

Декодер инструкций RISC-V

Лабораторная работа #4

В рамках лабораторной работы необходимо реализовать устройство, декодирующее инструкции и формирующее управляющие сигналы для функциональных блоков процессора (регистровый файл, АЛУ, мультиплексоры и т.д.).

Основной ход выполнения работы выглядит так:

1. Формализация задачи в тетради
2. Реализация и верификация модуля декодера на Verilog HDL
3. Знакомство с Jupiter
4. Разработка программы на языке ассемблера RISC-V по индивидуальному заданию

Архитектура RISC-V

Архитектура системы команд (Instruction Set Architecture, ISA) включает в себя систему команд процессора (то есть все команды, выполнение которых поддерживается на аппаратном уровне) и средства для выполнения этих команд (форматы данных, наборы регистров, модель памяти).

ISA RISC-V является представителем RISC-архитектуры (Reduced Instruction Set Computer), то есть архитектуры с сокращенным набором команд. Такие процессоры поддерживают относительно небольшой набор инструкций, которые имеют фиксированную длину, за счет чего устройство управления получается очень простым. Каждая команда из базового набора инструкций RISC-V кодируется 32-битным числом (словом).

Процессор RISC-V обладает регистровым файлом с 32 регистрами общего назначения по 32 бита каждый, при этом по адресу 0 располагается константное значение нуля.

Из основной памяти, подключенной к процессору, можно считывать и записывать байты, полуслова (16-битные числа) и слова (32-битные числа). Память имеет побайтовую адресацию и 32-битный адресный вход. RISC-V является load/store архитектурой, это значит, что для выполнения вычислений над данными их надо предварительно разместить в регистровом файле, для чего используются специальные команды загрузки данных из основной памяти. Результат вычисления также может быть записан только в регистровый файл, поэтому есть специальные команды сохранения данных из регистрового файла в основную память.

Набор инструкций RISC-V

Все инструкции архитектуры RISC-V можно условно разделить на три категории:

1. Вычислительные инструкции
 - Используемые в качестве операндов два регистра
 - Используемые в качестве операндов регистр и непосредственный операнд
2. Инструкции загрузки и сохранения данных
3. Инструкции управления программой
 - Условный переход
 - Безусловный переход

Ниже приводится список базовых целочисленных инструкций RISC-V, выполнение которых должно поддерживаться разрабатываемым процессором.

