Шаимов Никита Денисович

Степанов Никита Андреевич

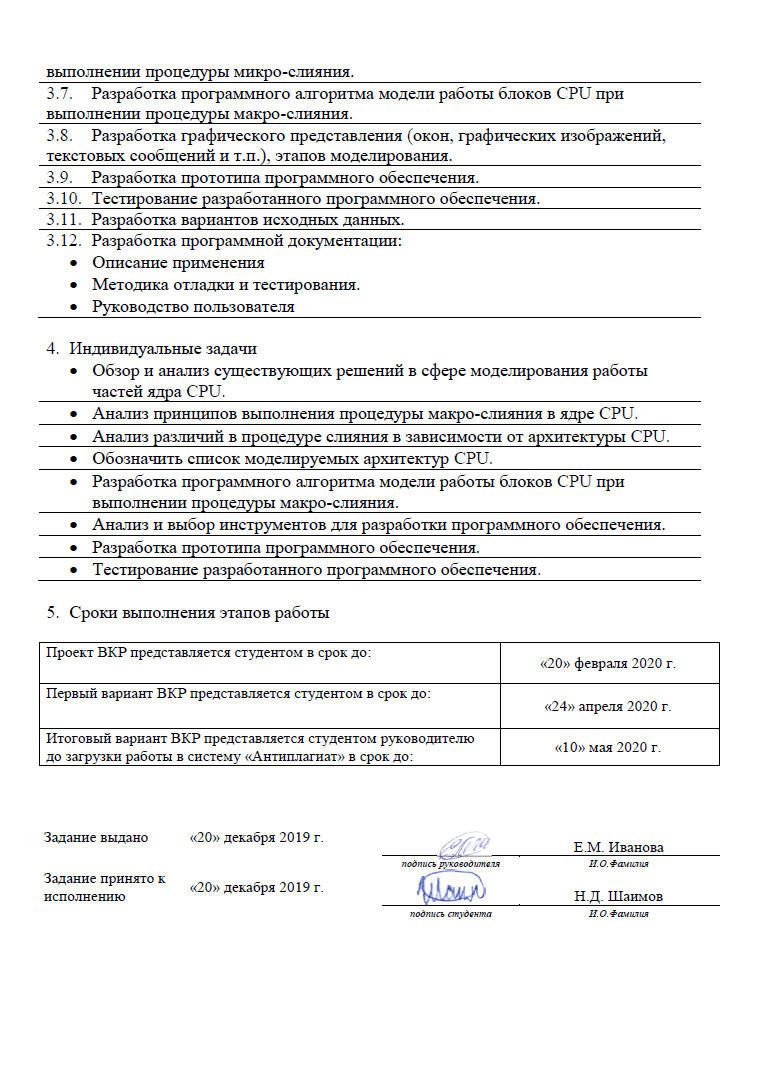
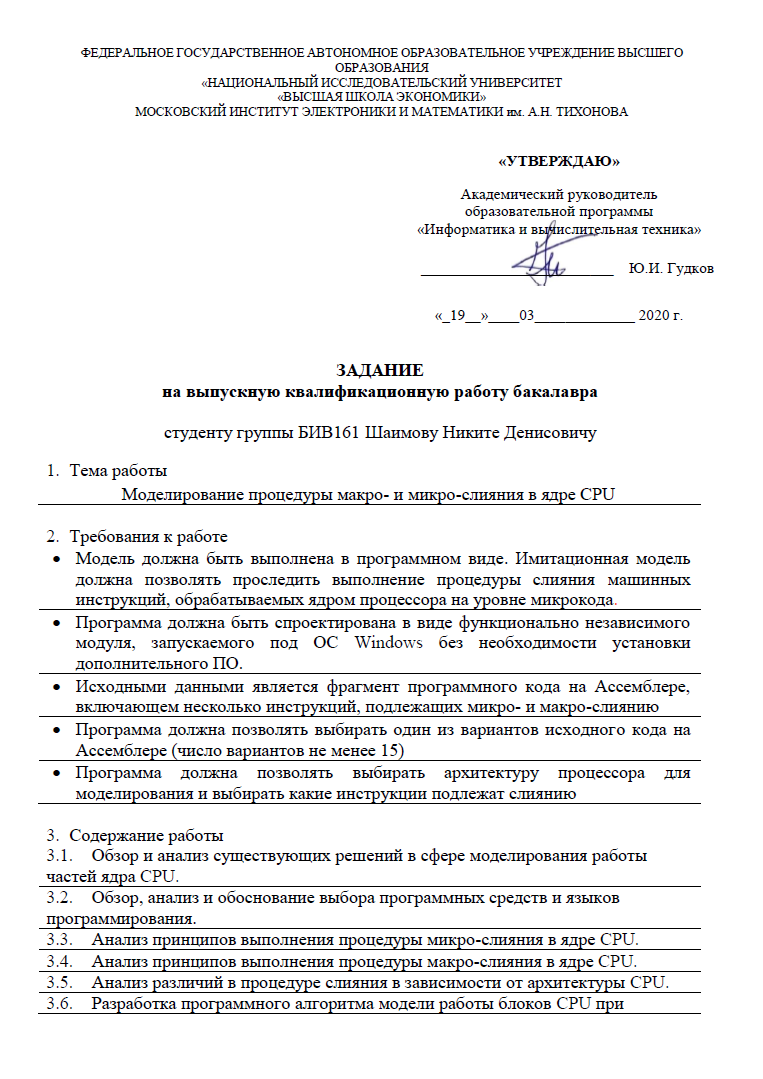
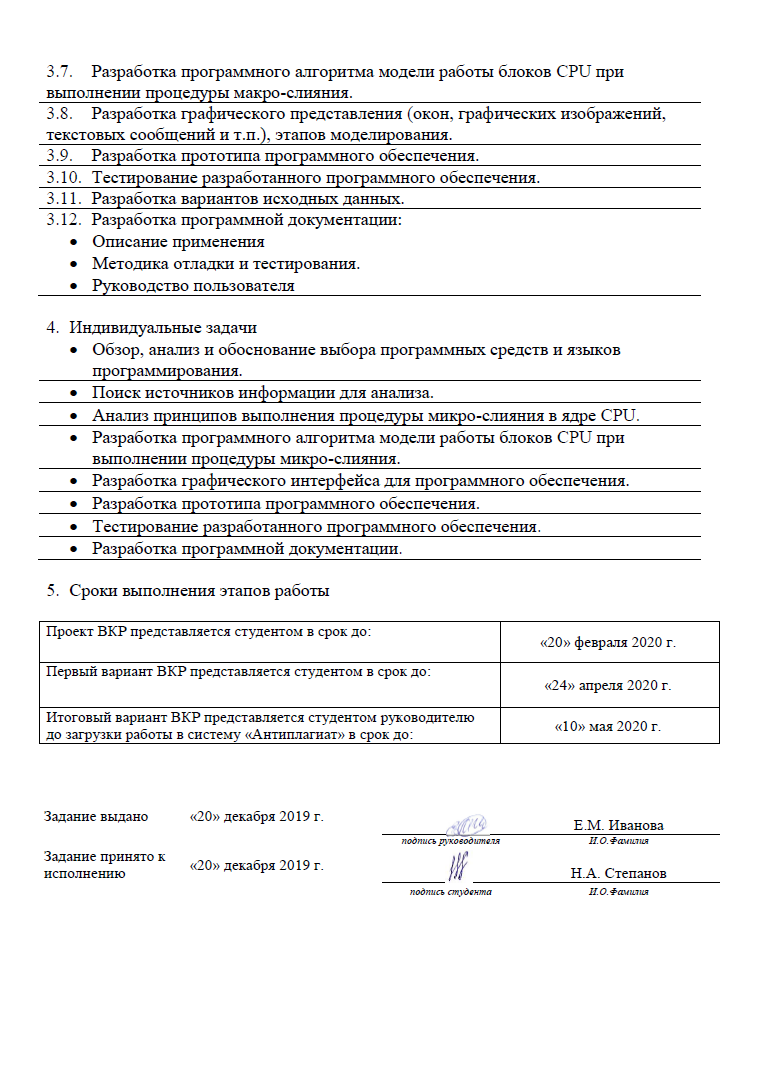
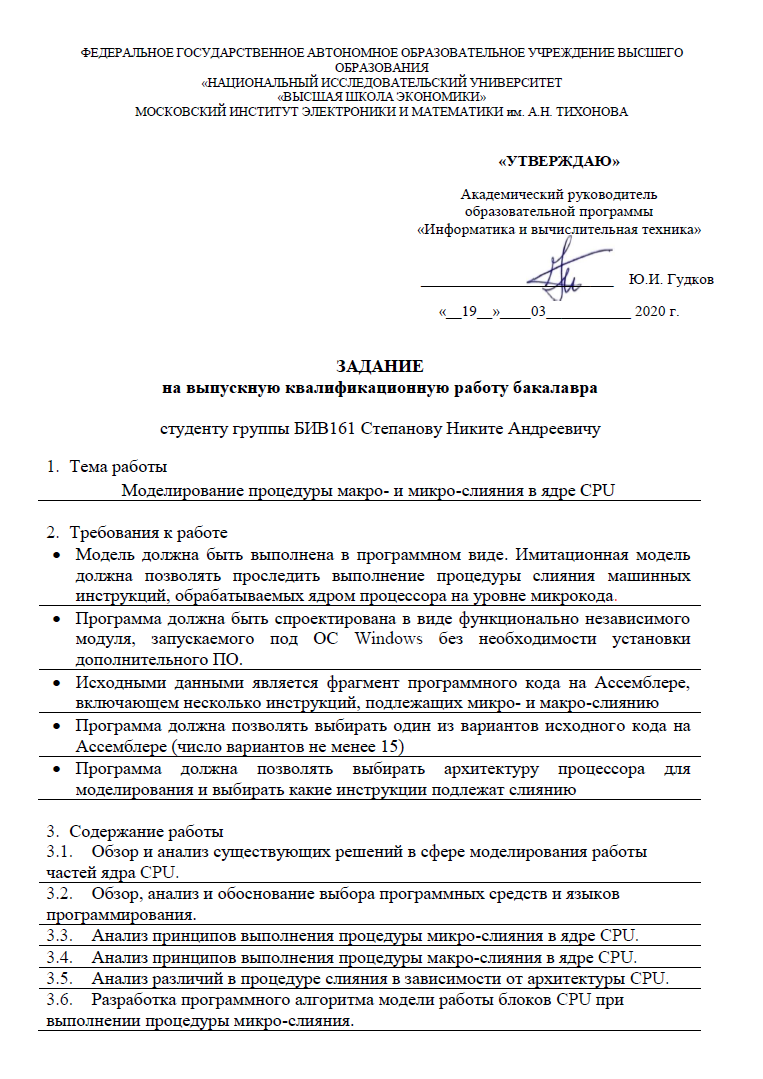
**МОДЕЛИРОВАНИЕ ПРОЦЕДУРЫ   
МАКРО- И МИКРО- СЛИЯНИЯ В ЯДРЕ CPU**

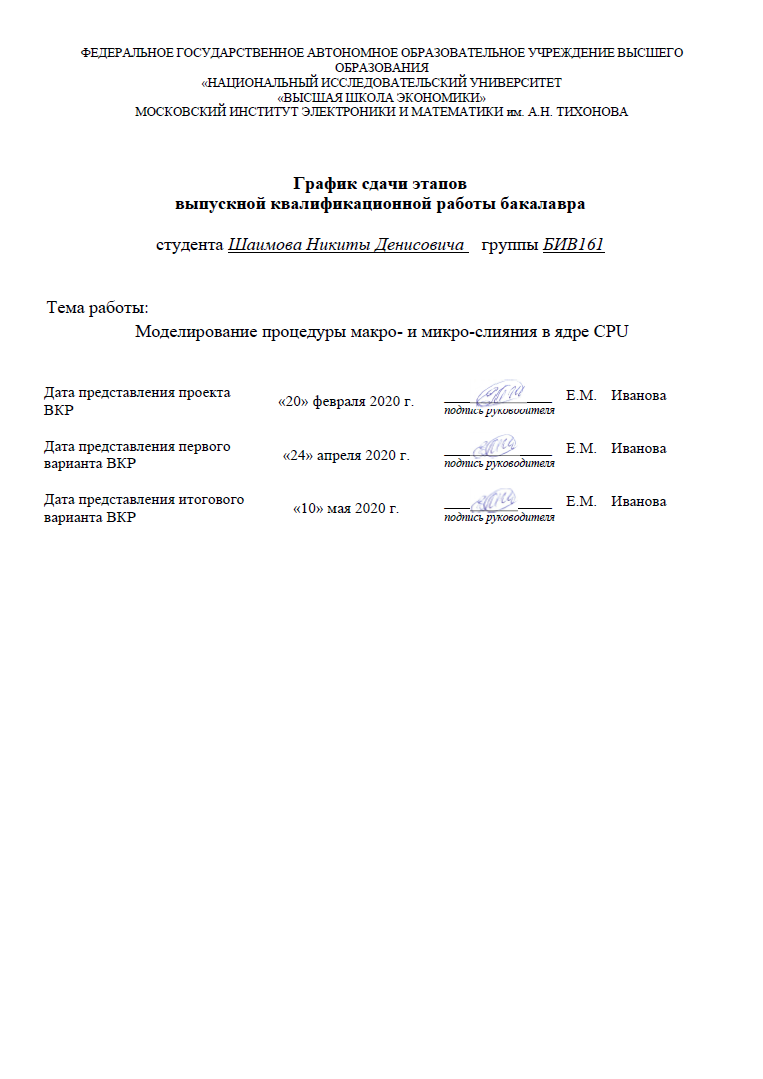
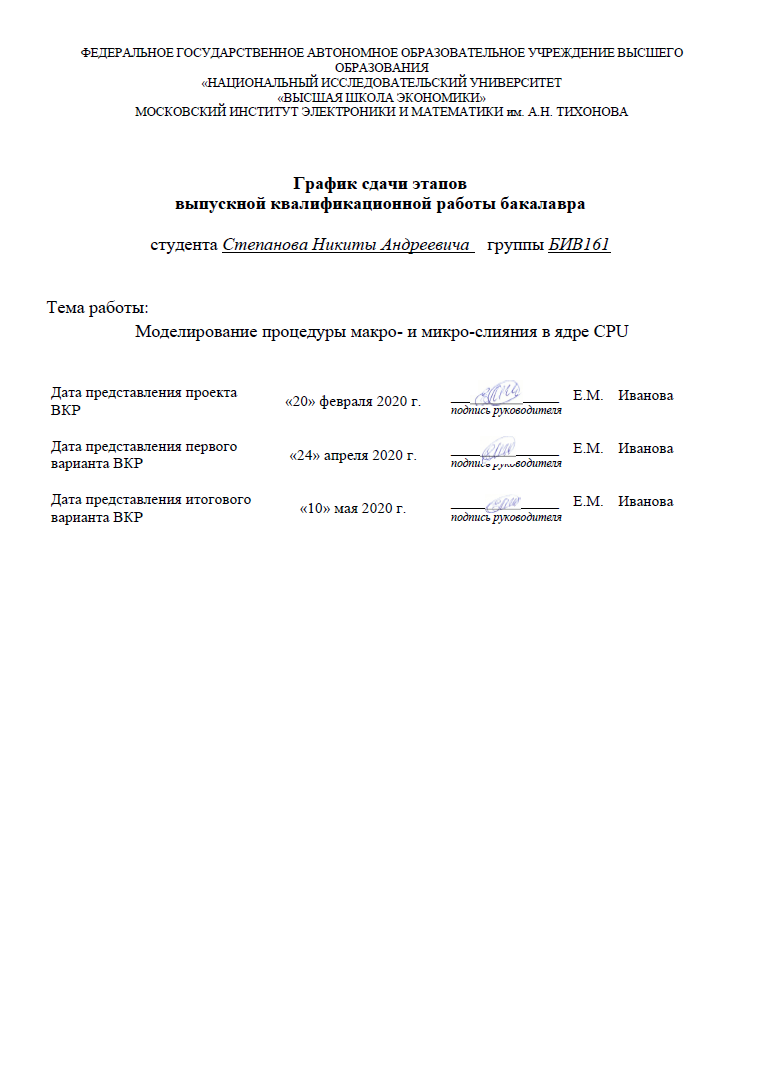
Выпускная квалификационная работа по направлению подготовки

