**Система команд процессора RISC-V**

В архитектуре RISC-V имеется обязательное для реализации базовое подмножество в количестве 47 команд и несколько стандартных опциональных расширений. В базовый набор входят: минимальный набор арифметических/битовых операций на регистрах, операций с памятью (load/store), инструкций условной и безусловной передач управления (или переходов), а также небольшое число служебных инструкций. Инструкции базового набора имеют длину 32 бита с выравниванием на границу 32-битного слова.

Операции условных переходов (ветвления) не используют каких-либо общих флагов, как результатов ранее выполненных операций сравнения, а непосредственно сравнивают свои регистровые операнды. Базис операций сравнения минимален, а для поддержки комплементарных операций операнды просто меняются местами.

Базовое подмножество команд использует следующий набор регистров: специальный регистр x0 (zero), содержащий значение 0, всегда получаемое при чтении из этого регистра, 31 целочисленных регистра общего назначения (РОНы) x1 - x31, регистр счётчика команд PC, задающий адрес текущей команды и используемый косвенно, а также множество CSR-регистров (Control and Status Registers), используемых для управления и изменения состояний системы на различных уровнях привилегий. Всего может быть адресовано до 4096 CSR-регистров.

Вся арифметико-логическая обработка данных может производиться только над регистрами, при этом в основном формате регистр-регистр используются два регистра-источника операндов Rsrc1 и Rsrc2 (далее rs1 и rs2) и регистр результата Rdest (далее rd). Для использования ячеек основной памяти применяются инструкции загрузки (Load) данных из ячеек памяти в РОНы и выгрузки (Store) данных из РОН в ячейки памяти.

Архитектура использует только [little-endian](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B1%D0%B0%D0%B9%D1%82%D0%BE%D0%B2" \l "%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%BE%D1%82_%D0%BC%D0%BB%D0%B0%D0%B4%D1%88%D0%B5%D0%B3%D0%BE_%D0%BA_%D1%81%D1%82%D0%B0%D1%80%D1%88%D0%B5%D0%BC%D1%83" \o "Порядок байтов) модель — первый байт операнда в памяти соответствует наименее значащим битам значений регистрового операнда (аналогично архитектуре х86).

К **базовым вычислительным инструкциям** относятся:

- арифметические add, sub ;

- сравнения slt, sltu (знаковое и беззнаковое);

- логические and, or, xor ;

- сдвиговые sll, srl, sra (логические сдвиги влево/вправо и арифметический сдвиг вправо).

Примеры вычислительных инструкций:

sub x3, x1, x2 # x3 = x1 - x2

slt x3, x1, x2 # if x1 < x2 then x3 = 1 else x3 = 0

and x3, x1, x2 # x3 = x1 & x2

sll x3, x1, x2 # x3 = x1 << x2

Вычислительные инструкции также могут использоваться в формате регистр- непосредственное данное, при котором один операнд размещается в регистре, а второй – константа, закодированная в самой инструкции:

Oper rd, rs1, const

Например:

add x3, x1, 7 # x3 = x1 + 7

sll x3, x1, 3 # x3 = x1 << 3

Операции умножения, деления и вычисления остатка не входят в базовый набор инструкций, а выделены в отдельное расширение (M — Multiply extension), рассматриваемое ниже.

К **базовым инструкциям управления** программой относятся:

1. инструкции условного перехода

beq (=), bne (!=), blt (<), bge (≥), bltu (<u), bgeu(≥u)

Формат: Oper rs1, rs2, label

Примеры:

1) blt x1, x2, lab1 # if x1<x2 then PC = PC+lab1

2) для реализации сложного ветвления типа

if x1<x2 then x3=x1+2 else x3=x2+4 следует использовать код

bge x1, x2, else

addi x3, x1, 2

beq x0, x0, end

else: addi x3, x2, 4

end: продолжение

1. инструкции безусловного перехода

- безусловный переход с сохранением адреса возврата

jal ra, label

Здесь место перехода определяется меткой label, которая задается 20-битным смещением относительно РС (адреса текущей инструкции); ra - регистр для сохранения адреса возврата (адреса следующей после jal команды); в качестве ra в архитектуре RISC-V используется РОН x1;

- безусловный переход по значению из регистра с сохранением адреса возврата

jalr ra, offset(rs1)

Здесь место перехода определяется значением из регистра rs1 плюс заданное 12-битное смещение offset; в этом случае можно перейти по любому 32-битному адресу.

Сохранение адреса возврата в регистре выполняется быстрее, чем в архитектурах, которые сохраняют адрес возврата в стеке в памяти.

