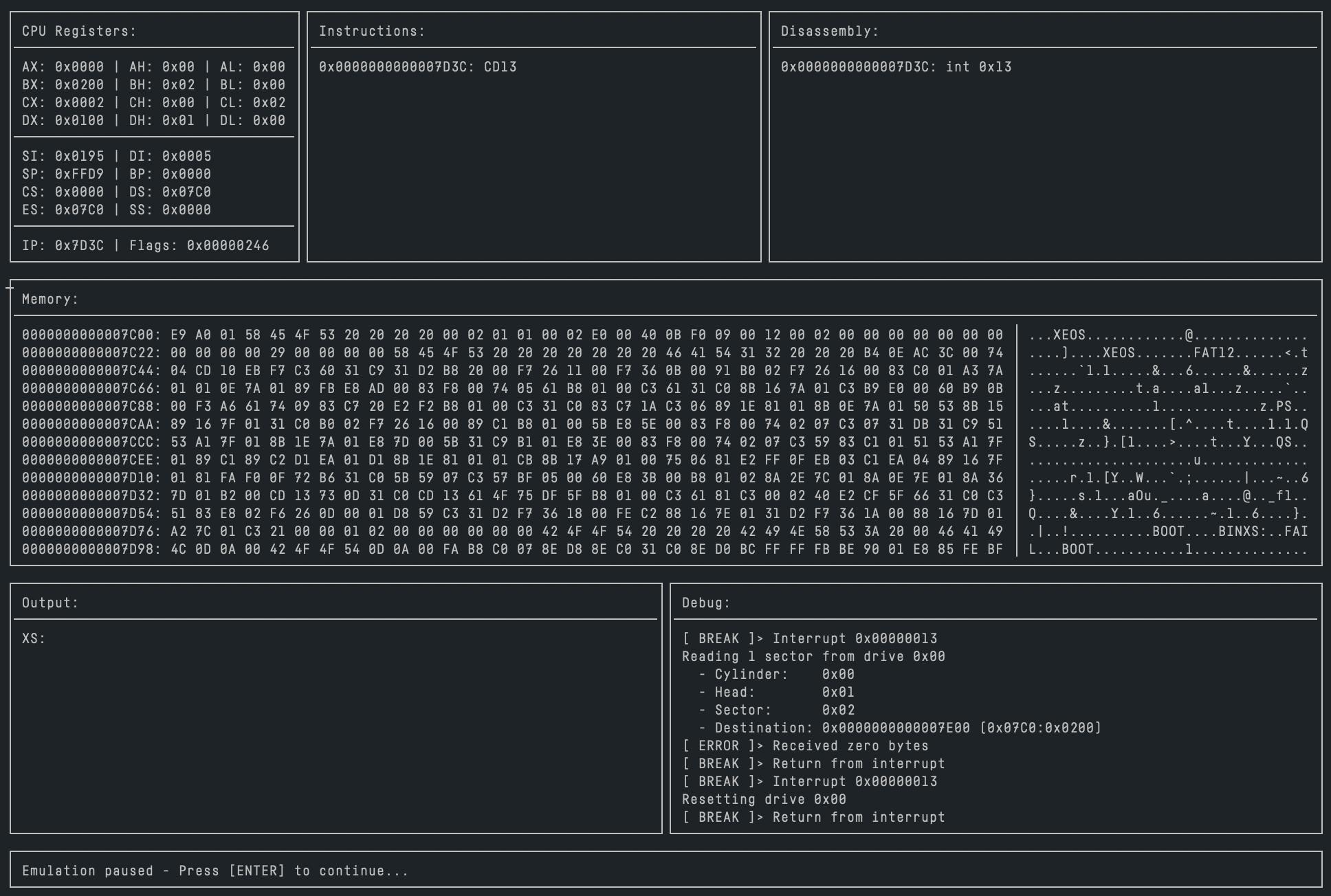


Рисунок – Эмуляция BIOS в Unicorn

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.