09.03.01. Информатика и вычислительная техника

студента образовательной программы   
«Информатика и вычислительная техника»

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Студенты** | | | / | Шаимов Н.Д. | | *подпись* | *И.О. Фамилия* | |  | | | / | Степанов Н.А. | | *подпись* | *И.О. Фамилия* | |  | | |  |  | |  |  | |  | | |  |  | |  |  | |  | | |  |  | |  |  | | |  | | --- | | **Руководитель** | | Доцент, к.т.н., Е.М. Иванова | | *должность, звание, И.О. Фамилия* | | **Консультант** | |  | | *должность, звание, И.О. Фамилия* | | **Рецензент** | |  | | *должность, звание, И.О. Фамилия* | |





Аннотация

В данном документе приведено описание работы по созданию программной реализации модели слияния макро- и микро- операций в ядре CPU. Документ разделен на шесть смысловых разделов.

В разделе «Цели и задачи работы» определена основная цель работы и поставлены основные задачи для ее достижения.

В разделе «Обзор и анализ существующих технических решений в сфере моделирования работы частей ядра CPU» проанализированы существующие технические решения в сфере моделирования работы частей процессора, в том числе программные симуляторы, использующиеся для изучения работы процессора.

В разделе «Анализ принципов выполнения процедуры слияния» определены основные различия между архитектурами процессоров касательно слияния процедур, а также описаны принципы выполнения микро- и макро- слияния.

В разделе «Определение алгоритма процедуры слияния» представлено описание алгоритмов макро- и микро- слияния и перечни условий их выполнения.

В разделе «Моделируемые архитектуры» определен перечень моделируемых архитектур на основе процессоров Intel начиная от Pentium M до Skylake. Описаны особенности и параметры каждой архитектуры, влияющие на слияние операций.

В разделе «Разработка программной реализации модели» представлен обзор, анализ и обоснование выбора программных средств и языков программирования, описаны процессы создания графического интерфейса, разработки прототипа программного обеспечения и программной документации, а также представлены результаты тестирования, проводимого в ходе разработки.

Количество иллюстраций в документе: 12.

Количество приложений в документе: 10 - шт.

Количество страниц в документе: 91.

Annotation

This document describes the process of creating a software implementation model for macro- and micro- operations fusion in the CPU core. The document contains six semantic sections.

In the “Goals and Tasks of the Work” section, the main goal of the work is defined and the main tasks for its achievement are set.

The section "Review and analysis of existing technical solutions in the field of modeling of the CPU core parts work" analyzes existing technical solutions in the sphere of modeling the work of processor parts, including software simulators, which are used for processor researches.

In the section “Analysis of the principles for the implementation of merger procedures”, the main differences between the merger procedures, as well as the principles for performing micro- and macro-actions, are identified.

In the section “Fusion algorithm definition”, the information about the macro- and micro-fusion algorithms and lists of conditions for their implementation is presented.

The “Simulated architectures” section defines a list of simulated architectures based on Intel processors ranging from Pentium M to Skylake. The features and parameters of each architecture that affect the fusion of operations are described.

In the section "Development of the model software implementation" provides an overview, analysis and justification of the software choice tools and programming languages, describes the processes of creating the graphical interface, development of the prototype and software documentation, as well as the results of program testing.

Оглавление

[Введение 12](#_Toc40107318)

[1 Цели и задачи работы 14](#_Toc40107319)

[1.1 Цель 14](#_Toc40107320)

[1.2 Задачи 14](#_Toc40107321)

[2 Обзор и анализ существующих технических решений в сфере моделирования работы частей ядра CPU 15](#_Toc40107322)

[3 Анализ принципов выполнения процедуры слияния 19](#_Toc40107323)

[3.1 Анализ различий в зависимости от архитектуры 19](#_Toc40107324)

[3.2 Принципы выполнения микро-слияния 20](#_Toc40107325)

[3.3 Принципы выполнения макро-слияния 21](#_Toc40107326)

[4 Определение алгоритма процедуры слияния 24](#_Toc40107327)

[4.1 Алгоритм микро-слияния 24](#_Toc40107328)

[4.2 Алгоритм макро-слияния 25](#_Toc40107329)

[5 Моделируемые архитектуры 29](#_Toc40107330)

[5.1 Pentium M 29](#_Toc40107331)

[5.2 Core 2 и Nehalem 30](#_Toc40107332)

[5.3 Sandy Bridge и Ivy Bridge 31](#_Toc40107333)

[5.4 Haswell и Broadwell 32](#_Toc40107334)

[5.5 Skylake 33](#_Toc40107335)

[6 Разработка программной реализации модели 34](#_Toc40107336)

[6.1 Обзор, анализ и обоснование выбора программных средств и языков программирования 34](#_Toc40107337)

[6.2 Разработка графического представления 35](#_Toc40107338)

[6.3 Разработка прототипа программного обеспечения 36](#_Toc40107339)

[6.4 Тестирование программного обеспечения 39](#_Toc40107340)

[6.5 Разработка программной документации 42](#_Toc40107341)

[Заключение 45](#_Toc40107342)

[Список использованных источников 47](#_Toc40107343)

[Приложение 1. Схема архитектуры Pentium III. 50](#_Toc40107344)

[Приложение 2. Схема архитектуры Core 2. 51](#_Toc40107345)

[Приложение 3. Схема архитектуры Nehalem. 52](#_Toc40107346)

[Приложение 4. Схема архитектуры Sandy Bridge. 53](#_Toc40107347)

[Приложение 5. Схема архитектуры Ivy Bridge. 54](#_Toc40107348)

[Приложение 6. Схема архитектуры Haswell. 55](#_Toc40107349)

[Приложение 7. Схема архитектуры Broadwell. 56](#_Toc40107350)

[Приложение 8. Схема архитектуры Skylake. 57](#_Toc40107351)

[Приложение 9. Примеры готовых вариантов кода. 58](#_Toc40107352)

[Приложение 10. Программная документация. 60](#_Toc40107353)

Введение

Последние десятилетия увеличение производительности процессоров ведется за счет уменьшения размера элементов кремниевых микросхем и увеличения их количества на единицу площади. Однако в скором времени инженеры-конструкторы в погоне за производительностью процессоров столкнутся с физическим барьером минимального размера технологического процесса. Считается что данный предел находится в пределах 1-2 нм. При данном техпроцессе может возникать эффект квантового туннелирования [1]. Также меньший техпроцесс удорожает производство и это может привести к нерентабельности производимых процессоров. Уже сейчас некоторые крупные компании испытывают проблемы при переходе на меньший техпроцесс. Вследствие этого в ближайшее время довольно остро встанет вопрос о смене кремниевой основы на иную и переход займет значительное количество времени, а современные тенденции требуют постоянного увеличения производительности. Один из возможных вариантов решения данной проблемы является оптимизация существующих решений различными путями.

Основным показателем производительности процессоров является скорость выполнения операций. Один из возможных вариантов увеличения скорости выполнения позволить нескольким операциям выполнятся одновременно что привело к появлению многоядерных процессоров и многопоточным вычислениям. Однако такой вариант приводит к тому что необходимо перестраивать вычисления разбивая задачу на подзадачи и передавая их потокам. Кроме того, таким образом удается увеличить число одновременно исполняемых программ, но каждая из программ в отдельности выполняется не быстрее, а возможно даже замедляется, из-за разделения ресурсов ядра между потоками. Для увеличения производительности одного потока необходимо оптимизировать работу самого ядра и алгоритма исполнения каждой инструкции в отдельности.

Ядро процессора в современном понимании представляет из себя конвейер, разбивающий команды на несколько простых операций с одновременным их исполнением. В результате могут возникать задержки из-за занятости определенных модулей или нехватки пропускной способности шины так называемые узкие места или bottleneck.

Производители процессоров разрабатывают различные решения данных проблем. В частности, для оптимизации пропускной способности микро- и макро- операций была разработана технология слияния операций что позволило увеличить пропускную способность путём объединения нескольких последовательных операций в одну. Развивая данную технологию и увеличивая список возможных вариантов слияния можно добиться гораздо большего увеличения пропускной способности.

Для изучения особенностей функционирования современных процессоров удобно иметь компьютерный симулятор, демонстрирующий все архитектурные решения, в том числе и данной технологии выполнения микро- и макро- операций. К сожалению единственный вариант на текущий момент — это производить симуляции на конкретных процессорах и платформах. Отсутствие программной модели замедляет исследования в данной сфере и не позволяет проверить влияние гипотетических вариантов слияния на производительность.

Также при подготовке специалистов в профильных заведениях отсутствие материалов по части технологий и моделей работы некоторых блоков процессора ведет к более низкому уровню подготовки в определенных областях. Наличие программной модели позволит дополнить существующие учебные курсы профильных учебных заведений и расширить спектр подготовки.

Наша работа направлена на устранение данных проблем путем создания программной модели работы процедуры слияния макро- и микро- операций что позволяет производить симуляцию работы как существующих архитектур так и путём изменения параметров модели.

Разработка программной модели производится на языке Python с использованием библиотек PyQt5 для реализации графического интерфейса и PyInstall для генерации независимого исполняемого файла.

1. Цели и задачи работы
   1. Цель

Данная работа ставит следующую цель:

Создать инструмент для проведения дальнейших исследований в области оптимизации исполнения процессорных команд с возможностью визуализации процедуры макро- и микро- слияния для возможности обучения и использования в учебных курсах.

* 1. Задачи

Для достижения поставленной цели необходимо решить следующие задачи:

* Определить основные принципы работы слияния
* Определить различия в работе слияния в зависимости от моделируемой архитектуры процессора
* Определить список моделируемых архитектур
* Разработать алгоритмы слияния операций
* Провести анализ и обосновать выбор программных средств и языков программирования
* Разработать программную реализацию модели с графическим интерфейсом
* Разработать варианты исходных данных для инициализации модели

1. Обзор и анализ существующих технических решений в сфере моделирования работы частей ядра CPU

Прежде чем приступать к выполнению задач нами был проведен анализ существующих решений в сфере моделирования работы частей ядра CPU. В результате были рассмотрены различные программные симуляторы, позволяющие изучить работу частей процессоров. Среди изученных симуляторов в качестве примера можно привести несколько решений:

DrMIPS – является бесплатным симулятором MIPS архитектуры с открытым исходным кодом [2]. Обладает удобным графическим интерфейсом и позволяет выстраивать собственную архитектуру компонентов. Из возможностей можно также выделить возможность пошагового выполнения операций и симуляцию задержек с указанием критического пути. Из недостатков можно отметить что симуляция ведется только для архитектуры MIPS, имеет специфическое применение и значительно проще архитектуры современных десктопных процессоров, что сужает круг возможных пользователей программы. Интерфейс программы представлен на рис. 1.

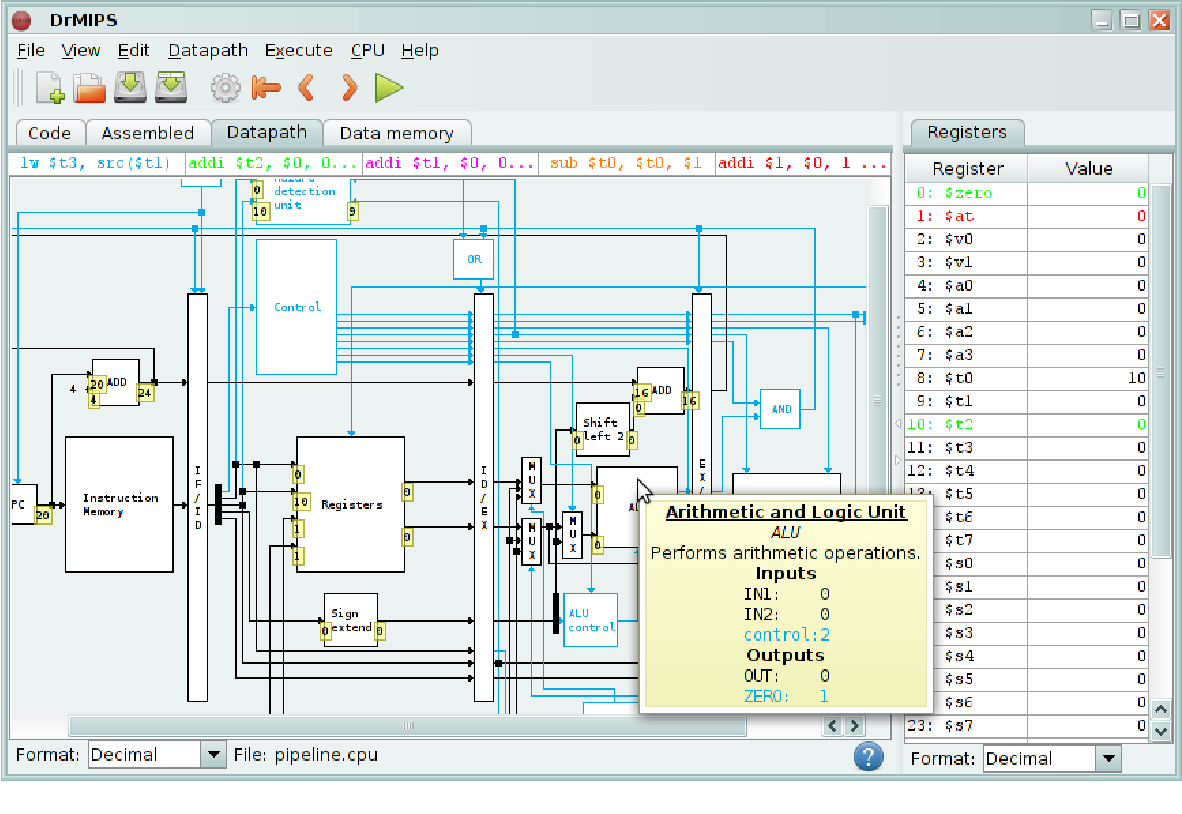


Рис. 1. Симулятор DrMIPS.