К **базовым инструкциям операций с памятью** относятся:

1. команда чтения из памяти в регистр (Load)

LB/H/W rd, rs1, imm # rd = M[rs1+imm]

1. команда записи из регистра в память (Store)

SB/H/W rs1, rs2, imm # M[rs1+imm] = rs2

Здесь в зависимости от размера данного Byte/Half/Word используются соответствующие версии операций. Для доступа к памяти задается базовый адрес, размещаемый в регистре rs1, и смещение в виде непосредственного значения (imm). Базовый адрес нужен для обращения ко всему объему памяти: в случае 32-битного адреса - к 4Гб памяти.

Пример вычисления выражения a = b + c для данных в памяти.

Память

|  |  |
| --- | --- |
| Адрес | Значение |
| 0х0 |  |
| 0х4 |  |
| 0х8 | b |
| 0хС | c |
| 0х10 |  |
| 0х14 |  |
| 0х18 | a |
| 0х1С |  |
| 0х20 |  |

x1 = Load (0x8)

x2 = Load (0xC)

Add x3, x1, x2

Store (0x18), x3

Реальные команды будут иметь вид:

x1 = LW 0x8(x0)

x2 = LW 0xC(x0)

Add x3, x1, x2

Store 0x18(x0), x3

Здесь базовый адрес равен 0.

Рассмотренные базовые инструкции процессора RISC-V для 32-битной архитектуры имеют форматы, представленные в таблице 1.

Таблица 1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тип | 31 … 25 | | 24 … 20 | 19 … 15 | 14 … 12 | 11 … 7 | | 6 … 2 | 1 | 0 |
| **Регистр/регистр** | funct7 | | rs2 | rs1 | funct3 | rd | | код операции | 1 | 1 |
| **С непоср. оп-дом** | ± | imm[10:0] | | rs1 | funct3 | rd | | код операции | 1 | 1 |
| **С длинным оп-ом опопоперандом** | ± | imm[30:12] | | | | rd | | код операции | 1 | 1 |
| **Сохранение (Store)** | ± | imm[10:5] | rs2 | rs1 | funct3 | imm[4:0] | | код операции | 1 | 1 |
| **Ветвление (Branch)** | ± | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | [11] | код операции | 1 | 1 |
| **Переход (Jump)** | ± | imm[10:1] | [11] | imm[19:12] | | rd | | код операции | 1 | 1 |

Здесь биты 0 и 1 всегда равны «11».

Остальные поля имеют следующее назначение:

* rs1 – номер регистра в котором находится первый операнд
* rs2 – номер регистра в котором находится второй операнд
* rd – номер регистра в который будет записан результат
* opcode – частично задает тип формата
* funct7 + funct3 – совместно с opcode определяют вид операции

Для упрощения программирования на ассемблере используются специальные псевдоинструкции, заменяемые ассемблером на реально существующие инструкции. Так псевдоинструкция пересылки данного из одного регистра в другой mv x2, x1 заменяется на инструкцию addi x2, x1, 0.

Позже мы рассмотрим полный состав псевдоинструкций.

При одинаковой кодировке инструкций в RISC-V предусмотрены реализации архитектур с 32, 64 и 128-битными регистрами общего назначения и наборы операций RV32I, RV64I и RV128I соответственно. Разрядность регистровых операций всегда соответствует размеру регистра, а одни и те же значения в регистрах могут трактоваться целыми числами как со знаком, так и без знака. Нет операций над частями регистров и нет каких-либо выделенных «регистровых пар». Размер операнда может отличаться от размера регистра только в операциях с памятью.

Операции не сохраняют где-либо биты переноса и/или переполнения, также аппаратно не генерируются исключения по переполнению и даже по делению на 0. Все необходимые проверки операндов и результатов операций должны производиться программно.

Целочисленная арифметика расширенной точности (большей, чем разрядность регистра) должна явно использовать операции вычисления старших битов результата. Например, для получения старших битов произведения регистра на регистр имеются специальные инструкции.

Поскольку кодировка базового набора инструкций не зависит от разрядности архитектуры, то один и тот же код потенциально может запускаться на различных RISC-V архитектурах, определять разрядность и другие параметры текущей архитектуры, наличие расширений системы инструкций, а потом автоконфигурироваться для целевой среды выполнения.

Полный состав базового набора команд приведен в таблице 2.