Базовый набор целочисленных инструкций RV32I

| Instr | Название | Функция | Описание | Формат | Opcode | Func3 | Func7 | Пример использования |
|--------|-------------------------------------|--|------------------------------------|--------|---------|-------|-------|--|
| add | ADDITION | Сложение | $rd = rs1 + rs2$ | R | 0110011 | 0x0 | 0x00 | op <i>rd</i> , <i>rs1</i> , <i>rs2</i> xor <i>x2</i> , <i>x5</i> , <i>x6</i> sll <i>x7</i> , <i>x11</i> , <i>x12</i> |
| sub | SUBtraction | Вычитание | $rd = rs1 - rs2$ | | | 0x0 | 0x20 | |
| xor | eXclusive OR | Исключающее ИЛИ | $rd = rs1 \wedge rs2$ | | | 0x4 | 0x00 | |
| or | OR | Логическое ИЛИ | $rd = rs1 \vee rs2$ | | | 0x6 | 0x00 | |
| and | AND | Логическое И | $rd = rs1 \& rs2$ | | | 0x7 | 0x00 | |
| sll | Shift Left Logical | Логический сдвиг влево | $rd = rs1 \ll rs2$ | | | 0x1 | 0x00 | |
| srl | Shift Right Logical | Логический сдвиг вправо | $rd = rs1 \gg rs2$ | | | 0x5 | 0x00 | |
| sra | Shift Right Arithmetic | Арифметический сдвиг вправо | $rd = rs1 \ggg rs2$ | | | 0x5 | 0x20 | |
| slt | Set Less Than | Результат сравнения $A < B$ | $rd = (rs1 < rs2) ? 1 : 0$ | | | 0x2 | 0x00 | |
| sltu | Set Less Than Unsigned | Беззнаковое сравнение $A < B$ | $rd = (rs1 < rs2) ? 1 : 0$ | | | 0x3 | 0x00 | |
| addi | ADDITION Immediate | Сложение с константой | $rd = rs1 + imm$ | I | 0010011 | 0x0 | - | op <i>rd</i> , <i>rs1</i> , <i>imm</i> addi <i>x6</i> , <i>x3</i> , -12 ori <i>x3</i> , <i>x1</i> , 0x8F |
| xori | eXclusive OR Immediate | Исключающее ИЛИ с константой | $rd = rs1 \wedge imm$ | | | 0x4 | - | |
| ori | OR Immediate | Логическое ИЛИ с константой | $rd = rs1 \vee imm$ | | | 0x6 | - | |
| andi | AND Immediate | Логическое И с константой | $rd = rs1 \& imm$ | | | 0x7 | - | |
| slli | Shift Left Logical Immediate | Логический сдвиг влево | $rd = rs1 \ll imm$ | | | 0x1 | 0x00 | |
| srl | Shift Right Logical Immediate | Логический сдвиг вправо | $rd = rs1 \gg imm$ | | | 0x5 | 0x00 | |
| srai | Shift Right Arithmetic Immediate | Арифметический сдвиг вправо | $rd = rs1 \ggg imm$ | | | 0x5 | 0x20 | |
| slti | Set Less Than Immediate | Результат сравнения $A < B$ | $rd = (rs1 < imm) ? 1 : 0$ | | | 0x2 | - | |
| sltiu | Set Less Than Immediate Unsigned | Беззнаковое сравнение $A < B$ | $rd = (rs1 < imm) ? 1 : 0$ | | | 0x3 | - | |
| lb | Load Byte | Загрузить байт из памяти | $rd = SE(Mem[rs1 + imm][7:0])$ | I | 0000011 | 0x0 | - | op <i>rd</i> , <i>imm</i> (<i>rs1</i>) lh <i>x1</i> , 8(<i>x5</i>) |
| lh | Load Half | Загрузить полуслово из памяти | $rd = SE(Mem[rs1 + imm][15:0])$ | | | 0x1 | - | |
| lw | Load Word | Загрузить слово из памяти | $rd = SE(Mem[rs1 + imm][31:0])$ | | | 0x2 | - | |
| lbu | Load Byte Unsigned | Загрузить беззнаковый байт из памяти | $rd = Mem[rs1 + imm][7:0]$ | | | 0x4 | - | |
| lhu | Load Half Unsigned | Загрузить беззнаковое полуслово из памяти | $rd = Mem[rs1 + imm][15:0]$ | | | 0x5 | - | |
| sb | Store Byte | Сохранить байт в память | $Mem[rs1 + imm][7:0] = rs2[7:0]$ | S | 0100011 | 0x0 | - | op <i>rs2</i> , <i>imm</i> (<i>rs1</i>) sw <i>x1</i> , 0xCF(<i>x12</i>) |
| sh | Store Half | Сохранить полуслово в память | $Mem[rs1 + imm][15:0] = rs2[15:0]$ | | | 0x1 | - | |
| sw | Store Word | Сохранить слово в память | $Mem[rs1 + imm][31:0] = rs2[31:0]$ | | | 0x2 | - | |
| beq | Branch if Equal | Перейти, если $A == B$ | $if (rs1 == rs2) PC += imm$ | B | 1100011 | 0x0 | - | comp <i>rs1</i> , <i>rs2</i> , <i>imm</i> beq <i>x8</i> , <i>x9</i> , <i>offset</i> bltu <i>x20</i> , <i>x21</i> , 0xFC |
| bne | Branch if Not Equal | Перейти, если $A \neq B$ | $if (rs1 \neq rs2) PC += imm$ | | | 0x1 | - | |
| blt | Branch if Less Than | Перейти, если $A < B$ | $if (rs1 < rs2) PC += imm$ | | | 0x4 | - | |
| bge | Branch if Greater or Equal | Перейти, если $A \geq B$ | $if (rs1 \geq rs2) PC += imm$ | | | 0x5 | - | |
| bltu | Branch if Less Than Unsigned | Перейти, если $A < B$ беззнаковое | $if (rs1 < rs2) PC += imm$ | | | 0x6 | - | |
| bgeu | Branch if Greater or Equal Unsigned | Перейти, если $A \geq B$ беззнаковое | $if (rs1 \geq rs2) PC += imm$ | | | 0x7 | - | |
| jal | Jamp And Link | Переход с сохранением адреса возврата | $rd = PC + 4; PC += imm$ | J | 1101111 | - | - | jal <i>x1</i> , <i>offset</i> jalr <i>x1</i> , 0(<i>x5</i>) |
| jalr | Jamp And Link Register | Переход по регистру с сохранением адреса возврата | $rd = PC + 4; PC = rs1 + imm$ | I | 1100111 | 0x0 | - | |
| lui | Load Upper Immediate | Загрузить константу в сдвинутую на 12 | $rd = imm \ll 12$ | U | 0110111 | - | - | lui <i>x3</i> , 0xFFFFF auipc <i>x2</i> , 0x000FF |
| auipc | Add Upper Immediate to PC | Сохранить счетчик команд в сумме с константой $\ll 12$ | $rd = PC + (imm \ll 12)$ | | 0010111 | - | - | |
| ecall | Environment CALL | Передача управления операционной системе | Воспринимать как nop | I | 1110011 | - | - | - |
| ebreak | Environment BREAK | Передача управления отладчику | | | | | | |

