TL;DR;

Ниже приведены попытки описать проделанную работу-исследование работы симуляторов risc-v, изначальная цель которой была в оптимизации симулятора, написанного на Sail.

В какой-то момент цель трансформировалась в создание чего-то для тестирования различных симуляторов в принципе.

По ряду причин не получалось запускать одинаково собранный одинаковый тест на всех симуляторах, что в общем и целом мотивирует к созданию какой-нибудь штуки, которая бы облегчила тестирование под разными симуляторами/разных симуляторов.

Немножко описания общих вещей и описания сопутствующих проблем, которое скорее всего не особо подробное, уже не помню почему именно.

Полезность, наверное, ограничивается, разделом "Что-то о решении".

Задумывалось как черновик для текста к защите промежуточных результатов ВэКаЭр, получилось как всегда.

Введение

Центральное процессорное устройство, или же просто процессор, есть "мозг" компьютера, отвечающий за выполнение машинных инструкций. Для описания работы процессора существует архитектура набора команд (ISA).

RISC-V - это открытая архитектура набора команд, которая приобретает значительную популярность в последние годы. Так, например, Google в начале 2023 года объявил о планах по поддержке RISC-V Android-ом. (ссылка) Прежде всего RISC-V это открытая архитектура, поддерживаемая некоммерческой организацией RISC-V Foundation, а следовательно любой желающий может бесплатно использовать, модифицировать и распространять свою версию спецификации. RISC-V реализует подход к проектированию процессоров, при котором используются максимально простые инструкции (RISC). В отличие от архитектуры CISC, когда инструкций меньше, но они более трудоёмкие для процессора, в случае RISC может понадобиться использовать большее количество инструкций. Тем не менее лучшая производительность может достигаться засчёт компенсации объёмов кода его простотой. Перечисленные факторы приводят к тому, что RISC-V не прекращает набирать популярность, и открываются новые перспективы в индустрии относительно использования этой архитектуры.

По мере роста популярности RISC-V большее количество людей занимаются разработкой под эту архитектуру. Чтобы позволить разработчикам программного и аппаратного обеспечения тестировать свои решения для систем на RISC-V без необходимости иметь доступ к физической плате, разрабатываются симуляторы, включая Sail-RISCV, Spike, QEMU, Verilator, r2vm, Gem5 и другие. К сожалению в связи с рядом особенностей симуляторов могут возникать проблемы при тестировании на них решений под RISC-V. Скомпилированные одним и тем же способом программы могут запуститься на Qemu и не запуститься на Spike и наоборот. Для удобства разработчиков не хватает инструментов, позволяющих единообразно тестировать решения на различных симуляторах RISC-V, чтобы можно было сравнивать симуляторы по производительности и уровню поддержки архитектуры.

Постановка задачи

Целью данной работы является создание инструментария для удобного тестирования линейки симуляторов RISC-V. Для достижения данной цели были поставлены следующие задачи:

- Провести обзор симуляторов на RISC-V.
- Провести обзор и собрать набор тестов на соответствие и производительность. <-- вообще не было
- Обеспечить возможность удобно сравнивать производительность и соответствие актуальной ISA различных симуляторов.
- Поддерживать общедоступную аггрегированную отчётность о результатах тестов. <-- веб-сайт с данными об актуальных результатах тестов

// Для 1/2 вкр: //В рамках работы в течение осеннего семестра были поставлены следующие задачи: //- Провести обзор симуляторов на RISC-V. //- Провести обзор и собрать набор тестов на соответствие и производительность. //- Обеспечить возможность удобно сравнивать производительность различных симуляторов. <-- это не работает всё ещё, но хрен знает пока, какую переформулировку сюда присунуть

Обзор

Симуляторы

Spike

Spike - это симулятор RISC-V с открытым исходным кодом. Первоначально он был создан Эндрю Уотерманом, Юнсупом Ли и Дэвидом Паттерсоном в Калифорнийском университете в Беркли и в настоящее время поддерживается RISC-V Foundation. Он написан на C++ и разработан таким образом, чтобы быть простым, легковесным и удобным в использовании, что делает его популярным среди разработчиков. Spike поддерживает широкий спектр конфигураций RISC-V, включая 32-разрядные и 64-разрядные режимы, а также стандартные расширения.
Из недостатков --- spike это не full-system симулятор. И всё же из-за большого выбора при конфигурации он достаточно сложен для понимания новичками.

