ПРОЕКТИРОВАНИЕ И СТРУКТУРА ПРОЦЕССОРА НА БАЗЕ АРХИТЕКТУРЫ RISC-V

Тыманович Н.А.

Институт информационных технологий Белорусского государственного университета информатики и радиоэлектроники
г. Минск, Республика Беларусь

Скудняков Ю.А.– канд. техн. наук, доцент

Аннотация. Работа посвящена изучению принципов работы процессоров и кодовой инфраструктуры для взаимодействия с процессором. Рассматривается процесс проектирования и структура процессора на базе архитектуры RISC-V.

Центральный процессор — исполнительные машинные инструкции, часть аппаратного управления компьютером или программируемого логического контроллера, отвечающий за

выполнение операций, заданными программами. Актуальна проблема проектирования и производства собственных процессорных решений как по экономическим, так и военно-политическим причинам. Компании, производящие собственные ядра, получают сотни миллионов долларов чистой прибыли ежегодно, а санкции и прочие внешние факторы не затрагивают стабильные поставки электроники в соответствующие области.

В качестве архитектуры была выбрана RISC-V архитектура. RISC-V — это бесплатная и открытая ISA, открывающая новую эру процессорных инноваций благодаря совместной работе на основе открытых стандартов. RISC-V ISA обеспечивает новый уровень бесплатного, расширяемого программного и аппаратного обеспечения свободой архитектуры, прокладывая ориентиры на ближайшие 50 лет компьютерного проектирования и инноваций [1].

Так как любой процессор производит выполнение определенных процессорных инструкций, следует ознакомиться с ними детально. В случае с RISC-V ситуация имеет определенные особенности, которые отличают эту архитектуру от других: например, RISC-V имеет базовый набор инструкций (чуть меньше 50), которые должны быть реализованы в точности по специальной документации. Такая документация называется RISC-V ISA (instruction set architecture) [2]. Помимо базового набора инструкций, RISC-V имеет расширения, которые добавляют комплекты новых, логически собранных в пакеты инструкций. На конец 2021 года имеются лишь несколько расширений, например, арифметические операции (в базовом наборе инструкций есть только сложение и вычитание), в том числе и с плавающей точкой, 64-битное расширение базового 32-битного расширения и расширение атомарных операций. Пусть этого не так много для проектирования высокопроизводительных решений, стандарт активно развивается, появляются и стабилизируются новые расширения, такие как векторные расширения, которые обеспечат новые рекорды производительности. Подход создания базового набора инструкций с выбором необходимых расширений позволяет беспрепятственно развивать архитектуру без затруднений для обратной совместимости, а также производить на единой архитектуре с одинаковой эффективностью чипы, предназначенные для различных сегментов, как встраиваемые системы, где делается упор на энергоэффективность и цену, и пользовательские решения, где важна производительность.

Набор регистров является неотъемлемой частью любой процессорной архитектуры. В RISC-V имеются 32 регистра общего назначения. Все операции между ними производятся одинаково без каких-либо отличий, за исключением нулевого (zero) регистра: все попытки записи туда игнорируются, а чтение всегда возвращает 0. Это часто бывает полезно, когда нам не нужно сохранять результат некоторых операций или же есть необходимость иметь 0 в качестве операнда.

Как и в остальных архитектурах, операции в RISC-V кодируются в особый формат, однако (за исключением особого расширения сжатых инструкций), инструкции имеют всегда одинаковую длину кодировки (32 бита) вне зависимости от разрядности процессора. Такой подход существенно упрощает декодирование инструкций, работу с памятью и переиспользование частей расширения непосредственно в реализациях других процессоров.

