# **RISC-V: стандарт, изменивший мир микропроцессоров**

Микропроцессоры, используемые сегодня, далеко ушли от систем, применяемых еще 30 лет назад: увеличились частота, объем памяти, количество ядер и пр. Но главное в другом: процессоры становятся все сложнее, что затрудняет их дальнейшее развитие, поэтому сегодня ключевыми факторами, влияющими на решение о создании новых систем, стали стоимость и целесообразность разработки. Как запланировать развитие линейки процессоров на долгосрочную перспективу? Какую роль сегодня играет система команд процессора? В этой связи весьма перспективна архитектура RISC-V — новый открытый стандарт системы команд и архитектуры процессоров.

Компьютерные архитектуры стремительно прогрессируют, однако работа с современными системами принципиально не ускорилась — чаще всего их используют для решения примерно тех же задач, что и 30 лет назад. В этой области человечество топчется на месте, и одна из причин этого — низкая фактическая эффективность использования оборудования: в среднестатистическом коде на С++ применяется не более 10% возможностей современных микропроцессоров. Расстояние между программистом и аппаратурой сегодня намного больше, чем раньше, — до сих пор нет эффективного интерфейса между разработчиком и оборудованием, для которого создается программа. Основными недостатками являются следующие:

* алгоритмическое описание программы на языке программирования высокого уровня транслируется в ассемблер с применением массы специфических оптимизаций, которые трудно контролировать без знания постоянно меняющихся особенностей компилятора;
* отсутствует возможность управления работой подсистемы памяти (кэшированием), что особенно важно при увеличении числа ядер и организации эффективного взаимодействия между потоками;
* ассемблер не позволяет получить представление о том, насколько оптимизирован код, даже несмотря на то, что он напрямую транслируется в команды конкретного микропроцессора.

Помимо архитектурного уровня (системы команд, доступной компилятору и программисту), имеется уровень микроархитектуры, отвечающий за то, как на самом деле команды выполняются внутри процессора и сколько команд удается фактически обработать за такт. Таким образом, система команд — это не что иное, как интерфейс, а конкретное поколение процессоров (AMD Zen2, Intel Sky Lake и пр.) — его реализация. Когда интерфейс перестает удовлетворять необходимым требованиям, программист его меняет. Правда, если интерфейс используют другие разработчики, то задача усложняется, но решается: старый интерфейс, как правило, может быть реализован через новый, и можно сохранить поддержку для пользователей старого интерфейса. Однако для аппаратуры ситуация совсем иная.

**Проблема 1: избыточность системы команд.**

Избыточность означает дополнительный расход ресурсов кристалла (транзисторы, частота, тепловыделение) и рост сложности — например, поддержка обратной совместимости означает, что с каждым новым поколением процессора базовый интерфейс расширяется, поэтому проблемы растут как снежный ком.

**Проблема 2: отсутствие кроссплатформенности.**

Высокопроизводительные программные компоненты (библиотеки) сегодня сильно зависят от конкретной аппаратуры. Производители процессоров намеренно выпускают открытое ПО, напичканное максимально возможным количеством инструкций для своего микропроцессора. Например, Intel предоставляет бесплатную библиотеку (в частности, трассировки лучей Embree), испещренную intrinsic-функциями (функциями, вызовы которых заменяются компилятором на высокоэффективный код конкретного микропроцессора). Стоимость переноса подобной библиотеки на кристалл с другой системой команд сравнима со стоимостью разработки ее аналога с нуля.

**Проблема 3: компилятор.**

Разработчики микропроцессоров сегодня активно вносят свой вклад в развитие компиляторов и ОС, особенно gcc и ядра Linux — зависимого от производителя аппаратуры системного ПО. Нельзя ожидать, что один и тот же код можно одинаково хорошо оптимизировать для работы с разными процессорами, даже если они имеют практически идентичную функциональность (LLVM лишь отчасти амортизирует эту проблему).