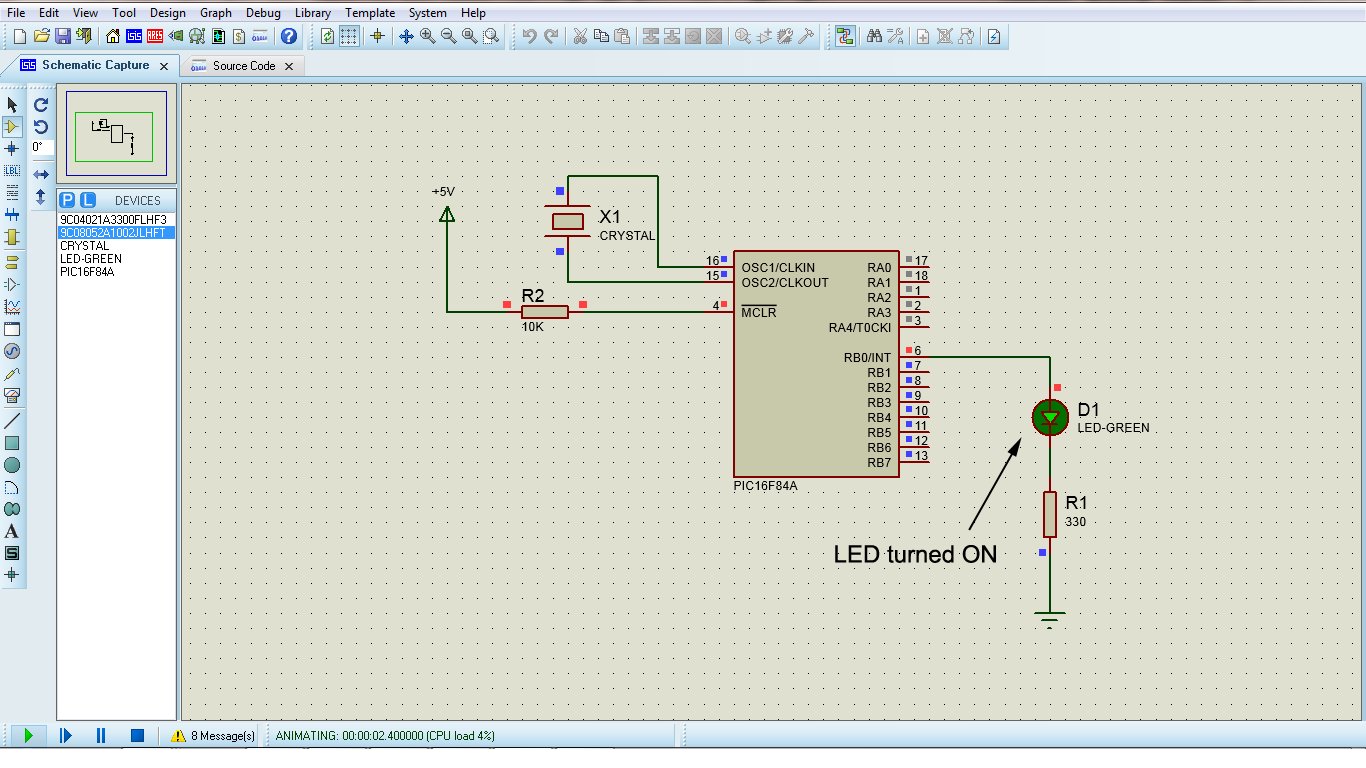


Рисунок - Эмуляция работы микроконтроллера семейства PIC16 в Proteus

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.

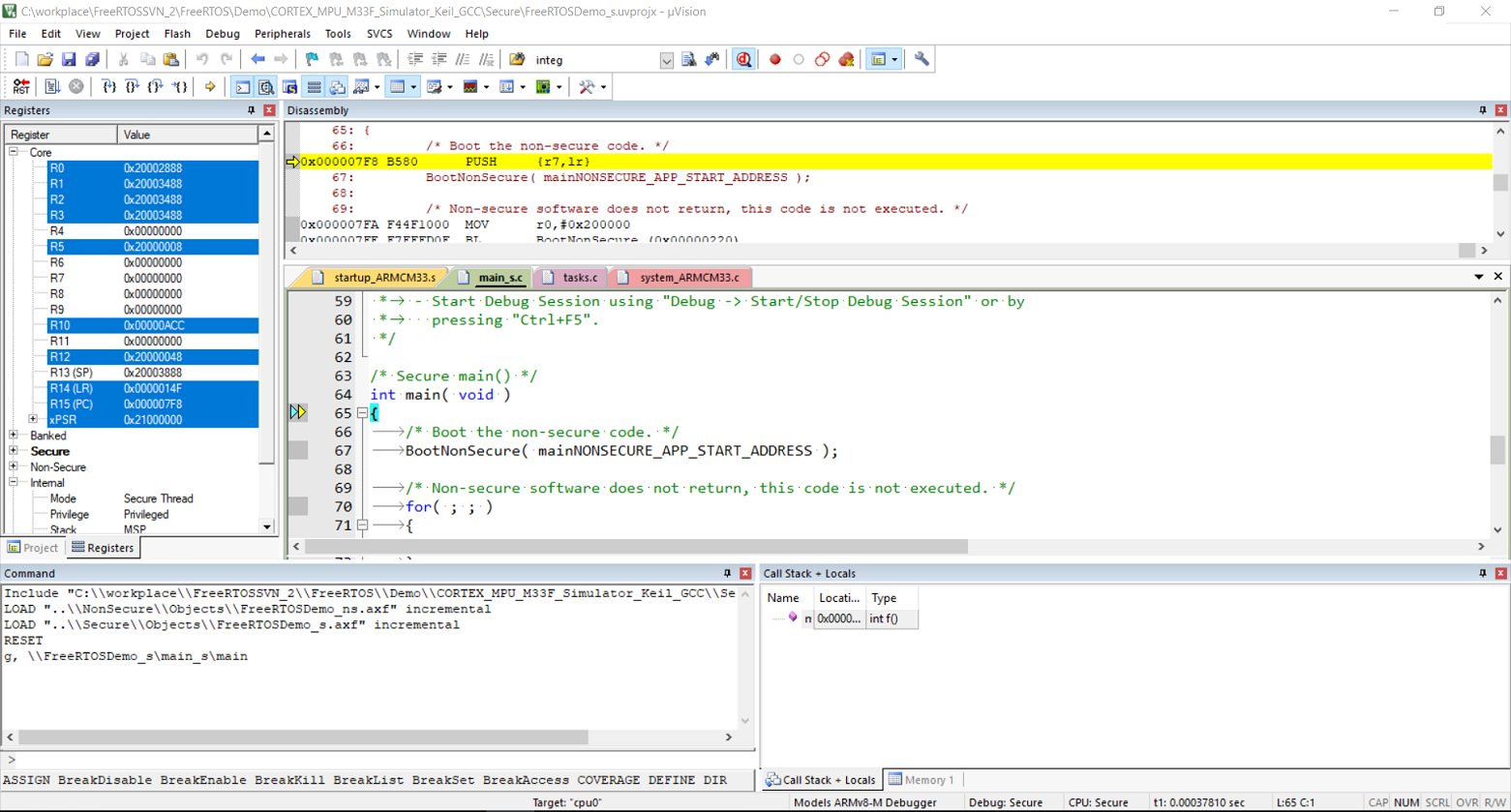


Рисунок - Эмуляция работы ядра ARM Cortex-M33 в IDE Keil uVision

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 |

## 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 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 сравнения скорости выполнения математических операций были выбраны операции умножения матриц и поиска всех простых чисел до заданного целого числа. Умножение матриц выполнено с помощью стандартного алгоритма с небольшим отличием в том, что вторая матрица транспонируется для более эффективного использования кэш-памяти. Блок схема алгоритма умножения матриц представлена на рисунке