MikroSim – симулятор работы виртуального процессора созданный в образовательных целях [3]. Данный симулятор позволяет наглядно изучить все базовые аспекты работы процессора в наглядной форме на примере простейших программ. Позволяет проследить исполнение микрокода на всех этапах, а также оценить скорость выполнения операций благодаря встроенным инструментам для отслеживания частоты и оценки вычислительной мощности. Однако данный симулятор не подразумевает изменение архитектуры что ограничивает его применение в сугубо учебных целях для чего он и был создан. Интерфейс симулятора можно наблюдать на рис. 2.

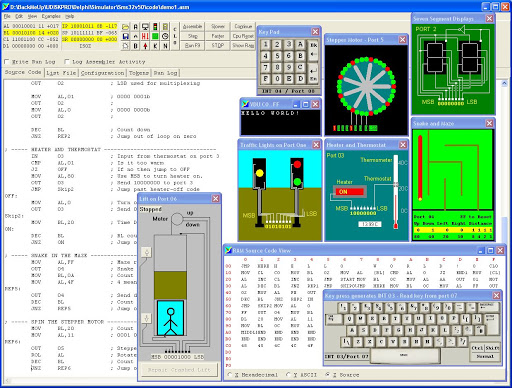


Рис. 2. Симулятор MikroSim.

YASS – Интерактивный мульти-уровневый симулятор работы процессора, разработанный для помощи в преподавании и изучении современных компьютерных технологий [4]. Предоставляет широкие возможности по изменению условий симуляции и включает в себя множественные функции для помощи в изучении наиболее сложных для понимания аспектов работы процессора. Данный симулятор состоит из 3 компонентов, включающих в себя: обучающий компилятор, симулятор процессора и симулятор операционной системы. Компилятор служит для интерактивного обучения ассемблеру и позволяет рассмотреть преобразование более высокоуровневого кода в макро-операции, а также для передачи кода двум симуляторам. Симулятор процессора основан на RISC архитектуре и позволяет отследить процесс выполнения команд. Симулятор операционной системы позволяет отследить два главных аспекта системы: распределение ресурсов и оперирование памятью. Внешний вид симулятора представлен на рис. 3.

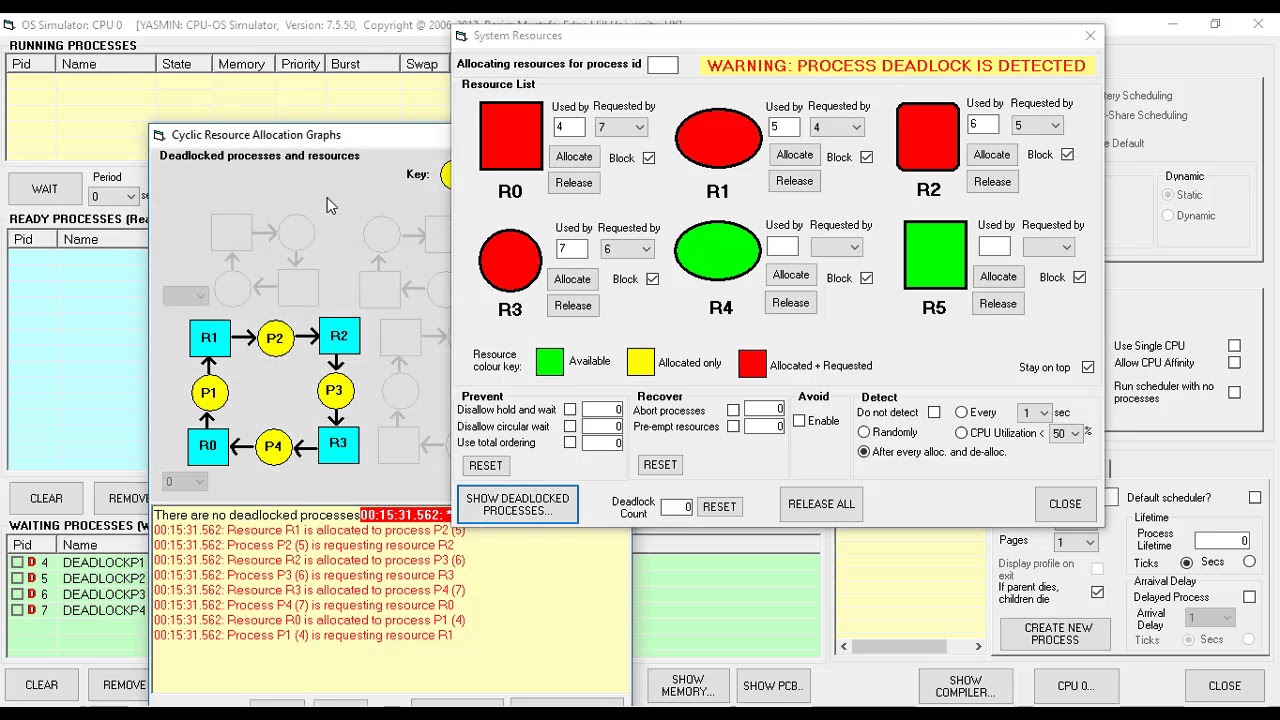


Рис. 3. Симулятор YASS.

Выводы

Для всех проанализированных решений можно сделать следующие выводы:

* Большинство существующих технических решений являются кране устаревшими так как были выпущены более 10 лет назад и лишь малую часть можно считать современными решениями.
* Часть симуляторов не имеют графического представления, и информация предоставляется в текстовом виде.
* У всех решений отсутствует возможность симулировать существующие архитектуры, однако у некоторых имеется возможность гибко настраивать симуляцию что позволяет вручную создать архитектуру, приближенную к настоящей, что безусловно интересно, но очень трудоемко.
* Некоторые решения довольно сложны в освоении, что затрудняет их использование, а в некоторых случаях исключает возможность использования в учебных целях.
* Ни в одном из найденных решений нет возможности проследить процедуру слияния операций визуально.

1. Анализ принципов выполнения процедуры слияния
   1. Анализ различий в зависимости от архитектуры

Прежде чем приступать к описанию процедуры слияния операций важно определить понятия макро и микро операций и их различия в зависимости от архитектур компаний производителей. На данный момент большинство центральных процессоров в мире производится двумя компаниями Advanced Micro Devices (далее AMD) и Intel.

Разница в структуре процессоров настолько значительна что понятия макро и микро операций не являются исключением. В процессорах компании Intel под макро-операцией подразумевается x86 инструкция произвольной длины с возможностью выполнения нескольких операций с памятью и вычислениями одновременно, а микро-операция определяется как одна или более операции фиксированной длины R-типа, декодированные из x86 инструкции [5]. Макро-операцией в терминологии AMD считается операция фиксированной длины которые могут состоять из операции с памятью и арифметической операции, одна макро-операция может выполнить операцию чтения, изменения и записи. Микро-операций считается одна простейшая операция например одна вычислительная операция или одно действие с памятью [6]. Данные операции являются составными частями макро-операции.

Так как технология слияния макро операций запатентована компанией Intel в 2000 году [7], в процессорах компании AMD данная технология отсутствует. В нотации Intel макро-слияние это слияние двух смежных макро-операций в одну операцию до или во время декодирования.

Относительно микро-операций процессоры также имеют различия. В процессорах Intel слияние микро-операций происходит после декодирования макро-операции [5]. Однако в процессорах AMD разбиение на микро-операции и следовательно слияние операций происходит только перед непосредственным исполнением [6].

Принимая во внимание факты изложенные выше и отсутствие слияния макро-операций в процессорах компании AMD всё дальнейшее рассмотрение процессов и определения будут изложены применительно к процессорам компании Intel.

* 1. Принципы выполнения микро-слияния

Процедура слияния микро-операции позволяет двум и более микро-операциям, полученным в результате декодирования макро-операции, сливаться в одну операцию, что позволяет экономить место в буферах вследствие чего растет пропускная способность, а слитые операции выполняются на тех же исполнительных блоках, что и не объединенные операции [8]. Для удобства участки процессора где микро-операции передвигаются в объединённой форме обозначают как fused domain, а те где операции передвигаются раздельно unfused domain. Схему движения микро-операций можно наблюдать на рис. 4.

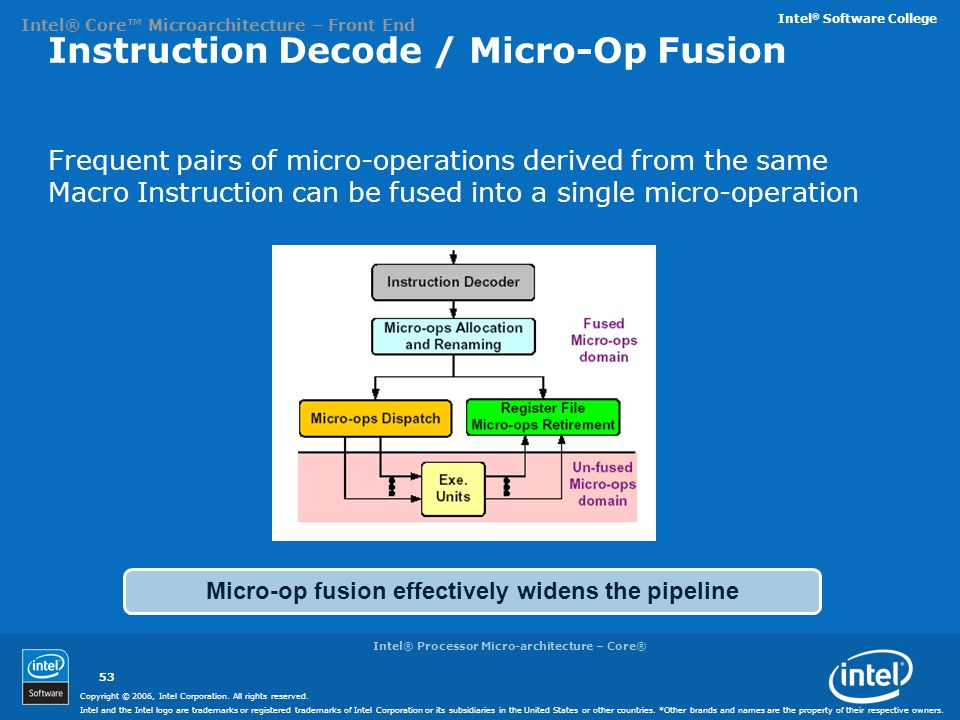


Рис. 4. Схема движения микро-операций.

Микро-слиянию подвержены только два типа комбинаций: операции записи в память и операции чтения-изменения памяти [8]. В первом случае происходит объединение двух последовательных операций вычисления адреса в памяти и операции перемещения данных. Во втором типе происходит объединение двух последовательных операций чтения данных из памяти и операции преобразования данных. В случае если среди операций присутствуют оба типа возможных слияний происходит только слияние первого типа, а именно двух последовательных операций вычисления адреса и записи в память при этом слияния последовательных операций чтения и изменения данных не происходит, однако в современных архитектурах возможно одновременное слияние двух типов.

* 1. Принципы выполнения макро-слияния

Некоторые макро-операции могут объединятся в одну микро-операцию при декодировании. В некоторых случаях декодер сливает арифметическую инструкцию или инструкцию сравнения или логическую инструкцию с инструкцией условного перехода в одну микро-операцию и полученная таким образом операция не разделяется на этапе исполнения и выполнятся как одна микро-операция [8]. На рис. 5 и 6 можно наблюдать пример влияния слияния макро-операций на пропускную способность.

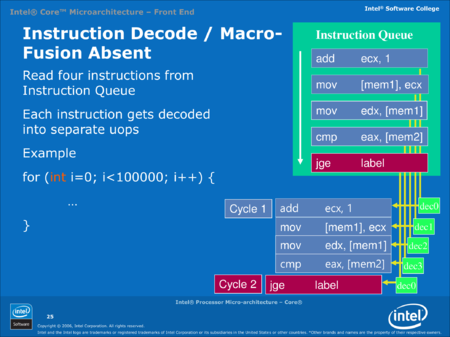


Рис. 5. Пример выполнения операций без слияния макро-операций.



Рис. 6. Пример выполнения операций с слиянием макро-операций.

Слияние может производится любым из четырёх декодеров, но не одновременно, что позволяет повысить пропускную способность с 4 до 5 инструкций в такт. Однако использование макро-слияния может незначительно уменьшить пропускную способность в противовес значительному выигрышу от слияния. Если инструкция имеет возможность слияния и попадает в последний из 4 декодеров она будет отложена и направлена в первый декодер на следующий такт для проверки возможности слияния со следующей инструкцией [8].

Макро-слияние и микро-слияние могут выполняться вместе. Команды чтения, сравнения и условного перехода образуют одну единственную микро-операцию. Например, операции CMP или TEST оперирующие операндом памяти и операндом регистра сливаются на макро и микро уровне с последующим условным переходом образуя единую микро операцию включающую в себя чтение, сравнение и переход. Однако данное слияние не всегда является возможным из-за ограничений размера памяти на микро-операцию [8]. Данный тип слияния часто указывается как обычное макро-слияние, однако без содействия микро-слияния макро-слияние было бы возможно только при слиянии макро-операций производящих только изменение флагов и переход.