Таблица 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Обозначение  в ассемблере | Наименование | Тип формата | Описание | Примеча-ние |
| add | ADD | R | rd = rs1 + rs2 |  |
| sub | SUB | R | rd = rs1 - rs2 |  |
| xor | XOR | R | rd = rs1 ⊕ rs2 |  |
| or | OR | R | rd = rs1 ! rs2 |  |
| and | AND | R | rd = rs1 & rs2 |  |
| sll | Shift Left Logical | R | rd = rs1 << rs2 |  |
| srl | Shift Right Logical | R | rd = rs1 >> rs2 |  |
| sra | Shift Right Arith | R | rd = rs1 >> rs2 | msb-extends |
| slt | Set Less Than | R | rd = (rs1 < rs2)?1:0 |  |
| sltu | Set Less Than (U) | R | rd = (rs1 < rs2)?1:0 | zero-extends |
| addi | ADD Immediate | I | rd = rs1 + imm |  |
| xori | XOR Immediate | I | rd = rs1 ⊕ imm |  |
| ori | OR Immediate | I | rd = rs1 ! imm |  |
| andi | AND Immediate | I | rd = rs1 & imm |  |
| slli | Shift Left Logical Imm | I | rd = rs1 << imm[0:4] |  |
| srli | Shift Right Logical Imm | I | rd = rs1 >> imm[0:4] |  |
| srai | Shift Right Arith Imm | I | rd = rs1 >> imm[0:4] | msb-extends |
| slti | Set Less Than Imm | I | rd = (rs1 < imm)?1:0 |  |
| sltui | Set Less Than Imm (U) | I | rd = (rs1 < imm)?1:0 | zero-extends |
| lb | Load Byte | I | rd = M[rs1+imm][0:7] |  |
| lh | Load Half | I | rd = M[rs1+imm][0:15] |  |
| lw | Load Word | I | rd = M[rs1+imm][0:31] |  |
| lbu | Load Byte (U) | I | rd = M[rs1+imm][0:7] | zero-extends |
| lhu | Load Half (U) | I | rd = M[rs1+imm][0:15] | zero-extends |
| sb | Store Byte | S | M[rs1+imm][0:7]=rs2[0:7] |  |
| sh | Store Half | S | M[rs1+imm][0:15]=rs2[0:15] |  |
| sw | Store Word | S | M[rs1+imm][0:31]=rs2[0:31] |  |
| beq | Branch = | B | if(rs1 = rs2) PC += imm |  |
| bne | Branch != | B | if(rs1 != rs2) PC += imm |  |
| blt | Branch < | B | if(rs1 < rs2) PC += imm |  |
| bge | Branch ≥ | B | if(rs1≥ rs2) PC += imm |  |
| bltu | Branch < (U) | B | if(rs1 < rs2) PC += imm | zero-extends |
| bgeu | Branch ≥ (U) | B | if(rs1 ≥ rs2) PC += imm | zero-extends |
| jal | Jump And Link | J | rd = PC+4; PC += imm |  |
| jalr | Jump And Link Reg | I | rd = PC+4; PC = rs1 + imm |  |
| lui | Load Upper Imm | U | rd = imm << 12 |  |
| auipc | Add Upper Imm to PC | U | rd = PC + (imm << 12) |  |
| ecall | Environment Call | I | Transfer control to OS |  |
| ebreak | Environment Break | I | Transfer control to debugger |  |

**Дополнительные сведения по базовому набору команд**

1. ***Непосредственная адресация***

RISC-V обрабатывает 32-битные константы и адреса с помощью инструкций, которые устанавливают старшие 20 бит 32-битного регистра. Инструкция непосредственной загрузки lui, загружает 20 бит в биты от 31 до 12. Затем вторая инструкция addi, может установить младшие 12 бит.

При выполнении операций с константами большого размера возникают проблемы, связанные с их загрузкой в регистры. Так при вычислении выражения a = b + 0x123456 размещение константы в инструкции addi требует 24 бита, которых в формате не предусмотрено. Для разрешения проблемы можно использовать псевдоинструкцию li для загрузки больших констант, которая преобразуется ассемблером в две инструкции: lui x5, 0x123 (загрузить константу в старшие биты 31-12 регистра) и addi x5, x5, 0x456 (сложение того же регистра с младшими 12 битами).

Этот метод может быть расширен для того, чтобы создавать позиционно-независимый код путем добавления инструкции auipc, которая генерирует 20 старших битов адреса путем добавления смещения к счетчику программы и сохранения результата в базовом регистре. Это позволяет программе генерировать 32-разрядные адреса, используя смещение относительно счетчика программы.

1. ***Доступ к памяти и обмен с памятью***

Как и многие проекты с RISC-концепцией, RISC–V является архитектурой load-store: обычные инструкции адресуют только регистры, а инструкции load и store реализуют обмен с памятью.

Транзакции с памятью осуществляются блоками, размер в байтах которых должен быть целой неотрицательной степенью 2, от одного байта до размера регистра включительно. Операнд в памяти должен иметь «естественное выравнивание» (адрес кратен размеру операнда).