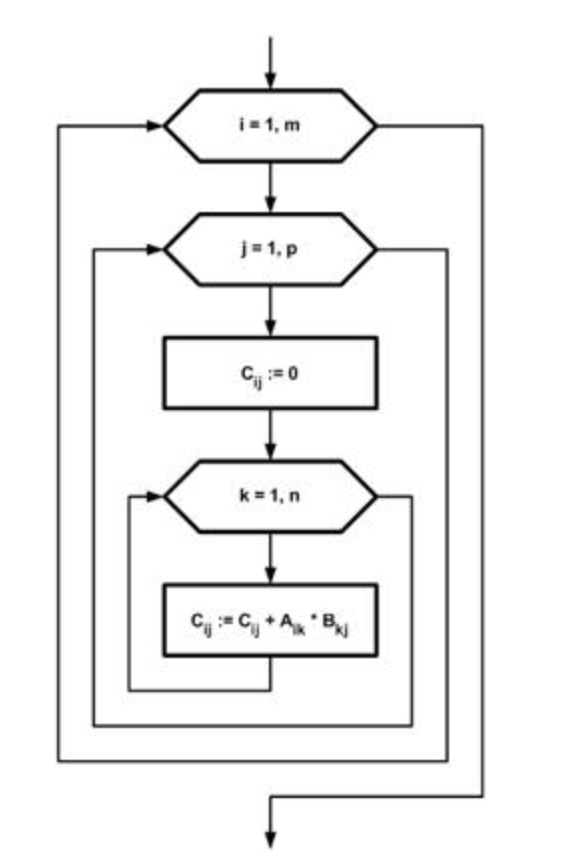


Рисунок – Блок схема алгоритма умножения матриц

Для нахождения простых чисел используется алгоритм, называющийся Решето Аткина. Блок схема алгоритма Решето Аткина представлен на рисунке.

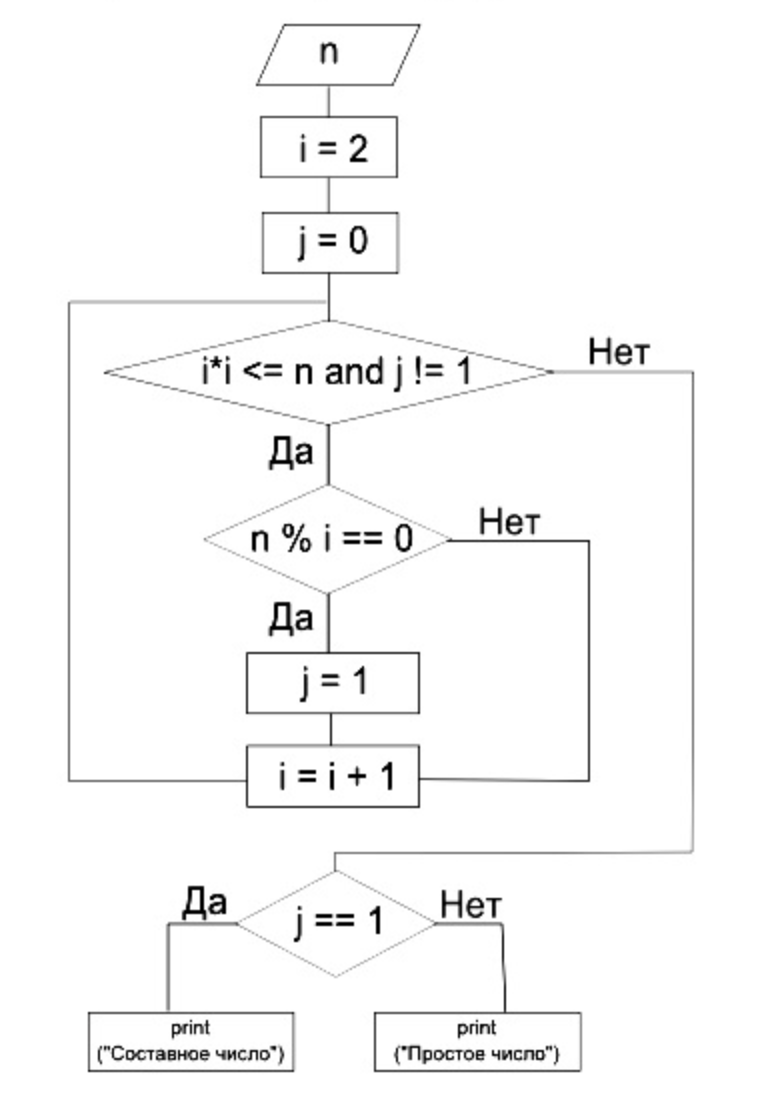


Рисунок – Блок схема алгоритма Решето Аткина

Универсальным алгоритмом, не относящимся к математическим операциям, является быстрая сортировка. Данная сортировка широко используется во многих программах для упорядочения различных данных. Она задействует основные средства языка и легко реализуема, поэтому подходит под исследуемую задачу. Блок схема алгоритма Быстрая сортировка представлен на рисунке.