Исходя из указанной информации для получения максимального эффекта от макро-слияния грамотному программисту следует учитывать следующее:

* В зависимости от используемой структуры микропроцессора располагать возможные для слияния инструкции рядом;
* Между парами сливаемых инструкций должно располагаться минимум 3 других инструкции.

1. Определение алгоритма процедуры слияния
   1. Алгоритм микро-слияния

Для симуляции процедуры слияния микро-операций был выведен обобщенный алгоритм для проверки всех условий и выполнения слияния.

Рассмотрим алгоритм процедуры слияния микро-операций в ядре процессора. Процедура слияния позволяет двум микро-операциям, полученным в результате декодирования макро-операции, сливаться в одну операцию, что приводит к улучшению проводящей способности процессора, увеличивая его производительность и общую скорость исполнения инструкций за счет того что на всех этапах за исключением исполнения данные операции считаются за единую операцию [9].

Микро-операции получаются в результате декодирования макро-операции. Среди них можно выделить 4 основные категории:

* Чтение (Read/Load)
* Изменение (Modify)
* Вычисление и сохранение адреса (Address)
* Запись (Write/Save)

Микро-слиянию подвержены два типа пар: последовательные операции вычисления адреса в памяти и перемещения данных и последовательные операции чтения данных и изменения данных. В зависимости от архитектуры выполняется слияние только одной пары или двух, но разных типов.

В случае ситуации, когда среди операций присутствуют две пары разных типов, а архитектура не позволяет выполнять слияние двух пар разных типов выполняется только слияние последовательных операций вычисления адреса в памяти и перемещения данных.

Среди условий, которые проверяются при слиянии микро-операций можно выделить допустимость типов операндов в макро-операции которая была в последствии декодирована. Также, микро-операции получаемые после декодирования зависят от моделируемой архитектуры, моделирование ведется в соответствии с выбранной архитектурой в которой будет исполняться слияние микро-операций. Например, в архитектуре «Pentium M» в макро-операцию условного перехода LOOP входит 11 микро-операций типа “Modify”, а в архитектуре следующего поколения «Core 2» в ту же декодированную макро-операцию LOOP входит 7 микро-операций того же типа [9].

* 1. Алгоритм макро-слияния

Для симуляции процедуры слияния макро-операций в представленных архитектурах был выведен обобщенный алгоритм для проверки всех условий и выполнения слияния.

Пары операндов, подлежащих слиянию, определяются моделируемой архитектурой. Также архитектура определяет допустимые операнды первой операции, разрядность регистров и прочие параметры такие как возможность слияния двух пар операций за такт.

В результате анализа процедуры слияния при помощи литературы были составлены списки операций, которые могут создать потенциальную пару для слияния [5][8].

Список первых операций в паре: ADD, SUB, ADC, SBB, INC, DEC, CMP, TEST, AND, OR, XOR, NOT, NEG.

Список вторых операций в паре: JA, JNA, JAE, JNAE, JB, JNB, JBE, JNBE, JC, JNC, JE, JNE, JG, JNG, JGE, JNGE, JL, JNL, JLE, JNLE, JS, JNS, JO, JNO, JP, JNP, JPO, JPE, JZ, JNZ, JCXZ, JECXZ, JRCXZ, LOOP.

Также был составлен список возможных операндов первой операции:

* Регистр, Регистр
* Регистр, Непосредственный операнд
* Регистр, Память
* Регистр, RIP (relative instruction-pointer) адрес
* Память, Непосредственный операнд
* Регистр
* Память
* RIP (relative instruction-pointer) адрес

В качестве опциональных условий были выделены следующие пункты:

* Возможность начала условного перехода на границе в 16 бит
* Возможность пересечение условным переходом границы в 16 бит
* Возможность слияния двух пар за такт
* Перенос первой операции из потенциальной пары на следующий такт при попадании на последний декодер

А также допустимые режимы слияния по разрядности регистров: 16, 32 и 64 бита.

В итоге алгоритм слияния выглядит следующим образом (рис. 7):

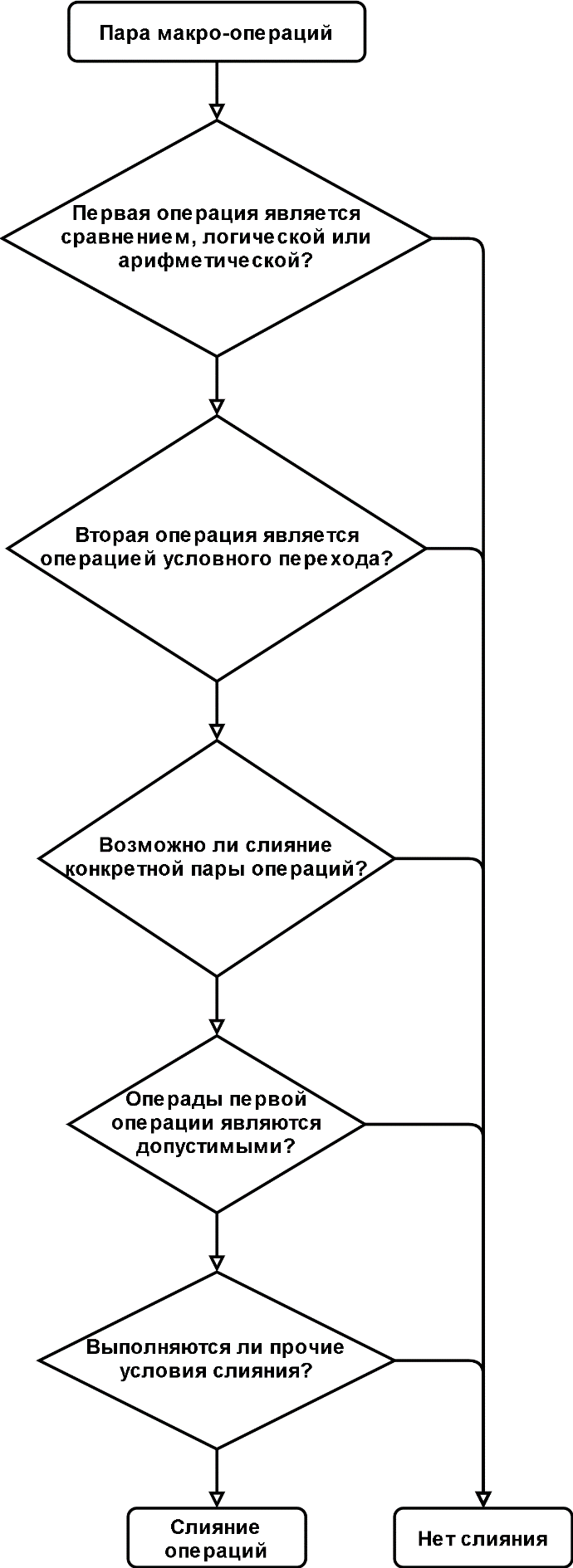


Рис. 7. Алгоритм макро-слияния.

Во-первых, на вход поступает пара инструкций где первая инструкция из пары является операцией сравнения, логической операцией или арифметической операцией, а вторая является операцией условного перехода.

Во-вторых, выполняется проверка возможности слияния макро-операций на данной архитектуре. В первую очередь проверятся доступность макро-слияния на выбранной архитектуре. Затем выполняется проверка возможности слияния конкретно данной пары операций. Далее проверятся что комбинация операндов первой операции является допустимой. И наконец проверяются оставшиеся условия такие как количество слияний за один такт или разрядность регистров.

В-третьих, при выполнении всех условий производится слияние данных операций в одну микро-операцию.

1. Моделируемые архитектуры

Для разработки модели мы выделяем следующие архитектуры процессоров Intel:

* Pentium M
* Core 2 и Nehalem
* Sandy Bridge и Ivy Bridge
* Haswell и Broadwell
* Skylake

Рассмотрим каждую из архитектур более подробно и рассмотрим их различия в процедуре слияния макро- и микро- операций, для этого воспользуемся крайне полезным мануалом Агнера Фога [8]. Также в других его мануалах представлен широкий спектр примеров и тестов [9][10].

* 1. Pentium M

Процессоры архитектуры Pentium M были представлены компанией Intel в марте 2003 года [11]. Данная архитектура является улучшенной версией архитектуры Pentium III а также первой архитектурой в линейке Intel поддерживающей слияние микро-операций [8]. Ввиду того что архитектура довольно старая и найти её схему крайне тяжело, а также малое количество различий с Pentium III в (Приложении 1) представлена архитектура Pentium III. Так как слияние макро-операций на данной архитектуре отсутствует рассмотрим только слияние микро-операций.

Согласно мануалу [8] архитектура Pentium M может выполнять слияние следующих микро-операций:

Операции вычисления адреса в памяти и перемещения данных.

Операции чтения-изменения работающие с регистрами общего назначения, с плавающей точкой и MMX регистрами кроме XMM регистров.

В случае операции чтения-изменения-записи будет выполнятся только слияние микро-операций вычисления адреса в памяти и записи данных.

Архитектуры Core 2 и Nehalem рассмотренные далее будут иметь больше возможностей для слияния.

* 1. Core 2 и Nehalem

Архитектуры Core 2 и Nehalem были представлены в июле 2006 года и ноябре 2008 года соответственно [12][13]. Так как данные процессоры имеют схожие алгоритмы слияния макро- и микро- операции и структуру архитектуры эта и последующие пары архитектуры будут рассмотрены вместе. Устройство данных архитектур изображено в (Приложение 2) и (Приложение 3). В отличии от Pentium M данные архитектуры могут не только выполнять слияние не только микро-операций но и макро-операций.

Так как принципы слияния микро-операций практически не отличаются от предыдущей архитектуры рассмотрим отличия в процедуре. Самым заметным отличием в данных архитектурах является возможность слияния в операции чтения-изменения-записи не только микро-операций  вычисления адреса в памяти и перемещения данных но и чтения-изменения. Также стали доступны слияния чтения-изменения оперирующие с некоторыми XMM регистрами. Операции содержащие RIP регистр и непосредственные операнды не подлежат слиянию.

Далее рассмотрим слияние макро-операций. Архитектуры Core 2 и Nehalem могут выполнять слияние макро-операций, а именно сливать две инструкции в одну микро-операцию. Декодеры сливают CMP или TEST инструкцию вместе с последующей инструкцией условного перехода получая в результате одну compare-and-branch микро-операцию в определенных случаях. Полученная микро-операция не разделяется и попадает в исполнительный порт 5. В результате макро слияние позволяет экономить пропускную способность на всем пути операции начиная от декодирования, одно не помогает увеличить пропускную способность до декодирования.

Слияние макро-операций возможно только при выполнении следующих условий:

* Первая инструкция CMP или TEST и следующая инструкция условный переход за исключением JECXZ и LOOP;
* Инструкции CMP и TEST могут иметь два операнда такие как: два регистра, регистр и непосредственный операнд, регистр и операнд памяти, кроме операнда памяти вместе с непосредственным операндом;
* Core 2 может выполнять макро слияние в 16 битном и 32 битном режиме, Nehalem также может выполнять и в 64 битном режиме;
* Операторы условного перехода проверяющие флаги CF и ZF (JE, JNE, JB, JBE, JA, JAE, ...) могут выполнять слияние с предыдущими инструкциями TEST или CMP, включая все без знаковые сравнения;
* Сравнения с учетом знаков (JL, JLE, JG, JGE, ...) могут выполнять слияние с предыдущими инструкциями TEST или CMP на архитектуре Nehalem, на архитектуре Core 2 только с TEST;
* Остальные условные переходы (JO, JNO, JP, JNP, JS, JNS, ...) могут выполнять слияние только с предыдущим TEST но не с CMP;
* Между сливаемыми инструкциями не может быть других инструкций;
* Инструкции условного перехода не должны начинаться на границе 16 бит или пересекать её;
* Если две пары подлежащих слиянию инструкций достигают этапа декодирования в один такт только первая пара будет объединена.
  1. Sandy Bridge и Ivy Bridge

Следующая пара рассматриваемых архитектур Sandy Bridge и Ivy Bridge были представлены в 2009 и 2011 году соответственно [14][15]. Согласно стратегии “тик-так” Ivy Bridge является этапом “тик”, а именно этапом миниатюризации технологического процесса [16]. Схемы архитектур приведены в (Приложение 4) и (Приложении 5).

В отношении слияния микро-операции данные архитектуры не отличаются от предыдущих.

Архитектуры Sandy Bridge и Ivy Bridge могут выполнять макро слияние в больших случаях в отличии от предыдущих архитектур. Основные различия следующие:

* Инструкции CMP, ADD и SUB могут выполнять слияние со знаковыми и без знаковыми инструкциями условного перехода;
* Инструкции INC и DEC могут сливаться со знаковыми инструкциями условного перехода;
* Инструкции TEST и AND могут сливаться со всеми инструкциями условного перехода;
* Первая инструкция может содержать непосредственный операнд или операнд памяти, но не оба;
* Инструкция не может содержать операнд адреса памяти в качестве точки назначения или операнд памяти имеющий отношение к RIP;
* Инструкции JECXZ и LOOP не могут участвовать в слиянии.

Более наглядное представление вариантов слияния представлено в таблице 1.

Таблица 1. Варианты слияния в архитектурах Sandy Bridge и Ivy Bridge.

|  |  |  |
| --- | --- | --- |
| Первая инструкция в паре | Вторая инструкция в паре | |
| Могут сливаться(включая инвертированные варианты инструкций) | Не могут сливаться |
| cmp, add, sub | jz, jc, jb, jbe, ja, jae, jl, jle, jg, jge (jnz, jnc, jnb, jnbe, jna, jnae, jnl, jnle, jng, jnge) | js, jp, jo |
| inc, dec | jz, jl, jle, jg, jge (jnz, jnl, jnle, jng, jnge) | jc, jb, ja, js, jp, jo |
| test, and | все |  |

* 1. Haswell и Broadwell

Архитектуры Haswell и Broadwell были представлены в 2013 году и 2014 году [17][18]. Haswell следует за Ivy Bridge и является “таком” что согласно стратегии Intel является изменением архитектуры, а Broadwell “тиком” и уменьшает техпроцесс до 14 нм. Схемы архитектур приведены в (Приложение 6) и (Приложении 7).

В процедуре слияния микро-операций в данных архитектурах изменений нет за исключением того что теперь декодеры могут оперировать с четырьмя слитыми микро-операциями в такт.