Для пары инструкций сохранения/загрузки регистра операнд в памяти определяется размером регистра выбранной архитектуры, а не кодировкой инструкции (код инструкции один и тот же для RV32I, RV64I и RV128I, но размер операндов 4, 8 и 16 байт соответственно).

Для всех допустимых размеров операндов в памяти, меньших, чем размер регистра, имеются отдельные инструкции загрузки/сохранения младших битов регистра, в том числе для загрузки из памяти в регистр есть парные варианты инструкций, которые позволяют трактовать загружаемое значение как со знаком (старшим знаковым битом значения из памяти заполняются старшие биты регистра) или без знака (старшие биты регистра устанавливаются в 0).

Большинство инструкций загрузки и хранения включают 12-битное смещение и два идентификатора регистра. Один регистр является базовым регистром. Другой регистр является источником (для записи) или местом назначения (для чтения.) Смещение добавляется в базовый регистр для получения адреса. Формирование адреса в виде базового регистра плюс смещение позволяет отдельным инструкциям получить доступ к структурам данных. Например, если базовый регистр указывает на вершину стека, отдельные инструкции могут обращаться к локальным переменным подпрограммы в стеке. Использование постоянного нулевого регистра в качестве базового адреса позволяет отдельным командам обращаться к памяти вокруг нулевого адреса.

Память адресуется как 8-битные байты, причем слова находятся в порядке little-endian. Слова, вплоть до размера регистра, могут быть доступны с инструкциями загрузки и хранения.

RISC-V управляет системами памяти, которые совместно используются процессорами или потоками, обеспечивая, чтобы поток выполнения всегда видел свои операции памяти в запрограммированном порядке. Но между потоками и устройствами ввода-вывода действия RISC-V упрощены: он не гарантирует порядок операций с памятью, за исключением конкретных инструкций, таких как fence.

Инструкция fence гарантирует, что результаты предшествующих операций видны для последующих операций других потоков или устройств ввода-вывода. Инструкция fence может гарантировать порядок комбинаций как операций работы с памятью, так и операций ввода-вывода с отображением в памяти. Например, он может разделять операции чтения и записи памяти, не влияя на операции ввода-вывода. Или, если система может работать с устройствами ввода/вывода параллельно с памятью, fence не заставляет их ждать друг друга. Один процессор с одним потоком может декодировать fence как холостую операцию nop.

Как многие наборы команд RISC архитектура RISC-V не имеет адресных режимов, которые записывают данные обратно в регистры. Например, не предусмотрено команд с автоинкрементом адресных регистров.

1. ***Вызовы подпрограмм, переходы и ветвления***

Битовый формат jalr похож на регистр-относительные команды load и store. Как и они, jalr может использоваться с инструкциями, которые устанавливают верхние 20 бит базового регистра, чтобы сделать 32-разрядные адреса условного перехода, либо абсолютные адреса (используя lui), либо относительно счетчика программ PC (используя auipc для позиционно-независимого кода). Использование постоянного нулевого базового адреса позволяет создавать однокомандные вызовы с небольшим фиксированным положительным или отрицательным смещением адреса.

RISC-V перерабатывает команды jal и jalr для получения безусловных 20-разрядных переходов относительно счетчика программ PC и безусловных 12-разрядных переходов на основе регистров. Переходы просто устанавливают регистр связи 0, чтобы обратный адрес не сохранялся.

RISC-V также использует команду jalr для возврата из подпрограммы: для этого базовый регистр команды jalr устанавливается как регистр связи, сохраненный jal или jalr. Если смещение jalr и регистр связи равны нулю, то смещение отсутствует, и обратный адрес не сохраняется.

Как и многие проекты RISC, при вызове подпрограммы компилятор RISC-V должен использовать отдельные инструкции для сохранения регистров в стеке при запуске, а затем для восстановления их из стека при выходе. RISC-V не имеет инструкций сохранения или восстановления нескольких регистров. Считалось, что они делают процессор слишком сложным и, возможно, медленным. Такая реализация может занять больше места в коде. Дизайнеры архитектуры планировали уменьшить размер кода с помощью библиотечных процедур для сохранения и восстановления регистров.

RISC-V не имеет регистра кодов условия или бита переноса. Разработчики считали, что коды условий делают быстрые процессоры более сложными, заставляя взаимодействовать между инструкциями на разных этапах выполнения, что требуют больше энергии. Вместо этого, RISC-V использует инструкции условного перехода с коротким смещением, которые выполняются при сравнении регистров на: равно, не равно, меньше, без знаковое меньше, больше или равно и без знаковое больше или равно. Десять операций сравнения-ветвления выполняются, используя только шесть инструкций, путем изменения порядка операндов в ассемблере. Инструкции ветвления-сравнения имеют двенадцатиразряднное знаковое смещение и переходят относительно счетчика программ PC.