**Проблема 4: безопасность.**

В реальности никто не гарантирует безопасности (например, существуют группы аппаратных уязвимостей Spectre и Meltdown). Хотите быть защищенными — делайте целиком свой компьютер плюс ОС, компилятор и драйверы с нуля.

**Проблема 5: эффективное взаимодействие потоков.**

В многопоточной программе вопрос оптимальной организации потоков в большинстве систем команд не решен на должном уровне, но с увеличением числа процессорных ядер эта проблема становится особенно важной.

Каждый производитель микропроцессоров обычно «тянет одеяло на себя», часто намеренно добавляя в свои изделия «удобные» инструкции, и разработчики ПО уже привыкли к этому «удобству», хотя и догадываются, к чему приведет перенос кода на процессор с другой системой команд. В связи с этим можно упомянуть графические процессоры, для которых неактуальна обратная совместимость, и, как следствие, их прогресс в развитии GPU был весьма значительным. Проблемы переносимости ПО решаются здесь на уровне графических и вычислительных API (Vulkan, OpenCL), и это обеспечивает большую гибкость, нежели использование системы команд. Сейчас производители GPU смогли договориться о едином открытом стандарте Vulkan для работы почти со всеми современными графическими процессорами. В результате программные системы и алгоритмы, интенсивно использующие GPU, в целом более интероперабельны, чем их CPU-аналоги.

**Процессоры и их команды**

Компании Intel и AMD до сих пор успешно работают на рынке, что во многом объясняется обратной совместимостью их микропроцессоров, а также активной оптимизацией технологий производства [**[1]**](https://www.osp.ru/os/2020/02/13055471#1), однако дизайн набора инструкций нельзя назвать сильной стороной этих производителей. Об этом говорит хотя бы тот факт, что инструкции в их процессорах архитектуры x86/x64 уже давно транслируются в некое простое RISC-подобное представление. Основная проблема x86/x64 — это крайне раздутая (больше 2 тыс.) и слабоструктурированная система команд. Многие команды уже давно не используются и не поддерживаются, но продолжают оставаться на случай обратной совместимости.

В архитектуре x86/x64 команды могут занимать от 1 до 15 байт. При этом более короткие операции со временем стали использоваться реже. Изначальная идея была в том, чтобы кодировать наиболее частые операции меньшим числом байтов (коды Хаффмана), но со временем у нее обнаружилась обратная сторона. Например, шесть коротких восьмибитных инструкций предназначены для работы с числами в десятичном представлении, что уже давно не поддерживается компиляторами gcc [**[1]**](https://www.osp.ru/os/2020/02/13055471#1) и не реализуется в современных процессорах, но эти инструкции по-прежнему занимают место в системе кодирования команд. Аналогичная ситуация — и в сопроцессоре x87.

В x86/x64 больше сотни регистров, что одновременно и много, и мало. Много — из-за требования обратной совместимости x64 с x86 и его сегментными регистрами, x87-стеком FPU и другими неиспользуемыми сущностями. Мало, потому что полезных регистров лишь 16 (в x86 — всего 8). При этом новые регистры все время «нарастают» поверх старых: 32-разрядные процессоры должны были уметь выполнять унаследованный бинарный 16-битный код, а 64-разрядные — 32-битный.

Сегодня в системах x86/x64 аккумулировано существенно больше недостатков, чем в других системах, поэтому неудивительно, что в области встроенных систем, где важна энергоэффективность, эта архитектура проигрывает архитектуре ARM, однако и здесь проблем достаточно:

* в ARMv7 отсутствует поддержка 64-битного адресного пространства;
* в ARMv7 имеется три системы команд: обычный режим и два сокращенных (Thumb и Thumb 2), — поэтому декодеры команд должны понимать все три системы, что увеличивает энергопотребление, задержку в конвейере и стоимость проектирования;
* в ARMv7 очень большая система команд, включающая в себя более 600 инструкций даже без плавающей точки, и ее трудно назвать классическим RISC-набором: например, программный счетчик — это один из адресуемых регистров, а значит, почти любая инструкция может изменить поток управления (выполнить переход).