Sail-RISCV

Sail-RISCV - это открытый эмулятор RISC-V, созданный исследователями из Кембриджского университета и выпущенный под лицензией BSD 2-Clause. Он написан с использованием языка Sail, специфичного для предметной области описания архитектур набора команд. Этот симулятор достаточно легко конфигурируемый и расширяемый, имеет понятный интерфейс командной строки К недостаткам этого симулятора можно отнести низкую производительность и отстутствие подробной документации, из-за чего работа с симулятором может внезапно очень сильно осложниться при возникновении каких-либо проблем.

QEMU

QEMU - это full-system симулятор, который включает поддержку широкого спектра архитектур, включая RISC-V. QEMU выпущен под лицензией GPLv2. Симулятор легко настраивается и может использоваться для имитации различных конфигураций RISC-V, включая как 32-разрядный, так и 64-разрядный

режимы. QEMU включает поддержку деталей системного уровня, таких как иерархия памяти и периферийные устройства, что делает его мощным инструментом для моделирования на системном уровне. Из недостатков можно отметить, что QEMU не поддерживает эмуляцию с точностью до цикла и достаточно сложен для освоения.

R2VM

R2VM (Rust for RISC-V Virtual Machine) - это легкий и быстрый симулятор RISC-V, специально разработанный для разработки встраиваемых систем. R2VM выпущен под лицензиями MIT и APACHE поддерживает RISC-V ISA и оптимизирован для моделирования систем RISC-V с низким энергопотреблением и памятью. Одним из ключевых преимуществ R2VM является его простота в использовании, что делает его популярным выбором для разработчиков, которым необходимо быстро тестировать и отлаживать код RISC-V на виртуальном оборудовании. Однако R2VM не включает поддержку деталей системного уровня, таких как иерархия памяти и периферийные устройства, что может ограничить его полезность для некоторых приложений.

gem5

gem5 - это модульный, расширяемый симулятор под лицензией BSD, который поддерживает несколько архитектур, включая RISC-V. Появился в результате слияния симуляторов GEMS и M5, созданных исследовательскими группами из Висконсинского и Мичиганского университетов соответственно. В настоящее время поддерживается сообществом разработчиков как из академических кругов, так и из промышленности, например, такими организациями, как RISC-V Foundation, Google и ARM. Симулятор имеет открытый исходный код и выпущен под лицензией BSD, которая в частности допускает коммерческое использование. gem5 написан на C++ и предоставляет настраиваемую среду для разработки и тестирования под RISC-V. Он поддерживает все стандартные расширения RISC-V и предоставляет ряд функций для моделирования различных аспектов процессоров, включая иерархии памяти и периферийных устройств. Ещё одним преимуществом данного симулятора является его модульность, которая позволяет разработчикам легко добавлять в симулятор новые модели процессоров, системы памяти и другие компоненты. Это делает gem5 популярным выбором для исследований, поскольку его можно настроить в соответствии с конкретными требованиями проекта.