В процедуре слияния макро-операций на данных архитектурах произошли некоторые изменения. Декодеры сливают логическую или арифметическую инструкцию вместе с последующей инструкцией условного перехода получая в результате одну compare-and-branch микро-операцию в определенных случаях. Полученная микро-операция не разделяется и попадает в исполнительный порт 0 или 6. В отличии от предыдущих архитектур слияние теперь можно даже при пересечении границы в 16 бит. Теперь в один и тот же такт допускаются две слитые пары макро-операций.

* 1. Skylake

Архитектура Skylake была представлена в 2015 году и является завершающей в нашем списке моделируемых архитектур [19]. Следуя за архитектурой Broadwell является “таком” и является кардинальным изменением архитектуры без изменения технологического процесса. Схема архитектуры приведена в (Приложении 8).

У данной архитектуры отсутствуют отличия в процедуре слияния макро- и микро- операций по сравнению с предыдущими архитектурами.

1. Разработка программной реализации модели
   1. Обзор, анализ и обоснование выбора программных средств и языков программирования

Прежде всего для разработки программной модели нужно определить язык программирования для написания программной части. Среди современных и широко используемых языков можно выделить следующие языки программирования:

* С++
* Java
* Python
* C
* C#

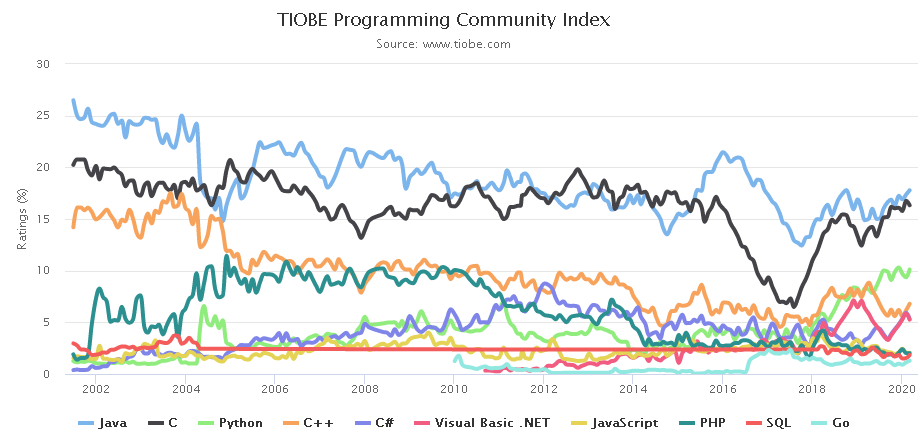


Рис.8. График популярности языков программирования.

Рассмотрим график популярности языков программирования на основе индекса TIOBE (рис. 8) [20]. Данный индекс оценивает популярность языка по частоте поисковых запросов, содержащих название языка. Согласно графику, можно наблюдать что на протяжении многих лет самыми популярными языками были C, Java и С++. Однако на текущий момент самые популярные языки это C, Java и Python. Ввиду наличия более обширного опыта работы и удобства при работе с текстом наш выбор пал на Python. Стоит также отметить рост популярности языка за последние пару лет. Это связано с простотой и удобством языка что повлекло его широкое распространение в научной сфере. Однако у данного языка есть и недостатки в числе которых низкая скорость работы и интерпретируемость языка. Низкая скорость работы не повлияет на работу нашей модели ввиду её небольшого объёма. Однако интерпретируемость языка приводит к тому что нам необходимо найти способ создать исполняемый файл.

После поисков способа компиляции кода на Python в исполняемый файл формата .exe мы нашли библиотеку PyInstaller. Согласно документации данной библиотеки [21], она позволяет упаковать код на Python и все используемые зависимости в один исполняемый файл формата .exe.

Для реализации графического интерфейса нам необходимо использовать библиотеку, позволяющую реализовать графическую оболочку. Опираясь на наш опыт разработки интерфейсов под платформой Qt мы решили использовать библиотеку PyQt5. Данная библиотека позволяет реализовать интерфейс при помощи удобного редактора и связать его с кодом на Python [22][23][24].

* 1. Разработка графического представления

Графический интерфейс итоговой программы представляет собой стандартный набор элементов, созданных с помощью QT Designer (рис. 9). Слияние операций подсвечивается желтым цветом в таблицах (рис. 10 и 11). В распоряжении пользователя находится набор различных настроек, устанавливаемых с помощью выпадающих списков, списков с множественным и одинарным выбором. Интерфейс прост и понятен, что позволяет использовать программу без необходимости глубокого изучения её компонентов.

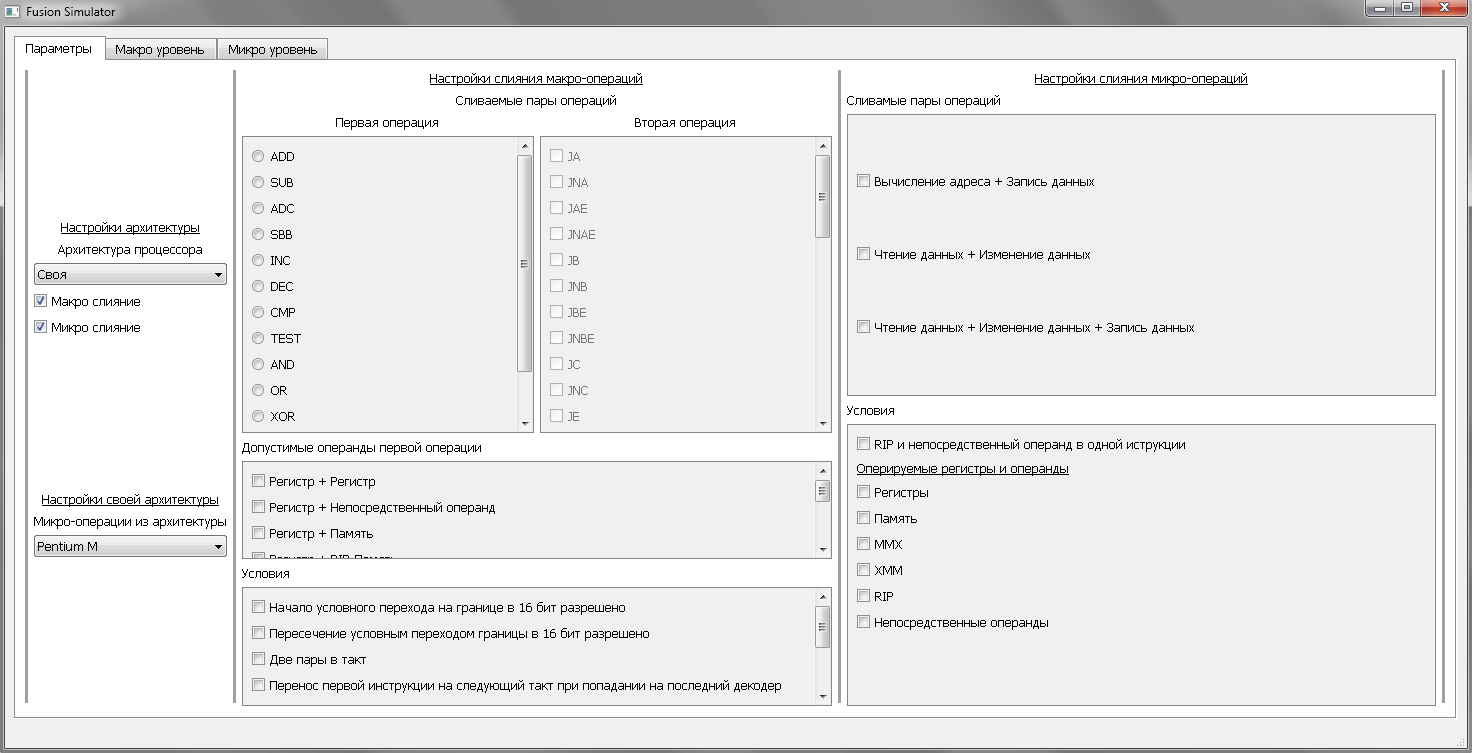


Рис. 9. Графический интерфейс программной модели.

Всё приложение поделено на три вкладки:

* Параметры, где пользователь может установить необходимые ему настройки микро- и макро- слияния;
* Макро уровень, где пользователь может записывать исполняемый на ассемблере код, и произвести симуляцию. Также в правой части отображается результат слияния;
* Микро уровень, где пользователь наблюдает полный спектр микро операций с их возможным слиянием;

Микро и макро операции отображаются в виде таблиц с номером такта, операндами и дополнительными параметрами, предоставляя пользователю полную информацию по их исполнению в блоках процессора.

* 1. Разработка прототипа программного обеспечения

В результате работ с использованием выбранных инструментов разработки был создан прототип программного обеспечения, позволяющий производить симуляцию процедуры слияния макро- и микро- операций в ядре CPU на архитектурах процессоров компании Intel, а также с применением собственных настроек симуляции.

Рассмотрим итоговый прототип программного обеспечения с точки зрения требований технического задания к возможностям программного варианта модели слияния макро- и микро- операций в ядре CPU.

Согласно заданию, программная модель должна позволять проследить выполнение процедуры слияния машинных инструкций на уровне макро- и микро- кода.

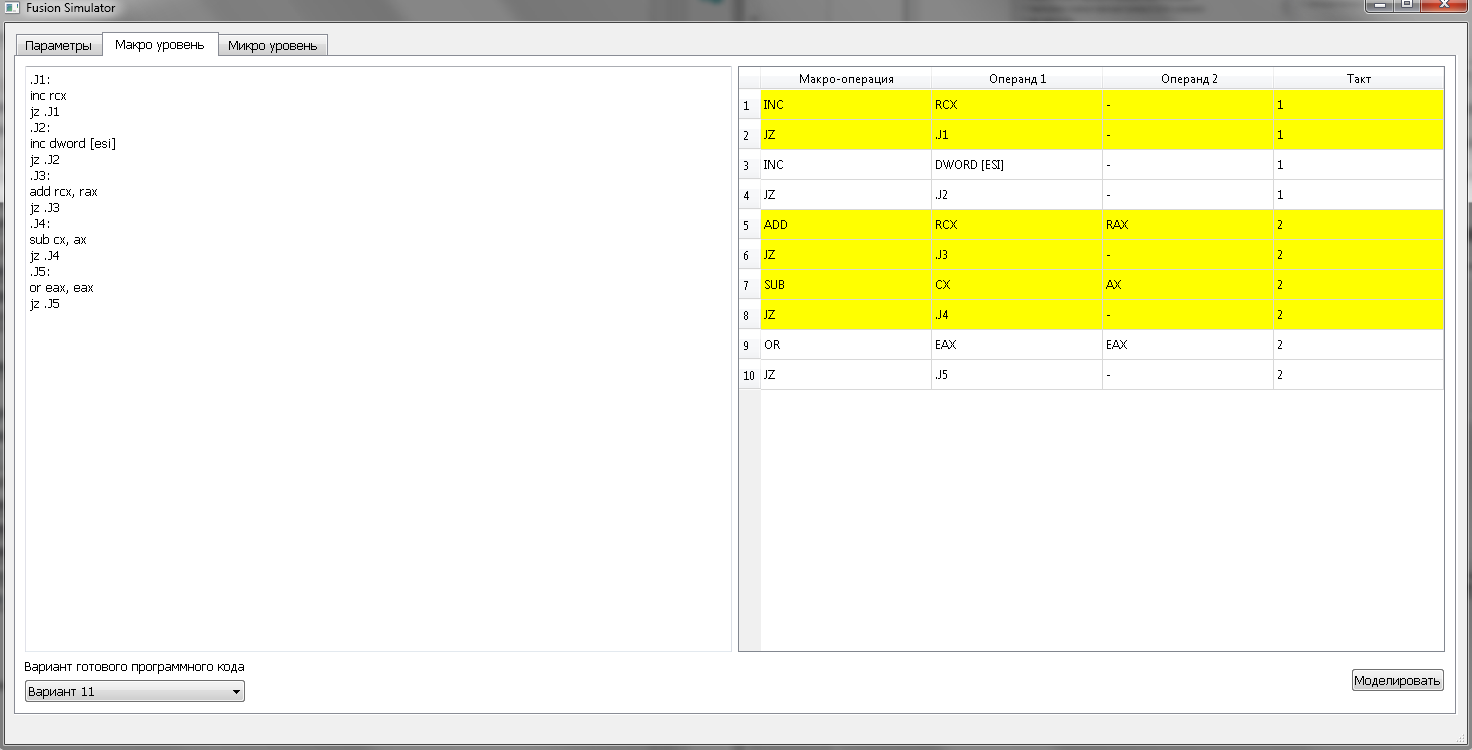


Рис. 10. Макро уровень симуляции.