В 2011 году была анонсирована полностью переработанная система команд ARMv8 с 64-разрядными адресами и расширенным набором целочисленных регистров. Были удалены функции ARMv7, усложняющие реализацию (счетчик команд больше не является частью набора целочисленных регистров); нет предикатных инструкций; удалены комплексные инструкции с множественной загрузкой и сохранением, а кодирование команд в целом упрощено. Однако были решены не все проблемы, а кроме того, добавились новые.

В целом ARMv8 оказалась коммерческой системой команд, напичканной самой разной функциональностью в еще большей степени, чем многие другие архитектуры: более 1000 инструкций в 53 форматах. При этом были упущены некоторые важные вещи: например, отсутствует какой-либо аналог Thumb (что важно для многих встроенных систем в силу ограниченной в объеме памяти для кода), нет объединенных инструкций «сравнение + переход» и др.

Проблема архитектуры MIPS — недостаточная гибкость системы команд. Если x64 и ARM зачастую содержат слишком высокоуровневые инструкции, то в MIPS перекос идет в другую сторону: команды опускаются до уровня микроархитектуры, что предполагает строго определенную реализацию в аппаратуре.

Отличительная черта SPARC — регистровые окна, и на первый взгляд это выглядит неплохо: предоставляется возможность расширения фактического числа регистров при помощи приема «база + смещение» (указывается R1, а фактически подразумевается SP + R1 — кусок стека сохраняется на регистрах). При вызове функций не нужно ничего перемещать, и компилятор становится проще. Однако регистровые окна дороги в реализации. Огромное количество регистров, находящихся «в стеке», не используется (сверхбыстрая память регистров дороже, чем L1-память, из-за многопортовости). Например, в системе «Эльбрус» регистровый файл содержит 20 портов, что очень много [**[2]**](https://www.osp.ru/os/2020/02/13055471#1). Кроме того, когда регистровая память заканчивается, «наступает апокалипсис» в операционной системе [**[1]**](https://www.osp.ru/os/2020/02/13055471#1). В итоге исследователи сошлись во мнении, что если в микропроцессоре имеются лишние транзисторы, то лучше сделать больше регистров и предоставить компилятору самому заниматься встраиванием функций, а не перекладывать эту задачу на уровень аппаратуры.

Систему команд Power нельзя назвать простой: чуть меньше 1000 инструкций в 25 форматах (включая векторные инструкции), — но она проще, чем ARM. Кроме того, особенность Power — модульность, а каждый регистр относится к функциональному классу и большинство инструкций внутри класса используют только принадлежащие этому классу регистры. Только несколько инструкций передают данные между разными функциональными классами: условным, с фиксированной точкой, с плавающей точкой и векторным. Таким образом, конкретная реализация может не поддерживать, например, операции с плавающей точкой (что важно для некоторых встроенных решений). Такое разделение делает прозрачной для компилятора информацию о зависимости между регистрами.

Так же как в ARM и SPARC, в Power есть специальные регистры, в которых выставляются 4 бита состояния после выполнения операций сравнения. Но в Power их восемь копий, каждая из которых может быть результатом инструкции сравнения, и каждая копия может быть источником ветвления. Такая избыточность применяется, для того чтобы инструкции могли без конфликта использовать разные условные состояния (когда в разных ветвях кода получаются разные состояния для последующих ветвлений). Кроме того, можно выполнять логические операции с этими четырехразрядными регистрами, что позволяет одной ветви проверять более сложные условия.

Таким образом, Power обладает рядом особенностей, позволивших системам на их основе занимать лидирующие позиции в рейтинге топ-500 и на рынке микроконтроллеров, главным образом в автомобильной и авиационной отраслях [**[5]**](https://www.osp.ru/os/2020/02/13055471#1). Однако разнообразие сфер применения Power привело к тому, что одной из основных проблем этого микропроцессора стала не система команд, а экосистема: узкоспециализированное применение не дает сформироваться широкому сообществу разработчиков. Правда, сегодня ситуация меняется: появились доступные персональные компьютеры на Power9 с открытыми «прошивками» (firmware) загрузчика и сопутствующих компонентов. Открытая система команд появилась в реализации FPGA с открытым исходным кодом.