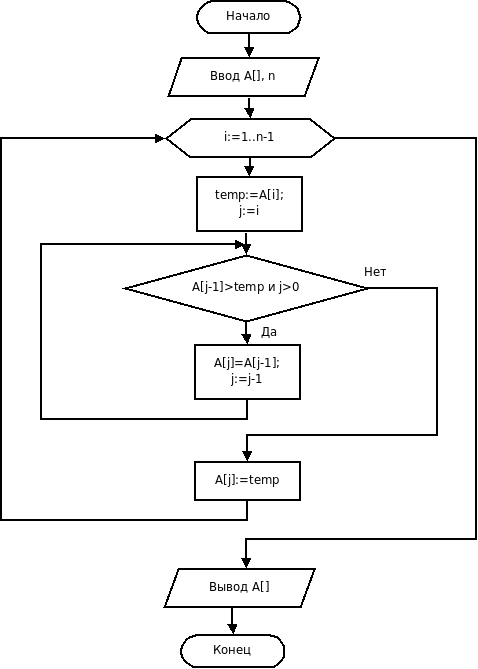


Рисунок – Блок схема алгоритма Быстрая сортировка

## 1.5 Выбор средств разработки ПО для проведения исследования

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.

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.

## 1.6 Разработка метода исследования средств разработки ПО

В данном разделе описан метод проведения исследования средств разработки ПО по критерию быстродействия.

## 1.7 Обоснование выбора языка программирования для разработки тестового окружения

Тестовое окружение представляет собой ПО, в которое можно расширять за счет добавления новых конфигураций и тестов на различных языках программирования.

Основание выбора языка программирования Python для написания тестового окружения для сравнения быстродействия нескольких языков программирования с использованием заданных алгоритмов может быть обосновано следующим образом.

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.

В приведенном листинге текста приведены характеристики системы, на которой будет проводится исследование языков программирования.

|  |
| --- |
| Architecture: x86\_64  CPU op-mode(s): 32-bit, 64-bit  Address sizes: 45 bits physical, 48 bits virtual  Byte Order: Little Endian  CPU(s): 2  On-line CPU(s) list: 0,1  Vendor ID: AuthenticAMD  Model name: AMD Ryzen 5 5600U with Radeon Graphics  CPU family: 25  Model: 80  Thread(s) per core: 1  Core(s) per socket: 1  Socket(s): 2  Stepping: 0  BogoMIPS: 4591.20  Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse ss e2 syscall nx mmxext pdpe1gb rdtscp lm constant\_tsc rep\_good nopl tsc\_reliable nonstop\_tsc cpui d extd\_apicid tsc\_known\_freq pni pclmulqdq ssse3 fma cx16 sse4\_1 sse4\_2 x2apic movbe popcnt aes xsave avx f16c rdrand hypervisor lahf\_lm cr8\_legacy abm sse4a misalignsse 3dnowprefetch osvw t opoext ibpb vmmcall fsgsbase bmi1 avx2 smep bmi2 erms rdseed adx smap clflushopt clwb sha\_ni xs aveopt xsavec xgetbv1 xsaves clzero arat umip vaes vpclmulqdq rdpid overflow\_recov succor fsrm  Virtualization features:  Hypervisor vendor: VMware  Virtualization type: full  Caches (sum of all):  L1d: 64 KiB (2 instances)  L1i: 64 KiB (2 instances)  L2: 1 MiB (2 instances)  L3: 32 MiB (2 instances)  NUMA:  NUMA node(s): 1  NUMA node0 CPU(s): 0,1  Vulnerabilities:  Gather data sampling: Not affected  Itlb multihit: Not affected  L1tf: Not affected  Mds: Not affected  Meltdown: Not affected  Mmio stale data: Not affected  Retbleed: Not affected  Spec rstack overflow: Mitigation; safe RET  Spec store bypass: Vulnerable  Spectre v1: Mitigation; usercopy/swapgs barriers and \_\_user pointer sanitization  Spectre v2: Mitigation; Retpolines, IBPB conditional, STIBP disabled, RSB filling, PBRSB-eIBRS Not affected Srbds: Not affected  Tsx async abort: Not affected  user  description: Computer  product: VMware Virtual Platform  vendor: VMware, Inc.  version: None  serial: VMware-56 4d ae 21 89 72 0c f0-5f 94 52 ae 50 42 5f 3b  width: 64 bits  capabilities: smbios-2.7 dmi-2.7 smp vsyscall32  configuration: administrator\_password=enabled boot=normal frontpanel\_password=unknown keyboard\_password=unknown power-on\_password=disabled uuid=21ae4d56-7289-f00c-5f94-52ae50425f3b  \*-core  description: Motherboard  product: 440BX Desktop Reference Platform  vendor: Intel Corporation  physical id: 0  version: None  serial: None  \*-firmware  description: BIOS  vendor: Phoenix Technologies LTD  physical id: 0  version: 6.00  date: 11/12/2020  size: 86KiB  capabilities: isa pci pcmcia pnp apm upgrade shadowing escd cdboot bootselect edd int5printscreen int9keyboard int14serial int17printer int10video acpi smartbattery biosbootspecification netboot  \*-cpu:0  description: CPU  product: AMD Ryzen 5 5600U with Radeon Graphics  vendor: Advanced Micro Devices [AMD]  physical id: 1  bus info: cpu@0  version: 25.80.0  slot: CPU #000  size: 2300MHz  capacity: 4230MHz  width: 64 bits  capabilities: lm fpu fpu\_exception wp vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx mmxext pdpe1gb rdtscp x86-64 constant\_tsc rep\_good nopl tsc\_reliable nonstop\_tsc cpuid extd\_apicid tsc\_known\_freq pni pclmulqdq ssse3 fma cx16 sse4\_1 sse4\_2 x2apic movbe popcnt aes xsave avx f16c rdrand hypervisor lahf\_lm cr8\_legacy abm sse4a misalignsse 3dnowprefetch osvw topoext ibpb vmmcall fsgsbase bmi1 avx2 smep bmi2 erms rdseed adx smap clflushopt clwb sha\_ni xsaveopt xsavec xgetbv1 xsaves clzero arat umip vaes vpclmulqdq rdpid overflow\_recov succor fsrm  configuration: cores=1 enabledcores=1 microcode=4294967295  \*-cache:0  description: L1 cache  physical id: 0  slot: L1  size: 16KiB  capacity: 16KiB  capabilities: asynchronous internal write-back  configuration: level=1  \*-cache:1  description: L1 cache  physical id: 1  slot: L1  size: 16KiB  capacity: 16KiB  capabilities: asynchronous internal write-back  configuration: level=1  \*-cache:2  description: L2 cache  physical id: 2  slot: L2  size: 512KiB  capacity: 24MiB  capabilities: burst external write-back  configuration: level=2 |