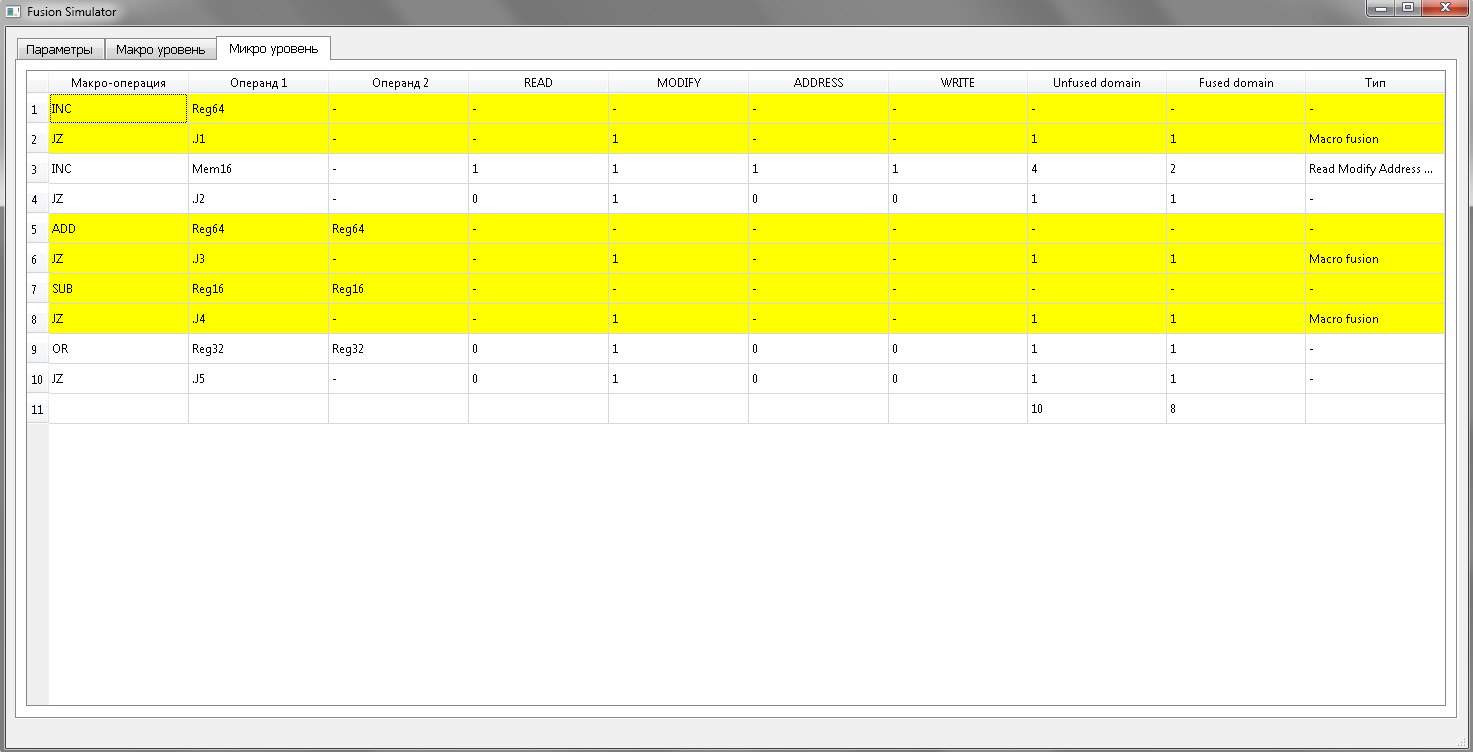


Рис. 11. Микро уровень симуляции.

Как можно наблюдать из рис. 10 и 11 данное условие выполняется. Программа указывает на сливаемые операции и учитывает их при подсчете итогового количества операций и тактов

Также среди условий технического задания указано что модель должна быть спроектирована в виде функционально независимого модуля, запускаемого под ОС Windows без необходимости установки дополнительного ПО. Данное условие выполнятся за счет использования библиотеки PyInstall описанной ранее что позволяет создать единый исполняемый exe файл без зависимостей.

Следующее требование гласит что исходными данными является фрагмент программного кода на Ассемблере, включающем несколько инструкций подлежащих слиянию. Как можно видеть на рис. 10 данное условие выполняется. Программа может оперировать только с ограниченным числом команд ввиду сложной реализации симуляции отдельных инструкций и результатов их выполнения. Также предусмотрена проверка кода перед запуском симуляции с выделением проблемных фрагментов что позволяет легко исправлять ошибки в программном коде и не допускать недопустимые для симуляции варианты программного кода, а также код содержащий ошибки в написании.

Также программа должна позволять выбирать один из вариантов готового исходного кода на Ассемблере. На рис. 10 представлен один из 15 вариантов исходного кода. Варианты готового программного кода позволяют продемонстрировать все возможности программы и различные варианты ситуаций, в которых те или иные настройки слияния оказывают наибольшее влияние (Приложение 9).

И последнее требование указывает на то что программа должна позволять выбирать архитектуру процессора для моделирования и выбирать какие инструкции подлежат слиянию. На рис. 9 можно наблюдать обширный список настроек и условий моделирования. Настройки позволяют создать свой уникальный вариант архитектуры для симуляции, а также проверить те или иные архитектуры с измененными настройками что позволяет проводить исследование влияния различных параметров слияния на итоговый результат.

* 1. Тестирование программного обеспечения

Существует множество методов тестирования. В случае тестирования программного приложения наиболее полезным является функциональное тестирование. Данный вид тестирования проводится с целью проверки соответствия программы функциональным требованиям в определенных условиях [ *Лайза Криспин, Джанет Грегори.* Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. — М.: «Вильямс», 2010. — 464 с. — (Addison-Wesley Signature Series). — 1000 экз. — [ISBN 978-5-8459-1625-9](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/9785845916259). ] . Остальные методы тестирования малоэффективны или не имеют смысла в случае тестирования данного программного приложения. Так, например, тестирование производительности программы на современных персональных компьютерах не приведет к нахождению ошибок или несоответствий ввиду низких требований к техническим средствам. Тестирование безопасности или локализации также не будет эффективным. В первом случае из-за высокой проектной защищенности (программа рассчитана на использование вне сетевых технологий), во втором – из-за расчета на использование программы лишь в одном регионе.

Программа представляет собой единое целое, в то же время состоящее из множества настроек и параметров выполнения. То есть, при включении или выключении различных параметров, выборе разных вариантов программного кода на ассемблере алгоритм выполнения программы изменяется, выполняется та или иная его часть. Это позволило нераздельно производить блочное (компонентное), интеграционное и системное тестирование без создания особых ситуаций. Компонентное тестирование подразумевает испытание отдельной части или компонента алгоритма [ [Bell, Ward](https://en.wikipedia.org/w/index.php?title=Ward_Bell&action=edit&redlink=1) (August 14, 2009). ["The Art of Unit Testing"](http://neverindoubtnet.blogspot.com/2009/08/art-of-unit-testing.html). *Never in Doubt*. Retrieved June 5, 2017. ]. Для его реализации достаточно изолировать нужный параметр, отвечающий за тестируемый компонент, от выполнения других параметров и запустить выполнение программы моделирования. Таким же образом производилось и интеграционное тестирование, которое предполагает испытание нескольких компонентов за одно выполнение алгоритма. Сложившаяся ситуация позволила объединить типы тестирования на конечных этапах разработки, и производить компонентное и интеграционное тестирование одновременно с системным (которое предполагает тестирование на полной системе и проверке ее соответствия требованиям [Critical Testing Processes. Plan, Prepare, Perform, Perfect, [Rex Black](https://www.google.ru/search?hl=ru&tbo=p&tbm=bks&q=inauthor:%22Rex+Black%22) Addison-Wesley Professional, 2004, ISBN-13: 978-0201748680]) после окончания создания основной части программы.

Далее можно выделить две основные стратегии – «Тестирование белым ящиком» и «Тестирование черным ящиком». Обычно, они производятся на разных стадиях разработки программы. Изначально, на стадии альфа-тестирования, являлось возможным только использование стратегии тестирования белым ящиком. Она подразумевает испытания компонентов, либо групп компонентов при видимом программном коде [Peli de Halleux, Nikolaj Bjorner. [Pex and Moles – Isolation and White Box Unit Testing for .NET](https://www.microsoft.com/en-us/research/project/pex-and-moles-isolation-and-white-box-unit-testing-for-net/), February 21, 2007. ]. На данной стадии тестирования выявлялись основные ошибки, было исправлено наибольшее количество недоработок. Этап альфа-тестирования ограничен окончанием разработки основной части приложения, связью графического интерфейса с программным алгоритмом и, как итог, созданием полной системы. Далее, на стадии бета-тестирования становится возможным использование стратегии тестирования черным ящиком. В ходе данного тестирования не используется знание внутреннего кода программы [*Бейзер Б.* Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем. — Питер, 2004. — 320 с. — [ISBN 5-94723-698-2](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/5947236982).]. Тестирование черным ящиком применялось для целостной системы, алгоритм которой исполнялся от начала и до конца без каких-либо прерываний и остановок. Это позволило выявить несоответствия во взаимодействии целых групп компонентов, а также доработать функциональность системы в соответствии с поставленными требованиями. Бета-тестирование, в отличии от альфа-, предполагает отладку конечной версии программы и может продолжаться на протяжении всего времени работы пользователей с приложением, что приводит к возможности постоянных доработок.

Для тестирования приложения использовался код на ассемблере, включающий в себя всевозможные вариации микро- и макро- слияния и позволяющий в полной мере проявить функционал программы.

В ходе тестирования программного продукта было выявлено несколько проблем:

Во-первых, выявлена невозможность реализации моделирования нескольких параметров слияния из-за сложности моделирования работы процессорных блоков, а именно тех что проверяют 16 битной границы в памяти что потребовало бы доскональной симуляции работы памяти. Подобные параметры в итоговом приложении представлены в параметрах, однако их симуляция и проверка не производится.

Во-вторых, выявлены неточности в работе алгоритма модели. Так, например, было обнаружено, что не все случаи слияния микро-операций в программе обрабатываются верно. Из-за недостаточности информации по теме слияния операций, не удалось изначально определить ситуации, при которых микро-операции будут сливаться после уже произведённого слияния макро-операций. Для программной реализации данных случаев был улучшен алгоритм работы приложения и принято решение расширить таблицу микро-слияния на поле «Тип», что позволяет пользователю понять, в ходе какого исполнительного процесса произошло слияние операций.

В-третьих, выявлен недостаточный функционал программы. Из-за влияния архитектуры на исполнение микро-операций и их слияние, ранее не было возможности точно производить модуляцию работы блоков процессора. Поэтому, после тестирования, был расширен функционал программы, добавлены необходимые данные для модуляции и добавлена возможность выбирать архитектуру, ориентируясь на которую программный алгоритм определяет места слияния микро-операций.

По результатам тестирования были устранены многие ошибки в программном коде, переделан интерфейс и добавлен функционал, изначально не предусмотренный техническим заданием для удобства моделирования. Примером такого функционала является проверка исходного кода на ошибки.

* 1. Разработка программной документации

Программная документация является перечнем документов, составляемых до, в процессе и после создания самой программы, предоставляющих собой полную описательную информацию относительно программы, включая ее создание, функционирование и эксплуатацию. Список согласно ГОСТ 19.101-77:



Рис. 12. Список программных документов по ГОСТ 19.101-77.

Согласно заданию на выпускную квалификационную работу, содержание перечня необходимых программных документов ограничивается «Описанием применения» программы, «Руководством оператора» и «Программой и методикой испытаний» (Приложение 10).

1. Описание применения.

Данный программный документ представляет собой описание применения программы, предназначенной для моделирования процессов макро- и микро- слияния в ядре CPU. В нем представлены:

* Описание назначения программы;
* Возможности данной программы;
* Основные характеристики и ограничения, накладываемые на область применения программы;
* Требования к необходимым для данной программы техническим средствам и другим программам;
* Общие характеристики входной и выходной информации;
* Требования и условия организационного, технического и технологического характера;
* Определения задачи и методы ее решения;
* Сведения о входных и выходных данных.

2. Руководство оператора.

Данный программный документ представляет собой руководство оператора по применению и эксплуатации программы, предназначенной для моделирования процессов макро- и микро- слияния в ядре CPU. В нем представлены:

* Сведения о назначении программы и информация, достаточная для понимания функций программы и ее эксплуатации;
* Условия, необходимые для выполнения программы;
* Минимальный состав аппаратных и программных средств;
* Последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы;
* Описание функций, формата и возможных вариантов команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы;
* Ответы программы на команды оператора.

3. Программа и методика испытаний.

В данном программном документе приведена программа и методика испытаний программного изделия, предназначенного для моделирования процессов макро- и микро- слияния в ядре CPU. В нем представлены:

* Наименование, область применения и обозначение испытуемой программы;
* Цель проведения испытаний;
* Указаны требования, подлежащие проверке во время испытаний и заданные в техническом задании на программу;
* Состав программной документации, предъявляемой на испытания;
* Специальные требования, если они заданы в техническом задании на программу;
* Технические и программные средства, используемые во время испытаний;
* Порядок проведения испытаний;
* Описания используемых методов испытаний.

Заключение

В рамках выполненной выпускной квалификационной работы была достигнута цель разработки программной модели для симуляции процедуры слияния макро- и микро- операций в ядре CPU. Для достижения поставной цели была проделана следующая работа:

* Произведен обзор аналогов в сфере симуляции работы различных компонентов процессора;
* На основе анализа литературы выведены основные алгоритмы процедур слияния макро- и микро- операций, их условия и параметры, а также список моделируемых архитектур;
* Определены и обоснован выбор средств и инструментов разработки;
* Создана программная реализация модели с графическим интерфейсом, отвечающая всем поставленным требованиям.

Полученное программное обеспечение послужит удобным инструментом для последующих исследований в сфере оптимизации работы процессоров, а также для подготовки будущих специалистов с использованием наглядной демонстрации процесса на интерактивной модели.

Среди дальнейших вариантов развития программной модели можно выделить расширение поддерживаемого набора инструкций, улучшение внешнего вида графического интерфейса и расширение списка моделируемых архитектур.

Данная ВКР выполнялась коллективом из двух авторов.

В рамках данной ВКР Шаимовым Никитой Денисовичем были достигнуты следующие результаты:

* По результатам анализа существующих решений не выявлено аналога, позволяющего отследить процедуру слияния операций в ядре CPU;
* Выявлены основные принципы выполнения и составлен алгоритм процедуры макро-слияния;
* Обозначен список моделируемых архитектур и произведен анализ различий в процедуре в зависимости от архитектуры;
* Определен перечень используемых инструментов для разработки программного обеспечения;
* Разработан программный алгоритм процедуры слияния и произведена связь алгоритмов с графическим интерфейсом программы.

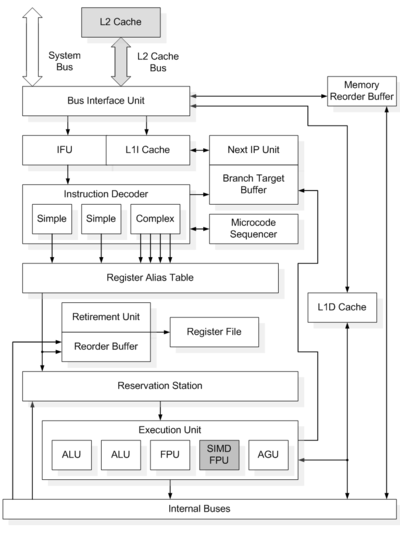
В рамках данной ВКР Степановым Никитой Андреевичем были достигнуты следующие результаты:

* Составлен перечень источников информации для анализа;
* Выявлены основные принципы выполнения и составлен алгоритм процедуры микро-слияния;
* Определены инструменты разработки графического интерфейса;
* Разработан графический интерфейс программного обеспечения;
* Произведено тестирование прототипа программного обеспечения и выявлены недостатки;
* Составлена программная документация в соответствии с единой системой программной документации.