**RISC-V**

Систему команд RISC-V c 2010 года разрабатывают в Университете Беркли (Калифорния, США), где решили, что необходим новый единый и открытый набор команд для всех типов систем — от микроконтроллеров до высокопроизводительных систем. К 2010 году в мире существовало огромное количество стандартов для интерфейсов в области программного обеспечения (например, OpenGL), но грамотного стандарта для системы команд CPU как программно-аппаратного интерфейса тогда не было (существовавшие RISC-I, RISC-II и OpenRISC не обладали всеми необходимыми возможностями с учетом расширения на будущее). Кроме того, закрытость популярных систем команд (а на тот момент почти все были закрытыми, кроме SPARC) создала коммерческие предпосылки для развития открытого стандарта. Например, ARM по условиям лицензии требует от 1 млн долл. лишь за право легального использования своей системы команд, запрещая при этом какие-либо изменения или расширения.

В действительности за перфекционистским лозунгом RISC-V сделать одну систему команд для всех стоит ряд практических и конкретных целей:

* **Повышение энергоэффективности и производительности.** Наличие минимального интерфейса позволяет разработчикам процессорных ядер для получения полноценно работающей системы реализовывать лишь относительно небольшой набор инструкций. Это дает возможность уменьшить количество транзисторов, удешевить разработку и снизить расходы на поддержку процессорных ядер.
* **Модульность и расширяемость.**Минимальных наборов команд сегодня достаточно много, и RISC-V далеко не самый компактный, однако основная проблема в том, что «минимальный» набор команд не означает «универсальный». RISC-V призван решить эту проблему за счет модульной структуры системы команд.
* **Удешевление разработки.** Единая открытая экосистема позволяет сократить расходы на разработку компиляторов, ОС, драйверов и периферии. Наличие в RISC-V открытых репозиториев, в которых многое уже реализовано и вследствие многолетней поддержки сообщества доведено до высокого качества, позволяет существенно уменьшить затраты.
* **Обратная совместимость.**Унаследованные программы, разработанные для старых процессоров, не поддерживающих новые расширения, должны работать без перекомпиляции на новых процессорах аналогичной архитектуры и с той же системой команд, но с множеством расширений. Кроме того, требуется совместимость библиотек на бинарном уровне, поскольку скомпилированная статическая библиотека для одной ОС и конкретного процессора может быть использована и на другой ОС с другим процессором благодаря совместимости на уровне компилятора и собственно процессора. RISC-V призван обеспечить обратную совместимость: базовый набор и основные расширения стандартизованы, что существенно упрощает, например, программную эмуляцию функциональности, которая не поддерживается аппаратурой.
* **Безопасность.** Критически важное ПО обычно проходит сертификацию и верификацию, стоимость которых может многократно превышать затраты на его разработку. Однако, даже если доказана корректность ядра ОС и выполнена процедура сертификации, сменить аппаратуру уже не представляется возможным. По этой причине система команд Power на сегодняшний день прочно обосновалась в области гражданской авиации. Помимо этого, если, например, в некотором процессоре обнаружена уязвимость, то заменить его на другой с аналогичной архитектурой бывает сложно, а в случае RISC-V можно использовать другой процессор (пусть и не такой эффективный), но без закладок. При этом на нем можно запустить унаследованное ПО с минимальными затратами усилий и времени. Взаимозаменяемость процессоров позволяет RISC-V обеспечить безопасность.

**Базовый набор инструкций**

Система команд RISC-V имеет ряд отличительных особенностей.

**Отсутствие неявных внутренних состояний.** Результат любой операции (кроме команд перехода) всегда помещается в регистр общего назначения, так как в RISC-V нет, например, флагов состояний, которые устанавливает инструкция cmp в x86. Вместо этого результат команды сравнения помещается в один из регистров общего назначения. Это существенно упрощает суперскалярную реализацию с внеочередным выполнением команд, но при этом не вносит существенных накладных расходов для простых реализаций.