ISA RISC-V требует предсказания ветвлений по умолчанию для проектируемых процессоров: условные ветви с переходом назад должны быть предсказаны, условные ветви с переходом вперед − нет. Предсказания легко декодировать в конвейерном процессоре: адреса перехода - это знаковые числа в дополнительном коде, которые прибавляются к счетчику программ PC. Ветви с переходом назад имеют отрицательные смещения адреса в дополнительном коде и содержат 1 в старшем значащем бите адреса. Ветви с переходом вперед имеют 0 старший бит адреса. Самый старший значащий бит находится в фиксированном месте в коде операции для ускорения конвейера.

Руководство ISA рекомендует оптимизировать программное обеспечение, чтобы избежать остановок конвейера, используя прогноз ветвлений по умолчанию. Это позволяет повторно использовать старший значащий бит знакового относительного адреса в качестве бита подсказки, чтобы предсказать, будет ли выполнен условный переход или нет. Эта особенность дает возможность использовать больше битов в кодах операций ветвления.

RISC-V не поддерживает предикацию (условное выполнение инструкций), поскольку ее разработчики утверждают, что процессоры без предикации легче проектировать, а оптимизирующие компиляторы менее склонны ошибочно использовать предикацию там, где она не должна использоваться. Дизайнеры утверждают, что очень быстрые, out-of-order CPU (исполнение [машинных инструкций](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B8%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F) не в порядке следования в  [коде](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), а в порядке готовности к выполнению) все равно делают предикацию, выполняя ветвь сравнения и условный код параллельно, а затем отбрасывая эффекты неиспользуемой ветви.

Наряду с базовым набором команд в архитектуре RISC-V предусмотрены стандартные расширенные наборы команд. Полный список наборов команд RISC-V приведен в таблице 3.

Таблица 3

|  |  |  |  |
| --- | --- | --- | --- |
| **Сокращение** | **Наименование** | **Версия** | **Статус** |
| **Базовые наборы** | | | |
| **RV32I** | Базовый набор с целочисленными операциями, 32-битный | 2.0 | Frozen |
| **RV32E** | Базовый набор с целочисленными операциями для [встраиваемых систем](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D0%BC%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), 32-битный, 16 регистров | 1.9 | Open |
| **RV64I** | Базовый набор с целочисленными операциями, 64-битный | 2.0 | Frozen |
| **RV128I** | Базовый набор с целочисленными операциями, 128-битный | 1.7 | Open |
| **Стандартные расширенные наборы** | | | |
| **M** | Целочисленное умножение и деление (Integer Multiplication and Division) | 2.0 | Frozen |
| **A** | [Атомарные операции](https://ru.wikipedia.org/wiki/%D0%90%D1%82%D0%BE%D0%BC%D0%B0%D1%80%D0%BD%D1%8B%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8) (Atomic Instructions) | 2.0 | Frozen |
| **F** | Арифметические операции с плавающей точкой над числами одинарной точности (Single-Precision Floating-Point) | 2.0 | Frozen |
| **D** | Арифметические операции с плавающей точкой над числами двойной точности (Double-Precision Floating-Point) | 2.0 | Frozen |
| **G** | Сокращеное обозначение для комплекта из базового и стандартного наборов команд | н/д | н/д |
| **Q** | Арифметические операции с плавающей точкой над числами четверной точности | 2.0 | Frozen |
| **L** | Арифметические операции над числами с фиксированной точкой (Decimal Floating-Point) | 0.0 | Open |
| **C** | Сокращённые имена для команд (Compressed Instructions) | 2.0 | Frozen |
| **B** | [Битовые операции](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D1%8B%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8) (Bit Manipulation) | 0.36 | Open |
| **J** | [Двоичная трансляция](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%82%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%86%D0%B8%D1%8F) и поддержка [динамической компиляции](https://ru.wikipedia.org/wiki/JIT-%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F) (Dynamically Translated Languages) | 0.0 | Open |
| **T** | [Транзакционная память](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C) (Transactional Memory) | 0.0 | Open |
| **P** | Короткие [SIMD](https://ru.wikipedia.org/wiki/SIMD)-операции (Packed-SIMD Instructions) | 0.1 | Open |
| **V** | [Векторные расширения](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BF%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F)) (Vector Operations) | 0.2 | Open |
| **N** | Инструкции [прерывания](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) (User-Level Interrupts) | 1.1 | Open |

Здесь статус «Frozen» означает, что разработка данного набора команд завершена. Статус «Open» означает, что набор команд находится в стадии разработки.