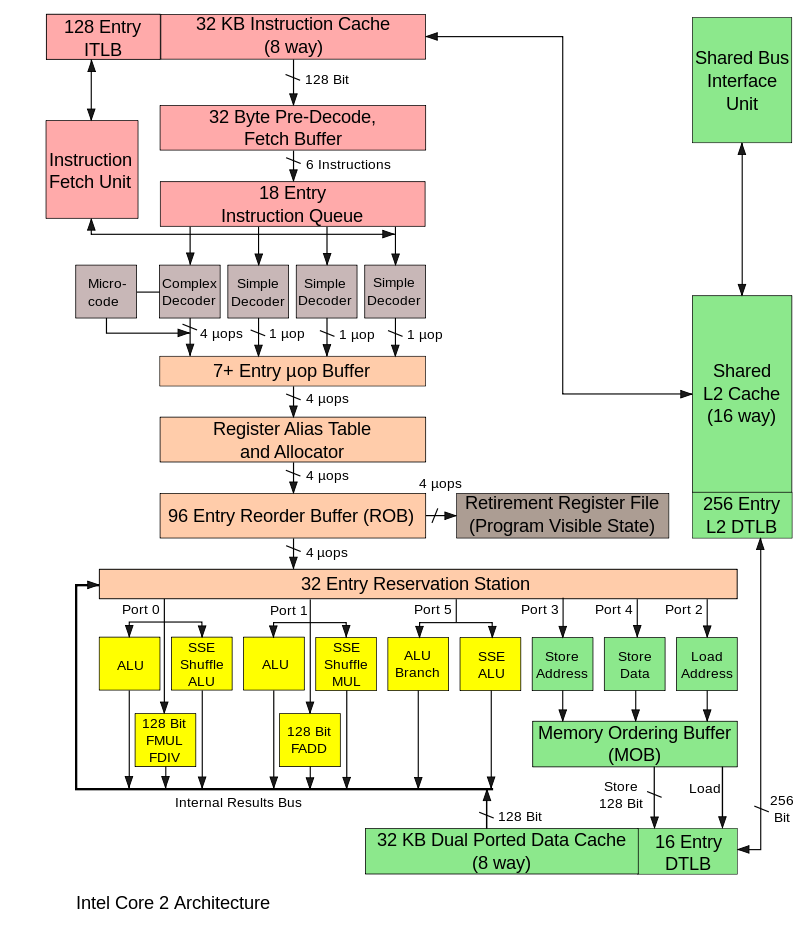
Список использованных источников

1. Туннельный эффект // Большая советская энциклопедия 3-е изд. / ред. Прохоров А.М. М.: Советская эециклопедия, 1969-1978.
2. DrMIPS – Educational MIPS simulator [Electronic resource]. URL: https://brunonova.github.io/drmips/ (accessed: 22.04.2020).
3. Mikrocodesimulator – MikroSim 2010 [Electronic resource]. URL: http://www.mikrocodesimulator.de/index\_eng.php (accessed: 22.04.2020).
4. Besim M. YASS: A system simulator for operating system and computer architecture teaching and learning. European Journal of Science and Mathematics Education Vol. 1, No. 1, 2013.
5. Intel Corporation. Intel® 64 and IA-32 Architectures Optimization Reference Manual. 2019.
6. Anandtech – Intel Core versus AMD’s K8 architecture [Electronic resource]. URL: https://www.anandtech.com/show/1998/3 (accessed: 16.04.2020).
7. Ronny R., Alexandeк P., Nathaniel H. System and method for fusing instructions: pat. US6675376B2 USA. 2004.
8. Fog A. The microarchitecture of Intel, AMD and VIA CPUs. Technical University of Denmark, 2019.
9. Fog A. Instruction tables. Technical University of Denmark, 2020.
10. Fog A. The microarchitecture of Intel, AMD and VIA CPUs. Technical University of Denmark, 2019.
11. Pentium M - Microarchitectures - Intel - WikiChip [Electronic resource]. URL: https://en.wikichip.org/wiki/intel/microarchitectures/pentium\_m (accessed: 09.04.2020).
12. Core - Microarchitectures - Intel - WikiChip [Electronic resource]. URL: https://en.wikichip.org/wiki/intel/microarchitectures/core\_(client) (accessed: 09.04.2020).
13. Nehalem - Microarchitectures - Intel - WikiChip [Electronic resource]. URL: https://en.wikichip.org/wiki/intel/microarchitectures/nehalem\_(client) (accessed: 09.04.2020).
14. Sandy Bridge (client) - Microarchitectures - Intel - WikiChip [Electronic resource]. URL: https://en.wikichip.org/wiki/intel/microarchitectures/sandy\_bridge\_(client) (accessed: 09.04.2020).
15. Ivy Bridge - Microarchitectures - Intel - WikiChip [Electronic resource]. URL: https://en.wikichip.org/wiki/intel/microarchitectures/ivy\_bridge\_(client) (accessed: 09.04.2020).
16. Tick-Tock - Intel - WikiChip [Electronic resource]. URL: https://en.wikichip.org/wiki/intel/tick-tock (accessed: 09.04.2020).
17. Haswell - Microarchitectures - Intel - WikiChip [Electronic resource]. URL: https://en.wikichip.org/wiki/intel/microarchitectures/haswell\_(client) (accessed: 09.04.2020).
18. Broadwell - Microarchitectures - Intel - WikiChip [Electronic resource]. URL: https://en.wikichip.org/wiki/intel/microarchitectures/broadwell\_(client) (accessed: 09.04.2020).
19. Skylake (client) - Microarchitectures - Intel - WikiChip [Electronic resource]. URL: https://en.wikichip.org/wiki/intel/microarchitectures/skylake\_(client) (accessed: 09.04.2020).
20. TIOBE - The Software Quality Company [Electronic resource]. URL: https://tiobe.com/tiobe-index/ (accessed: 09.04.2020).
21. Cortesi D. PyInstaller Documentation. 2020.
22. Harwani B.M. Introduction to Python programming and developing GUI applications with PyQT. Course Technology, 2012. 393 p.
23. Summerfield M. Rapid GUI programming with Python and Qt : the definitive guide to PyQt programming.
24. Николай Прохоренок. Python 3 и PyQt. Разработка приложений. СПб.: БХВ-Петербург, 2012.
25. Celio D., Dabbelt D., Patterson D. A., Asanovic K. The Renewed Case for the Reduced Instruction Set Computer: Avoiding ISA Bloat with Macro-Op Fusion for RISC-V. 2016.
26. Taram M., Venkat A., Tullsen D.M. Mobilizing the Micro-Ops: Exploiting Context Sensitive Decoding for Security and Energy Efficiency.
27. Антошина И.В., Котов Ю.Т. Микропроцессоры и микропроцессорные системы. М., 2005.
28. Таненбаум Э. Архитектура компьютера 5-е изд. СПб: Питер, 2007.
29. Моисеев И.А. Процессоры современных вычислительных систем. СПб., 2019.

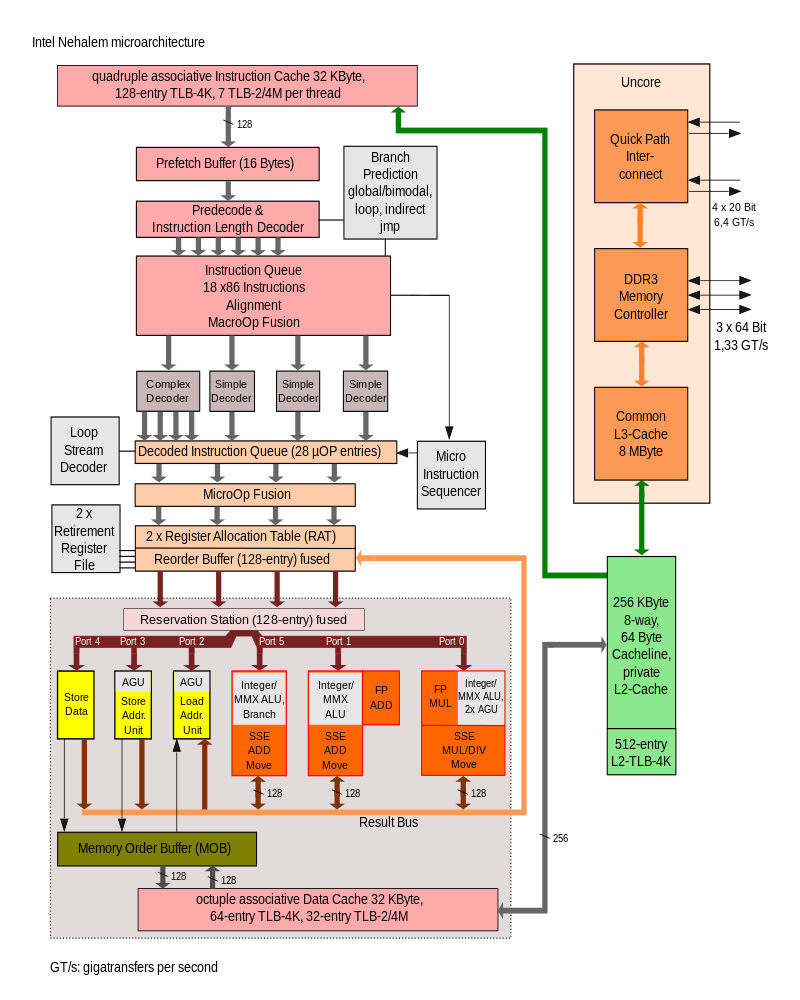
Приложение 1. Схема архитектуры Pentium III.



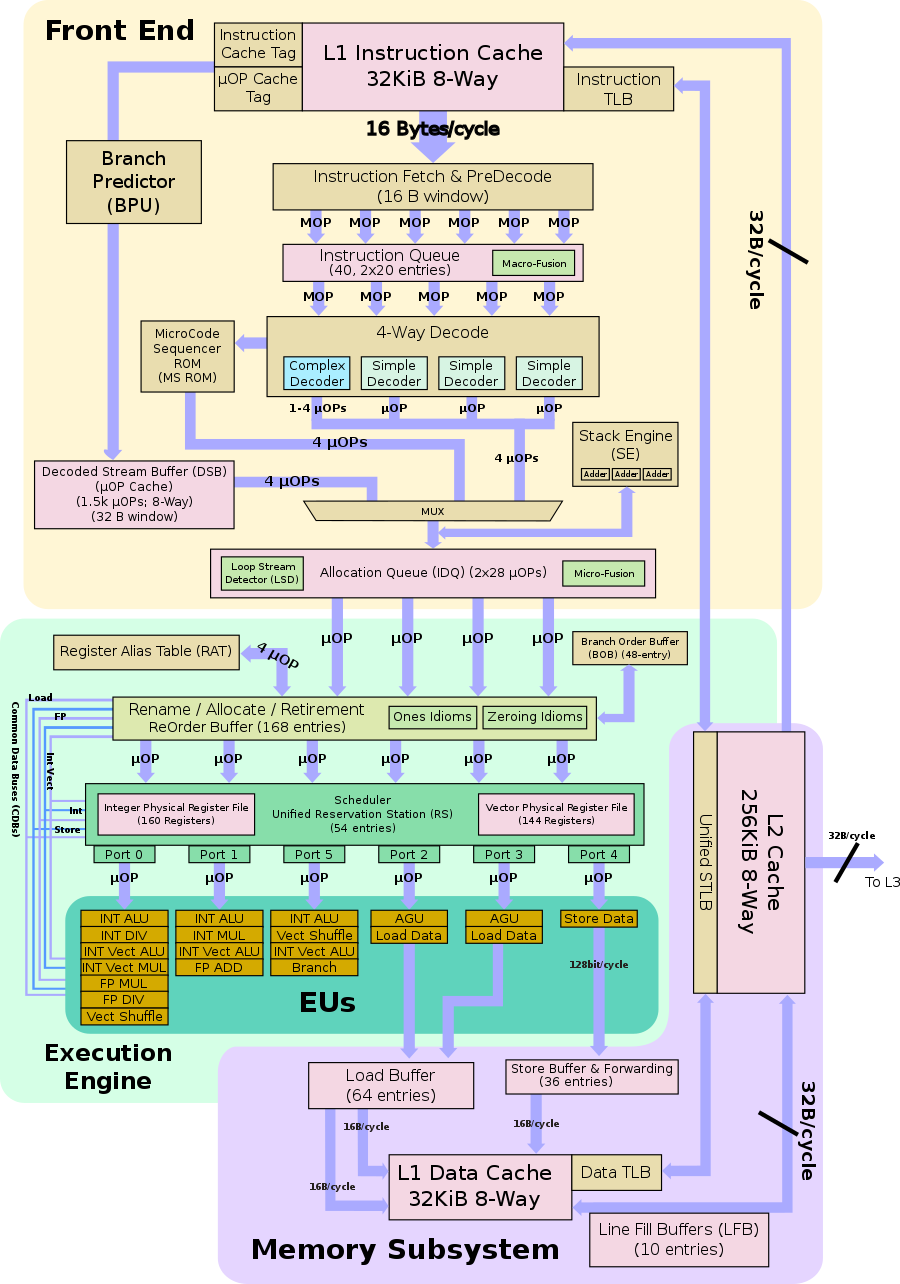
Приложение 2. Схема архитектуры Core 2.