**Отсутствие предикатных инструкций.**Предикатные инструкции необходимы для VLIW-процессоров (например, «Эльбруса») и для векторной обработки данных, однако в скалярном коде выигрыш от них обычно небольшой.

**Компактный базовый набор инструкций.** В базовом наборе всего 11 базовых арифметических инструкций (большинство из них могут встречаться в двух формах: R-формат (стандартный) и I-формат (второй операнд считывается непосредственно из самой инструкции, давая в сумме 21 инструкцию)), 10 инструкций для обращения в память и 8 инструкций для переходов. Итого — 39 инструкций.

**32 регистра общего назначения.**Это вдвое больше, чем у RISC-аналогов.

**Поддержка ослабленной модели памяти (Relaxed Memory Model) в базовом наборе инструкций.** Такая модель помогает программисту явно указывать ход выполнения кода, запрещая, например, запуск дорогостоящей операции синхронизации данных через кэш L2 и L3 при записи в ячейку памяти.

**Наличие спаренных (fused) операций «сравнение + переход», уменьшающих размер программного кода.**

**Инструкции ускорения вызова функций и выполнения операторов.**

Таким образом, компактный набор (менее 40 инструкций) содержит внушительный объем полезной функциональности.

**Плавающая точка**

Расширение RV32F для работы с вещественными числами добавляет 32 новых регистра. Здесь обращает на себя внимание наличие инструкций преобразования в целые числа и обратно, а также инструкций непосредственного перемещения данных из целочисленного регистра в регистр с плавающей точкой, что является проблемой для многих существующих систем команд, когда данные приходится перемещать через память.

**Векторное расширение**

Расширение RV32V предназначено для реализации векторных операций и имеет ряд ключевых особенностей.

В наборе инструкций длина вектора не фиксируется, а задается в программе специальной инструкцией, что принципиально отличается от существующих подходов и позволяет на процессорах с небольшой длиной вектора выполнять программы, скомпилированные под более широкий вектор, хотя и с меньшей производительностью.

В RISC-V имеется предикатное выполнение, то есть во всех векторных операциях используются маски, что хорошо при выполнении программы как на обычных CPU, так и на массово-параллельных системах (OpenCL и аналоги). При этом векторные регистры существуют отдельно от остальных регистров, а перемещение данных между скаляром и вектором поддерживается аппаратно. Благодаря этому в RISC-V нет проблем при работе смешанного скалярно-векторного кода, как это наблюдается, например, в x64 (и особенно в x86), где рекомендуется явно использовать векторный тип (\_\_m128) вместо float или int для уверенности в том, что данные лишний раз не выгружаются в память из векторных регистров.

В целом можно сказать, что векторное расширение спроектировано в RISC-V с учетом опыта многих архитектур (в частности, Cray) и выполнено достаточно грамотно.

**RISC-V сегодня**

Рост популярности RISC-V с момента его появления в 2010 году (используется в более чем 500 компаниях из 28 стран) свидетельствует: цели, поставленные при разработке этого стандарта, достигнуты. Ни одна другая система команд, тем более разработанная в университете, еще не добивалась такого успеха за 10 лет. Несмотря на то что рядовые пользователи не сразу почувствуют эффект от использования RISC-V, изменения, которые этот стандарт внес в экосистему и мировоззрение разработчиков электроники, уже сейчас необратимы. Мобильные телефоны, ПК, микроконтроллеры, другие встроенные системы и, наконец, суперкомпьютеры — все эти устройства могут использовать RISC-V в качестве основы программно-аппаратного интерфейса.

Применение таких открытых технологий, как RISC-V, играет ключевую роль при решении проблемы обеспечения импортонезависимости, поскольку отказ от использования и разработки закрытых решений позволит как сохранить инвестиции, так и обеспечить работоспособность критически важных систем независимо от политической конъюнктуры. При наличии множества производителей оборудования, поддерживающих стандарт RISC-V, теряет смысл установка любых закладок в процессоры — такой процессор всегда можно заменить на другой. Кроме того, сложность современных технологий почти всегда означает высокую стоимость их поддержки, поэтому так важно снижать стоимость разработки.