**Команды для встраиваемых применений**

Для встраиваемых применений может использоваться вариант архитектуры RV32E (Embedded) с сокращённым набором регистров общего назначения (первые 16). Уменьшение количества регистров позволяет не только экономить аппаратные ресурсы, но и сократить затраты памяти и времени на сохранение/восстановление регистров при переключениях контекста.

Инструкции с плавающей точкой в наборе Embedded не должны поддерживаться (спецификация запрещает это как неэкономичное), поэтому должна использоваться библиотека программного обеспечения с плавающей точкой. Также рекомендуется использовать сжатый набор команд C. Привилегированный набор команд поддерживает только машинный режим, пользовательский режим и схемы памяти, которые используют перемещение базовых и связанных адресов.

Ряд разработчиков также предложили меньшие нестандартные 16-битные ISA RV16E: то есть предлагалось использование 16-битных инструкций с 8 × 16-битными регистрами.

**Набор М целочисленных инструкций умножения и деления**.

Целочисленные инструкции умножения-деления (набор M) реализуются по формату «регистр-регистр» и включают в себя знаковое и беззнаковое умножение и деление.

MUL выполняет умножение XLEN-бит rs1 на XLEN-бит rs2 и помещает младшие XLEN-бит в регистр назначения rd. MULH, MULHU и MULHSU выполняют то же умножение, но возвращают верхние XLEN-биты полного 2×XLEN-разрядного произведения при умножении знаковых чисел, беззнаковых чисел и знакового на беззнаковое число соответственно. Если требуются как старшие, так и младшие биты произведения, то рекомендуемая кодовая последовательность такова: MULH[[S]U] rdh, rs1, rs2; MUL rdl, rs1, rs2 (спецификаторы исходных регистров должны быть в том же порядке, и rdh не может совпадать с rs1 или rs2). В микроархитектуре выполнение двух отдельных операций умножения можно объединить в одну операцию умножения.

DIV и DIVU выполняют целочисленное деление XLEN-бит rs1 на XLEN-бит rs2 со знаком и без знака с округлением в сторону нуля. REM и REMU представляют остаток соответствующей операции деления. Для REM знак результата равен знаку делимого. Как для деления со знаком, так и для деления без знака справедливо, что делимое = делитель × частное + остаток. Если требуется получить и частное, и остаток от деления, рекомендуемая кодовая последовательность такова: DIV[U] rdq, rs1, rs2; REM[U] rdr, rs1, rs2 (rdq не может совпадать с rs1 или rs2). В микроархитек туре можно объединить две отдельные операции деления. в одну.

**Наборы F, D, Q арифметических операций с плавающей точкой**

Для операций над числами с плавающей точкой используется набор дополнительных 32 регистров FPU (Floating Point Unit), которые совместно используются расширениями базового набора инструкций для трёх вариантов точности: одинарной — 32 бита (F extension), двойной — 64 бита (D - Double precision extension), а также четверной — 128 бит (Q - Quadruple precision extension).

Инструкции с плавающей точкой набора F включают арифметику с одинарной точностью, а также условные переходы, аналогичные целочисленной арифметике. Инструкции с плавающей точкой двойной точности (набор D) предполагают, что регистры с плавающей точкой являются 64-разрядными (двойной длины), и подмножество F совместимо с набором D. Также определяется 128-битный набор Q операций с плавающей точкой с квадратичной точностью. Процессоры RISC-V без плавающей точки могут использовать библиотеку программной эмуляции работы с плавающей точкой.

RISC-V не вызывает исключений по арифметическим ошибкам, включая переполнение, исчезновение значащих разрядов, субнормализацию и деление на ноль. Вместо этого как целочисленная, так и арифметика с плавающей точкой выдают разумные значения по умолчанию и устанавливают биты состояния. Деление на ноль может быть обнаружено одной ветвью после деления. Биты состояния могут быть проверены операционной системой или периодическим прерыванием.

**Атомарные операции с памятью**

В ISA RISC-V стандартизован отдельный набор атомарных операций (A — Atomic extension) - операции, каждая из которых либо выполняется целиком, либо не выполняется вовсе. Операция в общей области памяти называется **атомарной**, если она завершается в один шаг относительно других потоков, имеющих доступ к этой памяти.