## 1.8 Результаты проведения исследования

Научное 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.

На рисунке представлены результаты выполнения исследования сравнения быстродействия языков программирования на алгоритме умножения матриц.

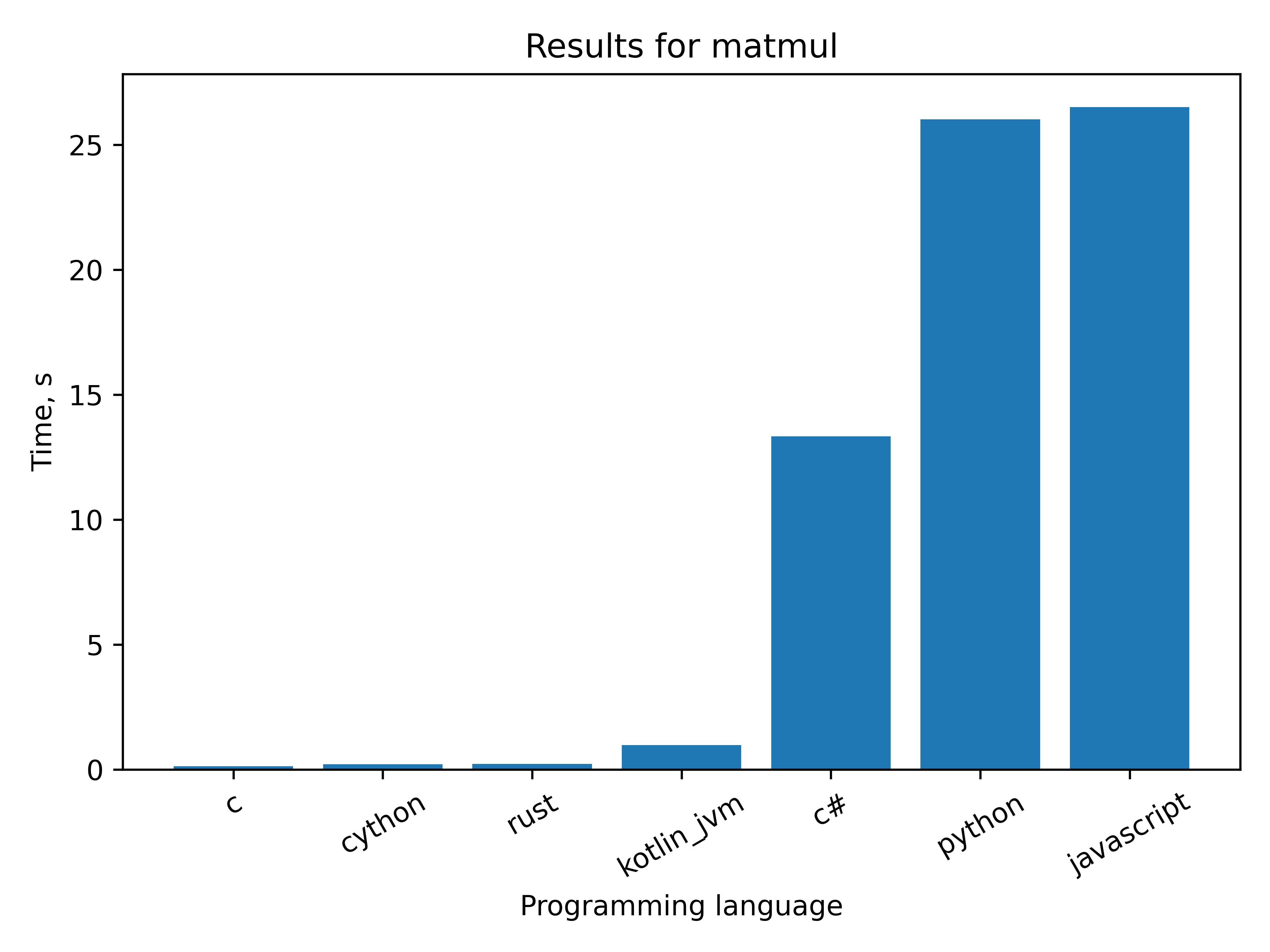


Рисунок – Сравнение результатов выполнения алгоритма умножения матриц

На рисунке представлены результаты выполнения исследования сравне-ния быстродействия языков программирования на алгоритме умножения «Быстрая сортировка».