В сообщество RISC-V входят многие ведущие компании [**[3]**](https://www.osp.ru/os/2020/02/13055471#1), включая Nvidia (применение RISC-V и языка Ада для создания беспилотных автомобилей) и Alibaba (IP-блок для искусственного интеллекта). Компания AMD использует RISC-V в новом поколении графических процессоров для анализа графов, обеспечивающего создание и обработку очереди команд. Кроме того, RISC-V получает поддержку на уровне государств и федеральных структур: Индия объявила RISC-V национальным стандартом, DARPA требует использовать RISC-V в качестве обязательного компонента в ряде финансируемых этой организацией программ (в том числе по всему спектру работ по безопасности на уровне оборудования), Управление по инновациям Израиля создает общую платформу GenPro на базе RISC-V, Китай объявил широкую программу субсидирования решений на базе RISC-V, а Евросоюз обсуждает такую возможность. Основная причина — доступный набор инструментов и открытая архитектура. Наконец, RISC-V получает признание и в России, где обосновались уже две компании: Syntacore (существующая практически с момента основания RISC-V в 2010 году) и CloudBear [**[4]**](https://www.osp.ru/os/2020/02/13055471#1).

Вокруг RISC-V уже сложилась экосистема открытого и коммерческого ПО (GCC, Linux, BSD, LLVM, QEMU, FreeRTOS, ZephyrOS, LiteOS, SylixOS, Lauterbach, Segger, Micrium, ExpressLogic и др.), а также открытых и коммерческих процессорных ядер (Rocket, BOOM, RI5CY, Ariane, PicoRV32, Piccolo, Syntacore SCR1, Hummingbird, Codasip, Cortus, C-Sky, Nuclei, SiFive, Syntacore и др.).

**Недостатки**

Несмотря на ряд преимуществ RISC-V по сравнению с аналогами, у этой архитектуры имеется ряд недочетов.

Деятельность любой экосистемы по стандартизации имеет два аспекта. Один аспект — открытость, совместимость и легкость вхождения в индустрию для небольших компаний. Другой — ограничение гибкости: предложить свое расширение системы команд позволяется лишь крупным спонсорам, и даже для них этот процесс непростой. Например, проект Libre-RISCV разработки открытого GPU стал использовать архитектуру Power по причине невозможности получить от сообщества необходимую документацию. Разработчики Libre-RISCV предлагали векторное расширение (Simple-V), которое, по-видимому, было неинтересно сообществу RISC-V из-за наличия собственного векторного аналога.

Когда речь идет о высокой производительности, то практически из всех материалов по RISC-V следует вывод о необходимости сильного смещения в сторону суперскалярных ядер с внеочередным выполнением команд. С одной стороны, это ожидаемо, поскольку такой способ повышения производительности соответствует идеологии RISC, однако, с другой стороны, такой подход выглядит как лоббирование определенной технологии с целью продвижения собственных разработок. Кроме того, сообщество RISC-V умалчивает о графических процессорах. По сравнению, например, с Power, экосистема RISC-V выглядит хотя и более целостной, но менее зрелой, поскольку реальных реализаций на кристалле у RISC-V еще меньше.

По большому счету RISC-V пока нельзя считать универсальным средством для решения проблем стандартизации всей микроэлектроники, поскольку в текущей реализации эта архитектура не может быть основой, например, для VLIW-процессоров, в связи с тем что VLIW в принципе не предполагает наличия какого-либо абстрактного интерфейса: наоборот, система команд делается с учетом конкретной реализации процессора. Здесь следует упомянуть наиболее известные отечественные процессоры «Эльбрус» и NMC4, которые как раз используют VLIW. С технической точки зрения это хорошие решения, особенно когда необходимо добиться высокой производительности. Например, современные микропроцессоры «Эльбрус» лидируют по числу команд за такт (до 50), опережая на многих задачах новинки от Intel и AMD. Но эти решения дороги, по большей части закрыты, и даже внутри страны их можно использовать лишь ограниченно. Например, отечественные создатели ПО для гражданской авиации [**[5]**](https://www.osp.ru/os/2020/02/13055471#1) не хотят работать с закрытой и специфической экосистемой «Эльбруса». Кроме того, для авиации необходимы процедуры сертификации, где недопустимы закрытые решения. Как следствие, в стране эксплуатируются экспортные системы, поставляемые с сертифицированной зарубежной электроникой.