RISC-V применяется в компьютерах, которые совместно используют память между несколькими процессорами и потоками. Стандартной моделью согласованности памяти RISC-V является согласованность освобождения. То есть операции load и store могут быть в целом переупорядочены, но некоторые операции load могут быть обозначены как операции подгрузки, которые должны предшествовать более поздним обращениям к памяти, а некоторые операции store могут быть обозначены как операции release (освобождения), которые должны следовать за более ранними обращениями к памяти.

Базовый набор команд включает в себя минимальную поддержку в виде команды fence для обеспечения упорядочения памяти. Хотя этого достаточно (операции fence r, rw обеспечивают подгрузку и fence rw, w обеспечивает освобождение), хотя комбинированные операции могут быть более эффективными.

Расширение atomic memory operation extension поддерживает два типа атомарных операций памяти для обеспечения release consistency. Во-первых, он предоставляет универсальные инструкции load-reserved (lr) и store-conditional (sc). Инструкция lr выполняет загрузку и пытается зарезерви-ровать этот адрес для своего потока. Выполняющаяся позднее операция sc для записи в зарезервированный адрес будет выполнена только в том случае, если резервирование не будет нарушено промежуточной операцией store из другого источника.

Вторая группа атомарных инструкций выполняет последовательности чтения-изменения-записи: load (которая необязательно является load-acquire) в регистр назначения, затем операция между загруженным значением и исходным регистром, затем store результата (которое может необязательно быть store-release). Создание дополнительных барьеров памяти позволяет комбинировать операции. Дополнительные операции включаются с помощью битов acquire и release, которые присутствуют в каждой атомарной инструкции. RISC-V определяет девять возможных операций: swap, add, побитовое and, or и exclusive-or; и знаковый и беззнаковый минимум и максимум.

Конструкция системы может оптимизировать эти совмещенные операции больше, чем lr и sc. Например, если регистр назначения для swap является постоянным нулем, загрузка может быть пропущена. Если сохраненное значение не изменено с момента загрузки, store может быть пропущено.

**Сокращенные команды**

Стандарт RISC-V ISA указывает, что все инструкции являются 32-битными. Это делает особенно простой реализацию, но приводит к большему размеру кода. Для компенсации 32-разрядных инструкций RISC-V 3⁄4 пространства кода операции зарезервировано для рекомендуемого сжатого набора команд переменной длины RVC, который включает 16-разрядные инструкции. Сокращенные команды уменьшают размер статического и динамического кода за счет добавления коротких 16-битных кодировок команд для обычных операций. Сжатые инструкции пропускают поля, которые часто равны нулю, используют небольшие непосредственные значения или обращаются к подмножествам (16 или 8) регистров. Операция addi очень распространена и часто сжимается.

Как и  в ARM's Thumb и MIPS16, сжатые инструкции являются просто псевдонимами для подмножества более крупных инструкций. В отличие от ARM's Thumb или сжатого набора MIPS, в RISC-V пространство было зарезервировано с самого начала, поэтому нет отдельного режима работы. Стандартные и сжатые инструкции могут быть смешаны свободно. Cжатие может быть реализовано в ассемблере, и компилятору не обязательно даже знать об этом.

**Привилегированный набор команд**

ISA RISC-V включает привилегированный набор инструкций. По состоянию на август 2019 года, спецификация версии 1.11 поддерживает несколько типов компьютерных систем:

* Системы, которые имеют только машинный режим, возможно, для встроенных систем,
* Системы как с машинным режимом (для супервизора), так и с пользовательским режимом для реализации операционных систем, запускающих ядро в привилегированном режиме.
* Системы с машинным режимом, гипервизорами, несколькими супервизорами и пользовательскими режимами под каждым супервизором.

*Здесь Гипервизор – это программное обеспечение, которое можно использо-вать для запуска нескольких виртуальных машин в одной физической машине. Каждая виртуальная машина имеет свою операционную систему и приложения. Гипервизор по мере необходимости выделяет базовые физические вычислительные ресурсы, такие как ЦП и память, для отдельных виртуальных машин. Таким образом поддерживается оптимальное использование физической ИТ-инфраструктуры.*

Указанные типы систем реализуются максимум с четырьмя кольцами привилегий и безопасности: машина, гипервизор, супервизор и пользователь. Каждый уровень также должен иметь тонкий слой поддерживающего программного обеспечения, которое взаимодействует с более привилегиро-ванным уровнем или аппаратным обеспечением.

Спецификация набора привилегированных инструкций явно определяет аппаратные потоки. Несколько аппаратных потоков являются обычной практикой в продвинутых компьютерах. Когда один поток останавливается, ожидая доступ к памяти, другие потоки могут продолжать выполняться. Аппаратные потоки могут помочь лучше использовать большое количество регистров и исполнительных блоков в быстрых процессорах out-of-order. Наконец, аппаратные потоки могут быть простым и мощным способом обработки прерываний: не требуется сохранение или восстановление регистров, просто выполняется другой аппаратный поток. Однако единственным аппаратным потоком, необходимым для компьютера RISC-V, является нулевой поток.