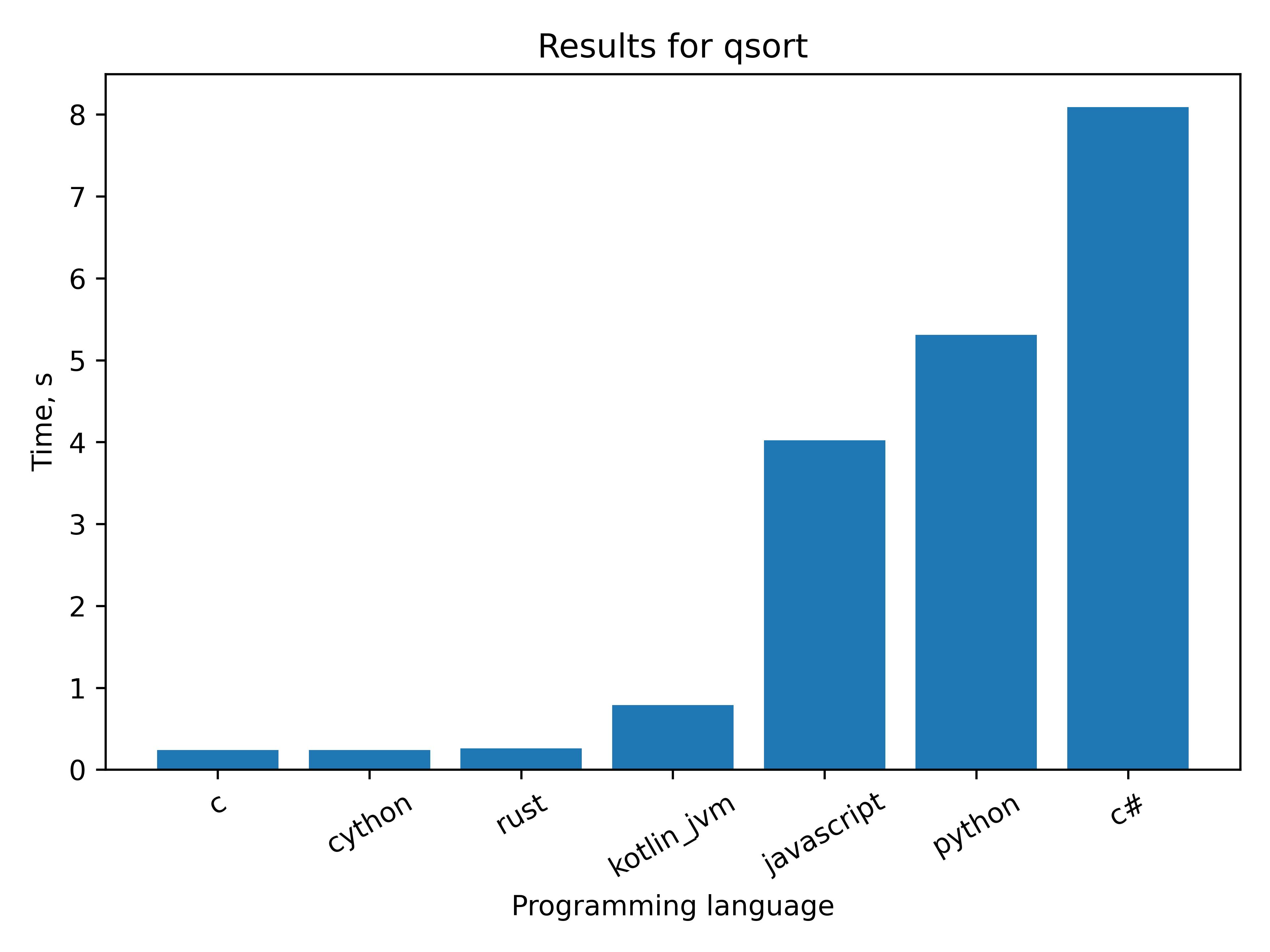


Рисунок – Сравнение результатов выполнения алгоритма «Быстрая сортировка»

## Выводы

# 2 КОНСТРУКТОРСКАЯ ЧАСТЬ

## 2.1 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 оснвое 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.