\*\*\*

Современные электронные системы настолько усложнились, что следование открытым стандартам стало критически важным для жизнеспособности и экономической целесообразности любого проекта разработки микропроцессора. Даже при острой необходимости использования какой-либо специфической аппаратной функциональности почти нет веских причин для того, чтобы не рассматривать открытые стандарты в качестве базовых. Те, кто не будут двигаться в этом направлении, постепенно вымрут из-за тяжести и дороговизны разработки и поддержки собственных решений с уникальной экосистемой, как бы хорошо это ни было для них в краткосрочной перспективе. В этой связи RISC-V выглядит весьма привлекательно: грамотно спроектированный стандарт, который позволяет решать проблемы совместимости (в том числе обратной в долгосрочной перспективе), безопасности, сертификации, энергопотребления, эффективной реализации многопоточности и удешевления разработки.

Стандарт RISC-V необратимо изменил мир микроэлектроники, и хотя эти изменения заметили еще не все, их последствия будут видны уже в ближайшее время. Вместе с тем не стоит принудительно переводить на RISC-V текущие проекты или уже работающие решения, однако благодаря этому стандарту многие вопросы микропроцессорной индустрии можно будет решать на другом уровне.

**Литература**

1. Waterman A. S. Design of the RISC-V instruction set architecture // Electrical Eng. and Comput. Sciences. Univ. California at Berkeley, Tech. Rep. No. UCB/EECS-2016–12016, 2016. URL: [**https://people.eecs.berkeley.edu/~krste/papers/EECS-2016-1.pdf**](https://people.eecs.berkeley.edu/~krste/papers/EECS-2016-1.pdf) (дата обращения: 31.03.2020).

2. А.К. Ким, В.И. Перекатов, С.Г. Ермаков. Микропроцессоры и вычислительные комплексы семейства «Эльбрус». СПб.: Питер, 2013. — 272 с.

3. Krste Asanovic. RISC-V. Past, Present, Future // Технический симпозиум RISC-V (Москва, 2019). URL: [**https://syntacore.com/media/riscv\_moscow\_2019/RISC-V%20Foundation%20State%20of%20the%20Union\_Krste.pdf**](https://syntacore.com/media/riscv_moscow_2019/RISC-V%20Foundation%20State%20of%20the%20Union_Krste.pdf)(дата обращения: 10.04.2020).

4. Krste Asanovic, Alexander Redkin и др. // Технический симпозиум RISC-V (Москва, 2019). URL: https://riscv.expert (дата обращения: 10.04.2020).

5. Борис Барладян, Алексей Волобой, Владимир Галактионов, Лев Шапиро. OpenGL для критически важных систем // Открытые системы. СУБД. — 2020. — № 1. — С. 22–24. URL: [**https://www.osp.ru/os/2020/01/13055346/**](https://www.osp.ru/os/2020/01/13055346/) (дата обращения: 20.05.2020).

***Владимир Фролов***([**v**](mailto:vova@frolov.pp.ru)[**ova@frolov.pp.ru**](mailto:ova@frolov.pp.ru)), ***Владимир Галактионов***([**vlgal@gin.keldysh.ru**](mailto:vlgal@gin.keldysh.ru))  —  научные сотрудники, ИПМ им. М.В. Келдыша РАН, ***Вадим Санжаров***([**vadim.sanzharov@graphics.cs.msu.ru**](mailto:vadim.sanzharov@graphics.cs.msu.ru))  —  научный сотрудник, ВМиК МГУ им. М.В. Ломоносова (Москва).

DOI: 10.26295/OS.2020.53.12.002