В спецификации поддерживается несколько систем памяти. Физический - подходит только для самых простых встроенных систем. Существует также три системы виртуальной памяти в стиле UNIX для кэширования памяти в системах массового хранения. Системы виртуальной памяти имеют три размера, с адресами длиной 32, 39 и 48 бит. Все системы виртуальной памяти поддерживают 4 страницы KiB, многоуровневые деревья таблиц страниц и используют похожие алгоритмы для обхода деревьев таблиц страниц. Здесь KiB *- Кибибайт* - [единица](https://ru.wikipedia.org/wiki/%D0%95%D0%B4%D0%B8%D0%BD%D0%B8%D1%86%D0%B0_%D0%B8%D0%B7%D0%BC%D0%B5%D1%80%D0%B5%D0%BD%D0%B8%D1%8F)  [количества информации](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D1%82%D0%B2%D0%BE_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8), равная 210 (1024) байт, она больше килобайта (1000 байт) на 24 байта.

**Битовые операции**

При разработке ISA RISC-V была проделана существенная работа по созданию предварительного, хотя и не одобренного, набора инструкций ISA (B) для битовых манипуляций. Подмножество битовых манипуляций может помочь криптографическим, графическим и математическим операциям. Критерии для включения – существенное улучшение плотности или скорости кода (сокращение инструкций, по крайней мере, 3 к 1). Версия 0.37 включала в себя инструкции для подсчета ведущих нулей, подсчета одного бита, выполнения и дополнения, сдвига, поворота, обобщенного битового реверса и перетасовки, байтовых свопов, битовых экстрактов и депозитов, а также некоторых битовых манипуляций для сжатого набора (not, neg и reverse).

**Компактный набор команд для SIMD**

Для простых экономичных систем RISC-V предлагается использовать биты регистров с плавающей точкой для выполнения параллельной одиночной инструкции с множественными данными (SIMD). Эти инструкции предназначены для ускорения мультимедиа и приложений цифровой обработки сигналов. На данное время они полностью не определены, но могут напоминать мультимедийные инструкции PA-RISC: Multimedia Acceleration eXtensions (MAX). Помимо 64-битной математики, в процессоре PA-RISC MAX2 выполняет арифметическую обработку сразу по четырем 16-битным подсловам с несколькими методами переполнения. 16-битный размер подслова был выбран для поддержки задач цифровой обработки сигналов. Эти инструкции недорогие для проектирования и производства, но увеличи-вают производительность процессора на задачах цифровой обработки сигналов более чем в 48 раз.

**Операции с векторами**

Предлагаемый набор команд векторной обработки (V) позволяет получить достаточную гибкость, чтобы процессор мог реализовать векторные инструкции в регистрах стандартного процессора. Это обеспечит минимальные реализации с аналогичной производительностью для мультимедийного ISA. Однако истинный векторный сопроцессор может выполнять тот же код с более высокой производительностью.

По состоянию на 2015 год предложение по векторной обработке представляет собой консервативную, гибкую конструкцию универсального векторного процессора смешанной точности, пригодного для выполнения вычислительных ядер. Код будет легко переноситься на процессоры с разной длиной вектора, в идеале, без перекомпиляции.

Напротив, короткие SIMD-расширения, используемые в x86, ARM и PA-RISC. менее удобны. В них изменение ширины слова вызывает изменение набора команд для расширения векторных регистров (в случае x86 - с 64-битных регистров MMX до 128-битных потоковых SIMD-расширений (SSE). Результатом является растущий набор команд и необходимость переносить рабочий код в новые инструкции..

В векторном ISA RISC-V вместо фиксирования длины вектора в архитектуре доступна команда (setvl), которая принимает запрашиваемый размер и устанавливает длину вектора равной минимуму аппаратного предела и запрашиваемого размера. Таким образом, предложение RISC-V больше похоже на векторный дизайн компьютера Cray. То есть каждый вектор, число которых достигает 32 векторов, имеет одинаковую длину.

Приложение задает общую ширину вектора, которая ему требуется, а процессор определяет длину вектора, которую он может предоставить с помощью доступных внутрикристальных ресурсов. Длина вектора ограничена доступным хранением на кристалле, деленным на количество байтов памяти, необходимых для каждой записи. За пределами векторных циклов приложение может запрашивать регистры с нулевым вектором, что позволяет операционной системе сохранять их при переключении контекста.