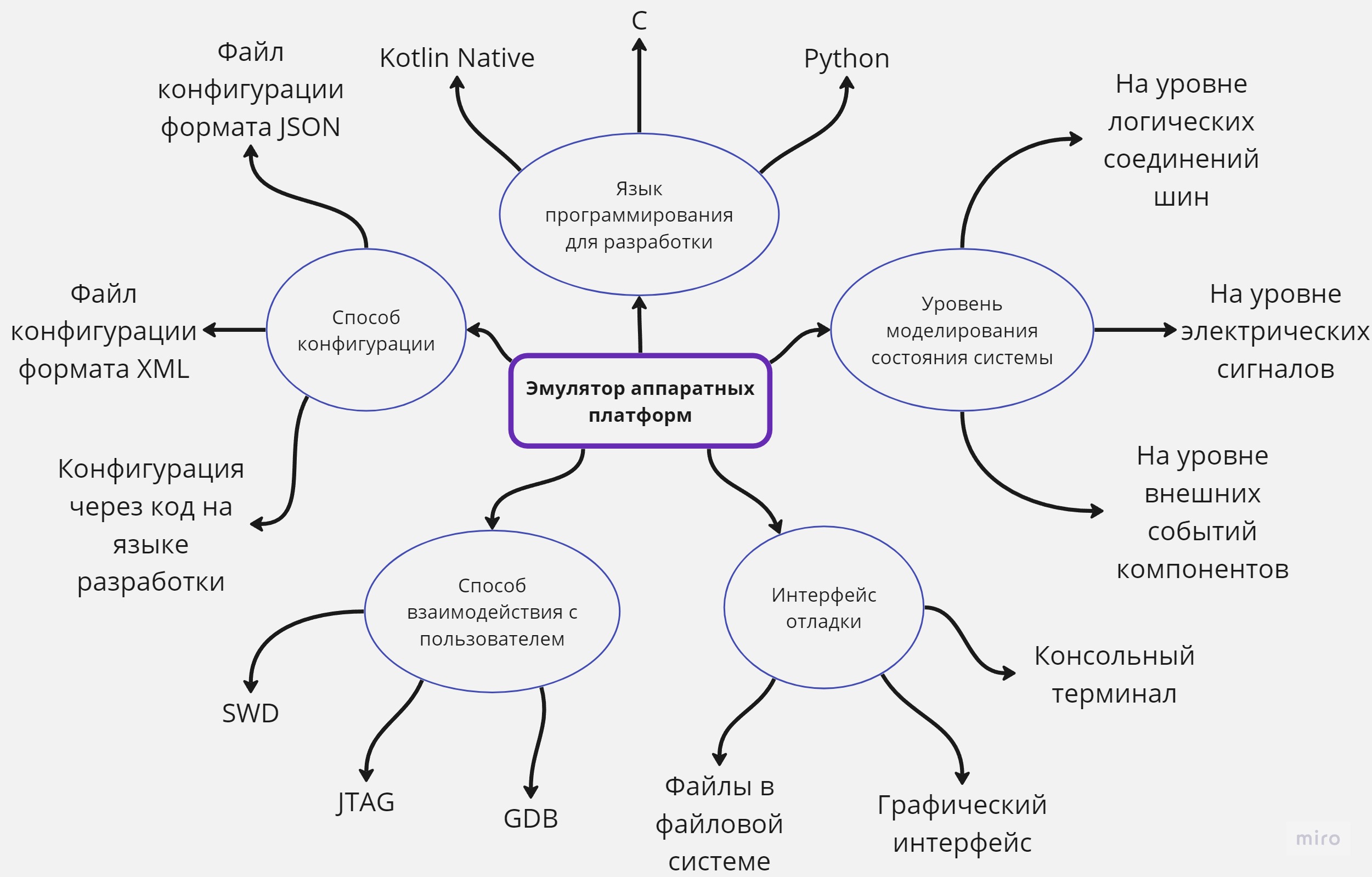


Рисунок – Выбор средств разработки эмулятора

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.

## 2.2 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

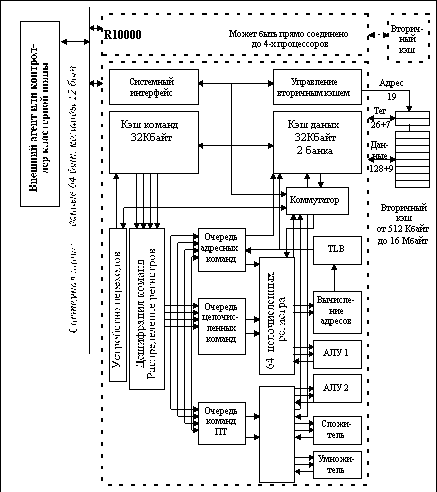


Рисунок – Блок схема процессора R10000

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.