(!) С верилятором на момент попыток написания того, что тут есть, не разобрался (сходу не понял, как нормально на нём запуститься, скорее всего что-то пропустил в доках) Verilator - это симулятор с точностью до цикла, специально разработанный для проектирования и верификации оборудования. Verilator может использоваться для моделирования RTL и включает поддержку широкого спектра конфигураций RISC-V, что делает его популярным выбором для разработчиков под RISC-V. Однако Verilator плохо подходит для разработки программного обеспечения или моделирования на системном уровне, поскольку он не включает поддержку деталей системного уровня, таких как иерархия памяти и периферийные устройства. (!) Аналогично тому, что выше, но этот по идее нестрашно пропустить RISC MARSS - это микроархитектурный симулятор с точностью до цикла, который можно использовать для имитации процессоров RISC-V на уровне RTL. RISC MARSS поддерживает широкий спектр конфигураций RISC-V и микроархитектурных функций, что делает его популярным выбором для исследований и разработок в области компьютерной архитектуры. Одним из ключевых преимуществ RISC MARSS является его высокий уровень точности, который позволяет проводить детальный анализ поведения процессоров RISC-V на микроархитектурном уровне. RISC MARSS также включает в себя множество инструментов анализа и функций, таких как трассировка, мониторинг производительности и оценка мощности, которые могут быть использованы для анализа и оптимизации производительности систем RISC-V.

Тесты

При создании решений под RISC-V разработчики в любом случае столкнутся с необходимостью тестировать результаты своей работы. Проблемой может оказаться тот факт, что не у каждого будет под рукой плата на RISC-V в любой момент времени, когда надо провести тесты. Даже в случае, когда разработчик озаботился приобретением платы, вполне возможно, что в процессе работы использование симулятора окажется предпочтительней. В связи со сказанным встаёт вопрос о том, как среди всего разнообразия симуляторов выбирать подходящий. Вполне естественными могут оказаться требования к тому, чтобы симулятор работал на определённых задачах быстрее остальных, или же чтобы симулятор максимально полно поддерживал актуальную версию ISA.

Тесты на соответствие (звучит как-то *meh*, по англ. compliance tests более органично, мб не тот перевод)

От процессоров и симуляторов RISC-V ожидается, что будут корректно работать все описанные в архитектуре инструкции. Из этого возникает желание тестировать реальные системы и симуляторы на соответствие актуальным стандартам. Для этого можно тестировать отдельные команды процессора. (тут вроде как ещё что-то, но я не родил формулировку)

Целью проведения тестов на соответствие в рамках данной работы является выявление того, соответствует ли разрабатываемый симулятор стандартам RISC-V или нет. По результатам тестирования можно получать актуальную информацию о том, насколько полно и точно тот или иной симулятор моделирует архитектуру.

Актуальный открытый набор тестов доступен на GitHub. -> оставить ссылки https://github.com/riscv-non-isa/riscv-arch-test, https://github.com/riscv-software-src/riscv-tests.

Тесты производительности

Для того, чтобы сравнить производительность симуляторов между с собой и с платами на RICV-V, следует прибегнуть к используемым для этого бенчмаркам. Разработчикам процессоров или симуляторов бенчмарки помогают оптимизировать решения, а разработчикам инструментов, которые тестируются на симуляторах, бенчмарки как минимум могут помочь составить полную картину о том, на каком симуляторе будет быстрее всего тестировать.

Для того, чтобы протестировать процессор или симулятор на скорость работы, пишутся тесты, содержащие, например, большое количество тех или иных арифметических операций с целочисленными значениями, значениями с плавабщей точкой. Бенчмарк может служить для проверки скорости работы присваиваний, вызовов процедур и т.д. Так, например, ещё в 1984г. появился бенчмарк Dhrystone, а в 2009 набор тестов CoreMark, который по своей концепции должен был заменить предшественника.

Существует набор бенчмарков, предложенный в официальном репозитории RISC-V на Github.

-> ссылка на dhrystone -> оставить ссылку https://github.com/riscv-software-src/riscv-tests.

Что-то о решении

//- тезис 1 -> изначальные проблемы Одна из первых задач в рамках изучения RISC-V и симуляторов с поддержокой этой архитектуры, в частности Sail-RISCV, подразумевала запуск произвольного бенчмарка, написанного на C, на различных симуляторах с целью анализа скорости работы Sail-RISCV относительно других решений. Для получения исполняемого файла использовалась (указать на riscv-gnu-toolchain с его gcc). Полученный таким образом бинарник можно успешно запустить на QEMU, но невозможно на Sail-RISCV.