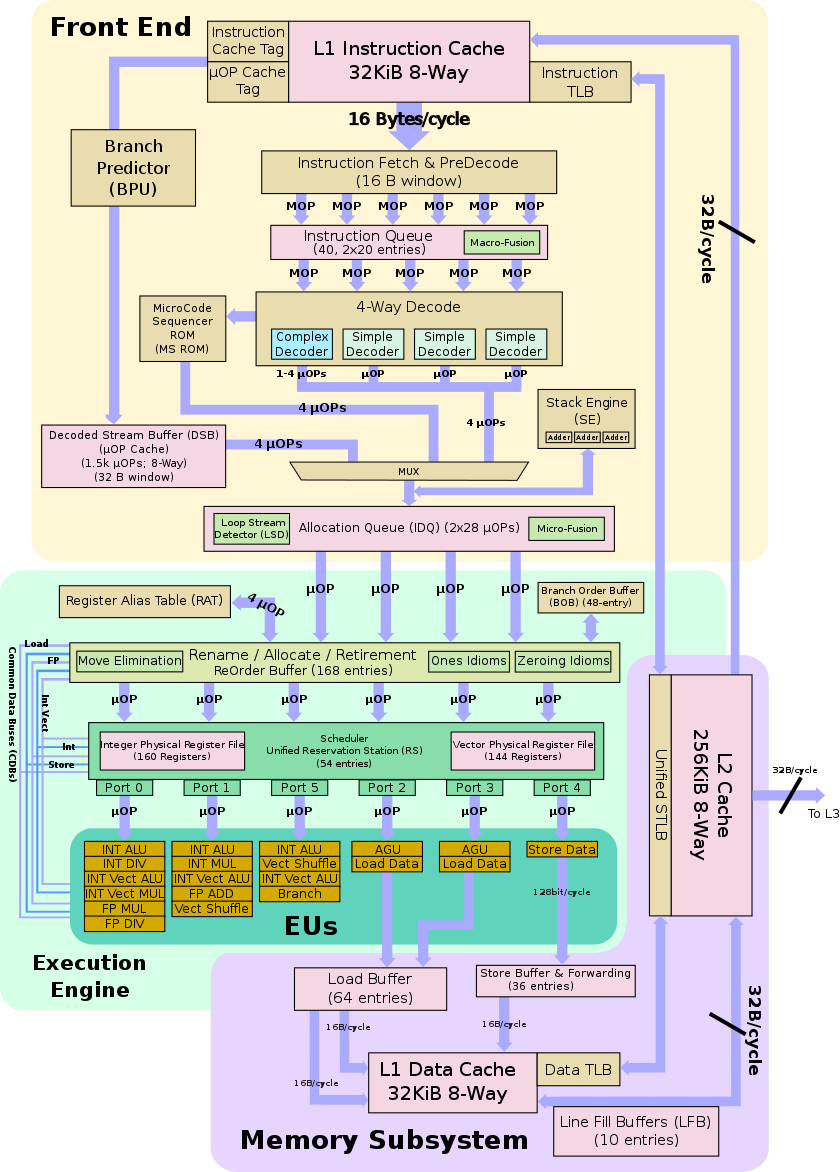
Приложение 3. Схема архитектуры Nehalem.



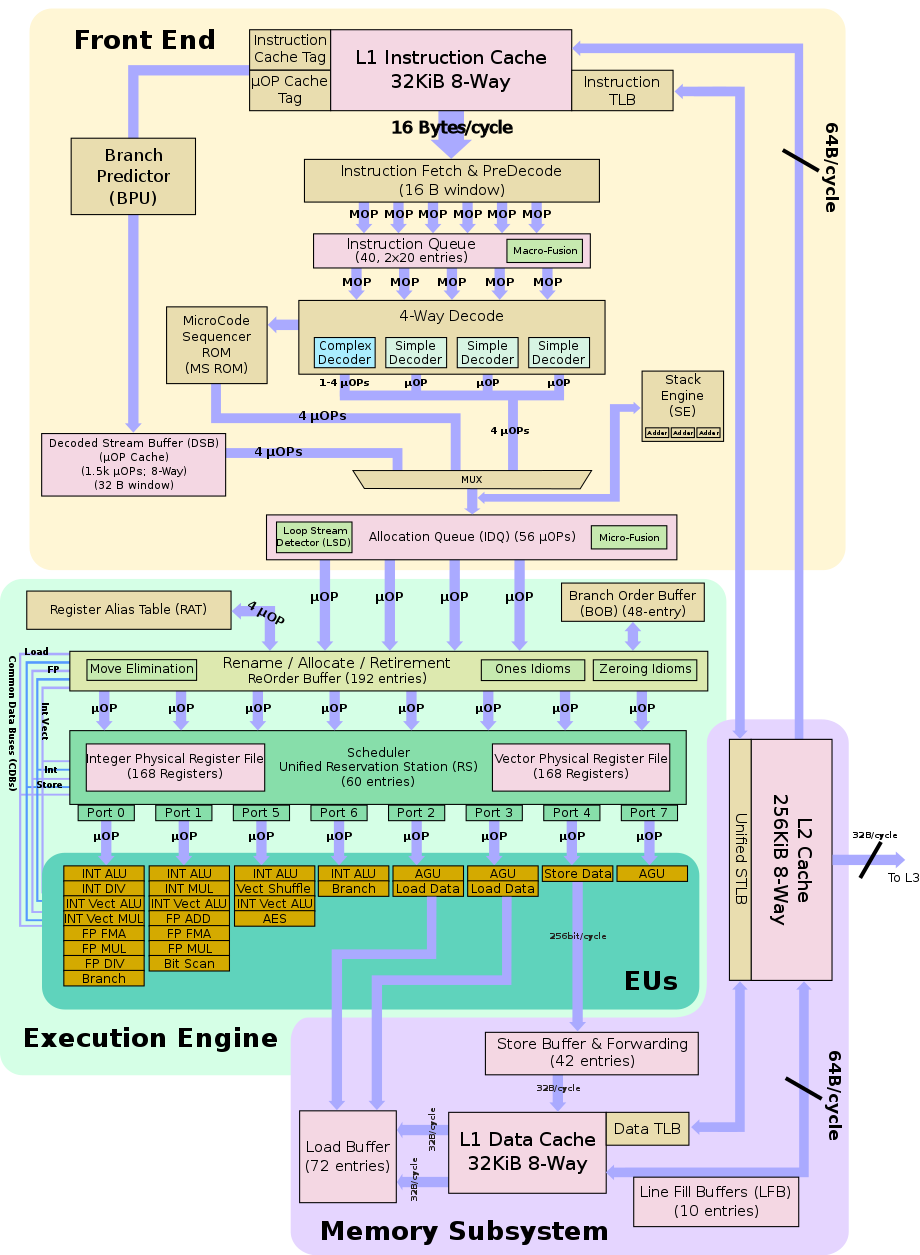
Приложение 4. Схема архитектуры Sandy Bridge.



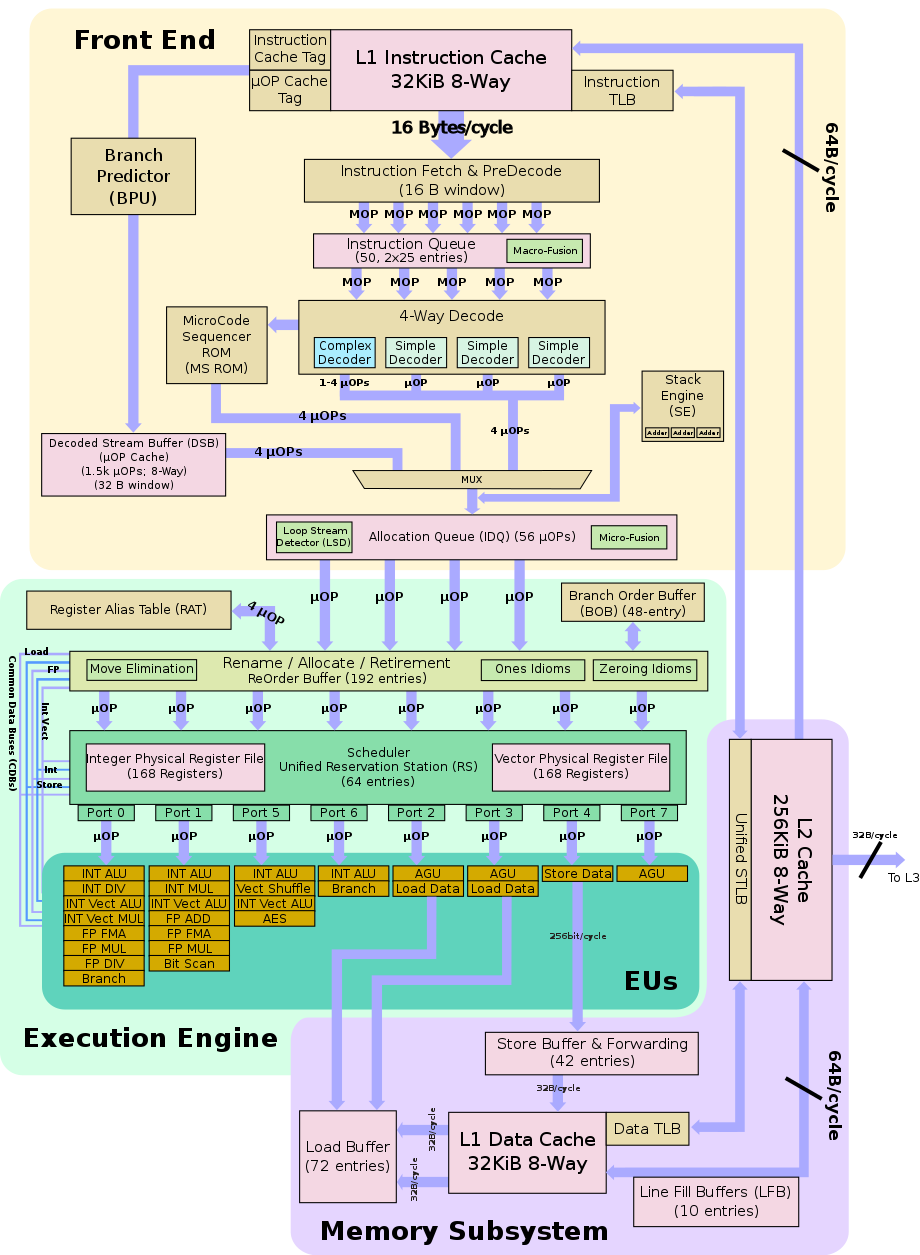
Приложение 5. Схема архитектуры Ivy Bridge.



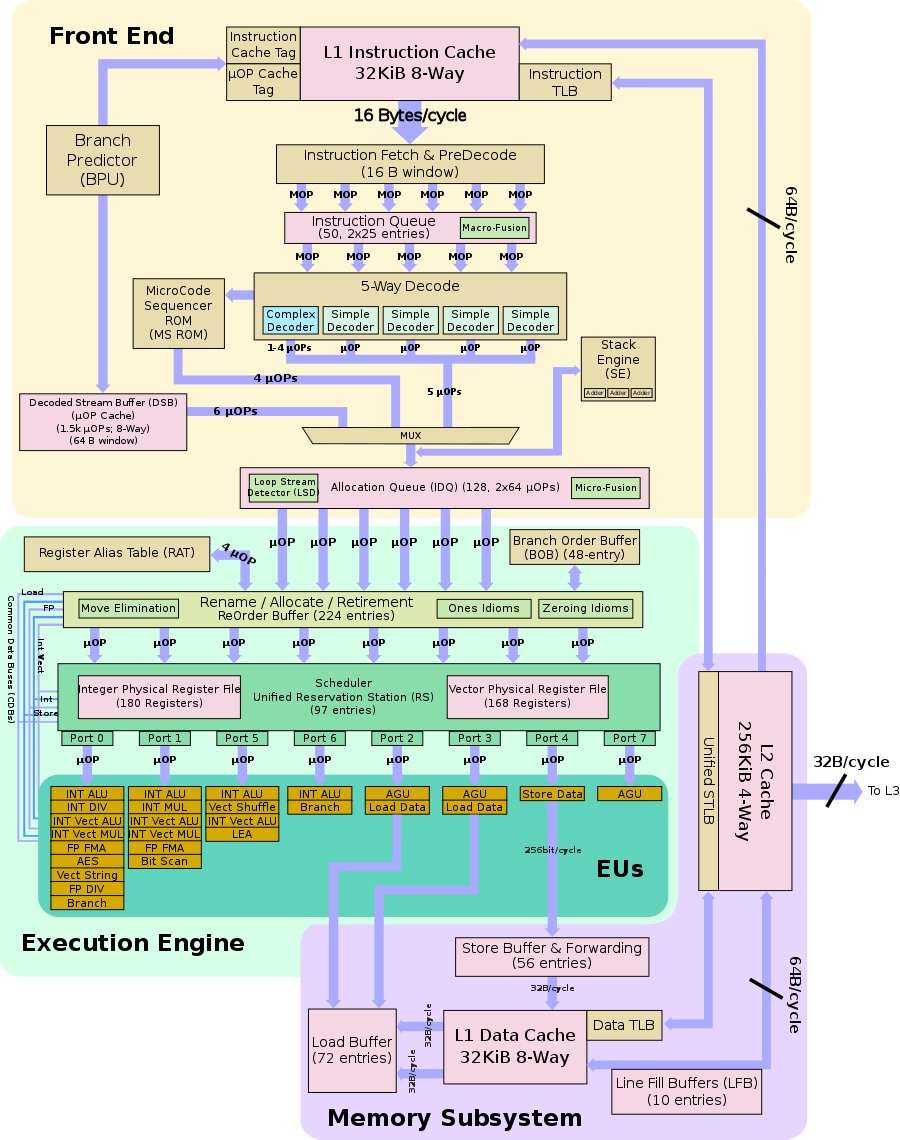
Приложение 6. Схема архитектуры Haswell.



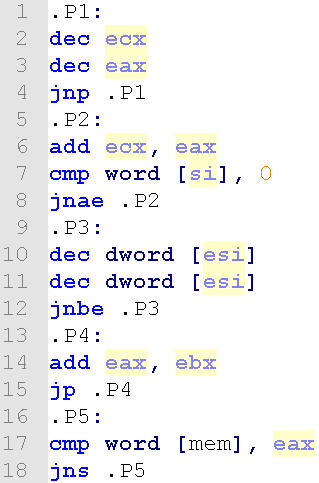
Приложение 7. Схема архитектуры Broadwell.

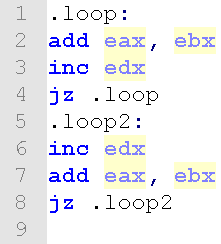


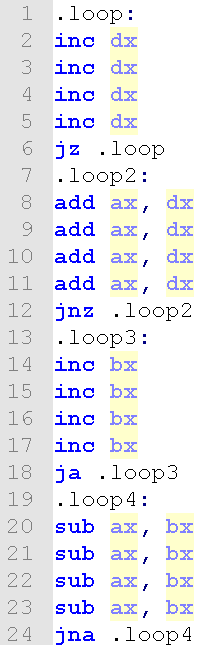
Приложение 8. Схема архитектуры Skylake.

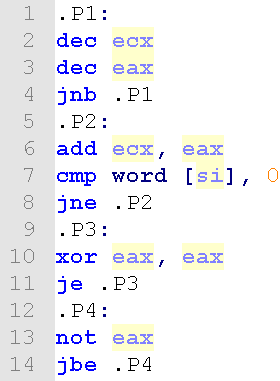


Приложение 9. Примеры готовых вариантов кода.









Приложение 10. Программная документация.