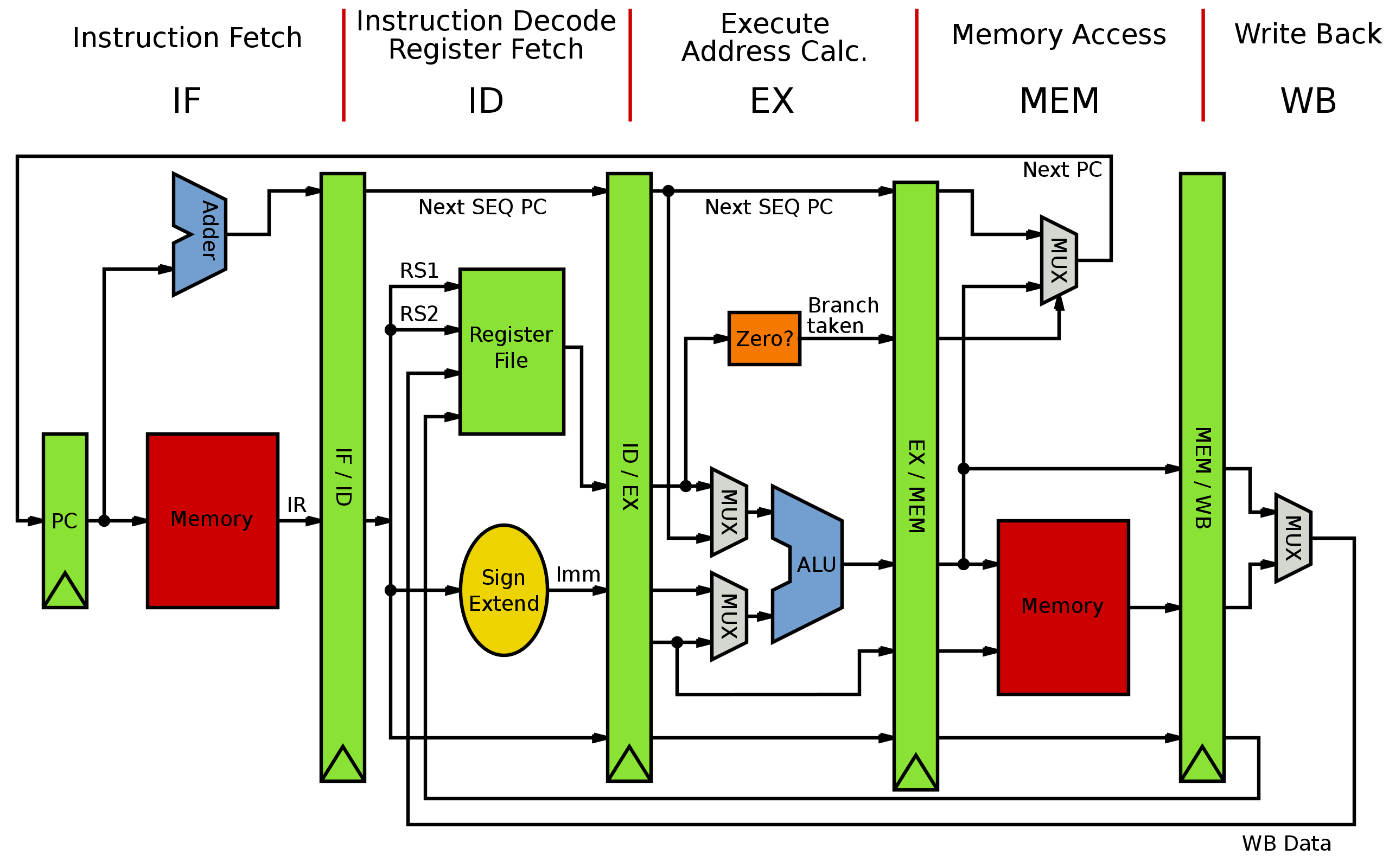


Рисунок - Конвейер MIPS, проходящий пять стадий (получение инструкции, декодирование, исполнение, доступ к памяти и вывод)

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.

## 2.3 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.

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.

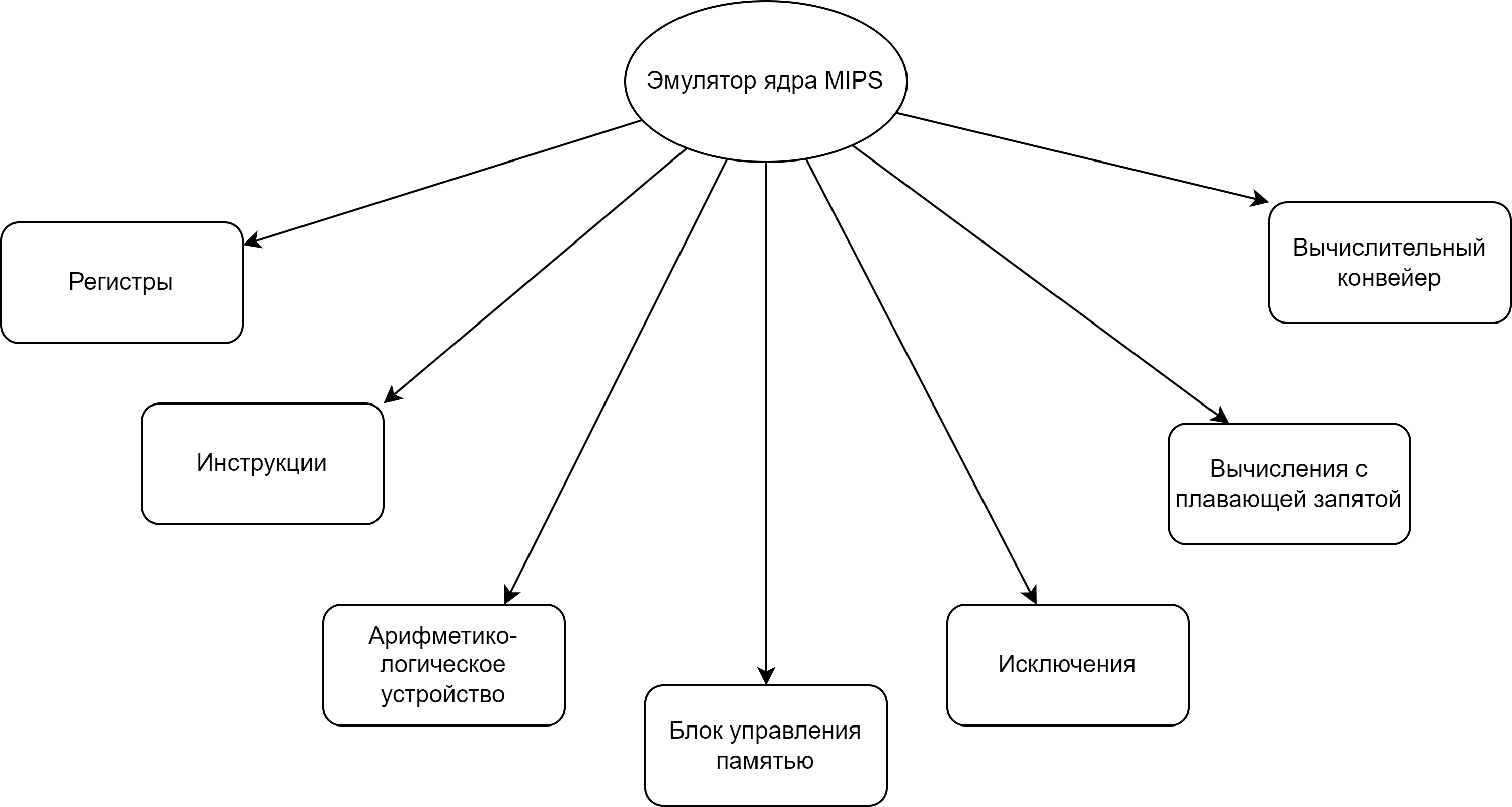


Рисунок – Диаграмма составных частей архитектуры ядра MIPS

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.

# 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.