//- тезис 2 --- HTIF+tohost/fromhost+остальное --> Добавить секции --> Добавить линкер скрипт Для запуска под Sail-RISCV или Spike сначала потребовалось реализовать протокол HTIF(ссылку) взаимодействия симулятора с хостом. Для этого необходимо было добавить в генерируемый ассемблерный листинг 4-байтовые секции fromhost и tohost. Как выяснилось, в случае с указанными симуляторами эти манипуляции необходимы для того, чтобы симулятор мог сообщить хосту, на котором он был запущен, что исполнение программы завершено --- успешно или же нет, в зависимости от того, какое значение записать в tohost. Стоит обратить внимание на то, что для поставленных задач fromhost никак не используется и описан выше исключительно с целью полноты реализации HTIF. В случае с кодом на Си для этого достаточно объявить extern переменные типа uint64t. Дополнительно сборка теперь требовала линкер скрипта для корректной сборки. Исходя из спецификации devicetree для симулятора необходимо указать начало стека по адресу 0x80000000 и выровнять секцию tohost. Ошибка, связанная с взаимодействием хоста и симулятора, решилась, но задача по запуску тестовой программы на Sail-RISCV не была решена. --> На этом моменте вообще хрен знает, что было не так, тогда я сначала потратил кучу времени, а потом просто увидел, что есть реализация в готовом виде, но почему оно работает, а описание выше --- нет, не особо понятно В связи с этим было решено прибегнуть к имеющейся генерации исполняемых файлов из бенчмарков в актуальном репозитории с тестами для RISC-V, которая изначально была проигнорирована в силу подозрительной усложнённости. По итогу только с её помощью получилось собрать тесты, запускающиеся под Sail-RISC-V и Spike. //- тезис 3 --- на QEMU не работает --> Режим Изначально на QEMU не получалось запустить тесты из-за того, что генератор тащил за собой ассемблерные инструкции, которые можно было использовать только в машинном режиме процессора, что, во-первых, подозрительно в случае с тривиальными тестами арифметики, а во-вторых не поддерживается qemu. (тут может быть большая ошибка, но я так не нашел) Код для генерации тестов был модифицирован с целью избежания этой ошибки (по факту --- тупо удалены ломающие вызовы), конечно же, с сохранением корректной работы тестов на Sail-RISCV и Spike. К сожалению, этого не хватило для того, чтобы запустить тесты на QEMU, gem5 или r2vm.

--> Стек сломался --- в целом всё выглядело как какая-то проблемка при работе со стеком, т.к. программы стабильно падали с segmentation fault во время попытки постучать по невалидному адресу. Что именно за проблема, так и не разобрался. Вроде как со стеком нормально обращается обёртка, которая используется для того, чтобы программки вообще запускались. Но это не точно..

С целью поиска ошибок было решено отлаживать исполнение тестовой программы на qemu под gdb (это очередная штука, которую уже лучше воткнуть в секцию с обзором), где достать полезную информацию можно было исключительно из ассемблерного листинга. Судя по ассемблерному коду, qemu представляет набор регистров RISC-V как поля некоторой структуры. (не объяснено ещё, почему, надо как-то расписывать) При вызове очерной операции на запись происходило обращение за пределы допустимой области памяти, что вело к падению программы.

--> Как починить --- пока так и непонятно, я достаточно времени угрохал на попытки как-то вмешаться в этот процесс, но безрезультатно, после чего начал пытаться описать текстом не до конца сформированную задачу и непонятно каким путём идущий процесс. Ещё пытался посмотреть под другими симуляторами, в точности ли возникала та же проблема, но это настолько проблематично шло, что опять же отложил это в пользу текста.

Заключение

В результате проделанной работы были достигнуты следующие цели: -> перефраз пред. части