Рассмотрим пример кодировки инструкции add – инструкции суммирования содержимых регистров и записи результата в третий регистр. На рисунке 1 представлен пример кодировки соответствующей инструкции. И типовой кодировки базового расширения R32I

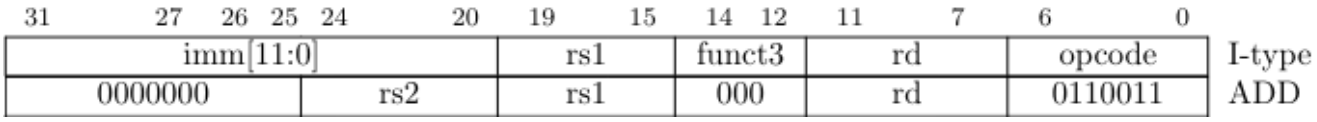

Рисунок 1 – типовая кодировка и кодировка add инструкции

Первая строчка отображает то, как обычно будет записана типовая инструкция в данном расширении. Opcode – номер операции, который описан в документации на каждую инструкцию константным значением. В случае с инструкцией add это будет 0110011. rd – номер регистр назначения результата. Funct3 – дополнительная подсказка декодеру, какую операцию реализовывать, на случай если опкоды инструкций одинаковы. Rs1 – номер первого операнда. Rs2 – номер второго операнда. Оставшееся место заполняем нулями. При детальном рассмотрении типовой кодировки и кодировки конкретной инструкции можно заметить, что у нас отсутствует imm(произвольная константа), которая занимает 12 бит. Это связано с тем, что логика инструкции сделана так, что никаких констант нам не нужно, а освободившееся место мы использовали под второй операнд, положение которого не стандартизировано в типовой кодировке. Если бы у нас была константа, но не было нужды в номере регистра второго операнда, с большой вероятностью оно бы находилось именно в 31-20-ом битах. Таким образом, если мы захотим сложить r1 и r2, а записать результат в r3, кодировка будет иметь следующий вид: 0000000_00010_00001_000_00011_0110011
Основная Задача проектировщика процессора состоит в том, чтобы согласно спецификации (RISC-V ISA в нашем случае) реализовать кодировку и логику работы каждой инструкции.

Реализация осуществляется непосредственно на языке описания аппаратуры, таком как, например, verolog, vhdl и т.п. По факту, должен быть создан блок памяти, который будет хранить в себе прошивку, декодер, который читает оттуда инструкции и блок, выполняющий их.

При верной реализации должна получиться машина, которая сможет работать в экосистеме RISC-V. Под экосистемой подразумевается набор программных решений, таких как комплиятор и линковщик и прочие binutils, созданные для данной архитектуры. Чтобы проверить работоспособность нашего процессора, необходимо скомпилировать произвольный код, например, на C, с использованием компилятора для встраиваемых систем (поддержка ОС процессором требует MPU и сильно усложняет пример), для этого нужно написать собственный скрипт компоновщика, который отобразит то, как должна храниться программа в памяти. Пример скрипта ликовщика я опущу, так как они сильно отличаются от модели памяти и способа загрузки конкретного процессора. Если все разработано правильно, процессор должен выполнять инструкции, скомпилированные из языка более высокого уровня компилятором, который был разработан под соответствующую архитектуру и является общим для всех процессоров, ее реализующим.

В случае с RISC-V кроме секции .text, где находятся инструкции, в отличии от arm, нет дополнительных секций, таких как, например, rodata, где находятся константы. Место этого константы заносятся инструкциями непосредственно в коде секции .text, что упрощает максимально линковку и дает возможность запускать сразу код без необходимости в предварительной инициализации памяти.

Таким образом, следуя реализации процессоров по существующим архитектурам, можно получить такие преимущества, как готовый, спроектированный ведущими мировыми специалистами стандарт, использовать все программные решения, которые созданы в мире под соответствующую архитектуру и комьюнити, которое может оказать поддержку в определенных областях.

Список использованных источников:

1. RISC-V official site [Электронный ресурс]. - Режим доступа: https: // riscv.org/about /– Дата доступа: 22.02.2022.

2. RISC-V ISA [Электронный ресурс]. - Режим доступа: https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf – Дата доступа: 22.02.2022.