Псевдоинструкции

Ассемблер RISC-V поддерживает псевдоинструкции – псевдонимы реальных базовых инструкций для упрощения написания кода. Ниже приводится таблица с некоторыми псевдоинструкциями, указанием реальной инструкции, на которую будет заменен псевдоним, и пояснением ее функции.

| Псевдоинструкция | Базовая инструкция | Смысл |
|-----------------------|------------------------------|---------------------------------------|
| nop | addi x0, x0, 0 | Нет операции |
| li rd, immediate | Различные последовательности | Загрузка константы |
| mv rd, rs | addi rd, rs, 0 | Копирование регистров |
| not rd, rs | xori rd, rs, -1 | Инверсия числа |
| neg rd, rs | sub rd, x0, rs | Изменение знака числа |
| seqz rd, rs | sltiu rd, rs, 1 | Установить 1, если == 0 |
| snez rd, rs | sltu rd, x0, rs | Установить 1, если != 0 |
| sltz rd, rs | slt rd, rs, x0 | Установить 1, если < 0 |
| sgtz rd, rs | slt rd, x0, rs | Установить 1, если > 0 |
| beqz rs, offset | beq rs, x0, offset | Перейти, если == 0 |
| bnez rs, offset | bne rs, x0, offset | Перейти, если != 0 |
| blez rs, offset | bge x0, rs, offset | Перейти, если <= 0 |
| bgez rs, offset | bge rs, x0, offset | Перейти, если >= 0 |
| bltz rs, offset | blt rs, x0, offset | Перейти, если < 0 |
| bgtz rs, offset | blt x0, rs, offset | Перейти, если > 0 |
| bgt rs1, rs2, offset | blt rs2, rs1, offset | Перейти, если > |
| ble rs1, rs2, offset | bge rs2, rs1, offset | Перейти, если <= |
| bgtu rs1, rs2, offset | bltu rs2, rs1, offset | Перейти, если >, беззнаковое |
| bleu rs1, rs2, offset | bgeu rs2, rs1, offset | Перейти, если <=, беззнаковое |
| j offset | jal x0, offset | Переход по метке |
| jal offset | jal x1, offset | Переход с сохранением адреса возврата |
| jr rs | jalr x0, 0(rs) | Переход по значению из регистра |
| jalr rs | jalr x1, 0(rs) | Переход с сохранением адреса возврата |
| ret | jalr x0, x1, 0 | Возврат из подпрограммы |

Неподдерживаемые инструкции

В базовом наборе инструкций RISC-V к операциям SYSTEM относятся ECALL и EBREAK, к операциям MISC-MEM – операция FENCE. В реализуемом процессорной ядре эти инструкции не должны приводить ни к каким изменениям. Иначе говоря, они должны быть реализованы как инструкция NOP (no operation).

Инструкция FENCE в RISC-V необходима при работе с несколькими аппаратными потоками, или хартами (hart – «hardware thread»). В RISC-V используется расслабленная модель памяти (relaxed memory model): потоки «видят» все инструкции чтения и записи, которые выполняются другими потоками, однако видимый порядок этих инструкций может отличаться от реального. Инструкция FENCE, использованная между двумя инструкциями чтения и/или записи гарантирует, что остальные потоки увидят первую инструкцию перед второй. Реализация FENCE является опциональной в RISC-V и в данном случае в ней нет необходимости, так как в системе не предполагается наличия нескольких аппаратных потоков.

Инструкции ECALL и EBREAK должны вызывать исключение с последующим переходом в обработчик исключения (вызова операционной системы и перехода в привилегированный режим работы). Помимо этого, их вызов должен обновить содержимое некоторых управляющих регистров (control-status registers – CSR). Так как реализация CSR и обработки исключений для операционной системы для разрабатываемого устройства не предусмотрены, то эти инструкции реализуются как NOP.

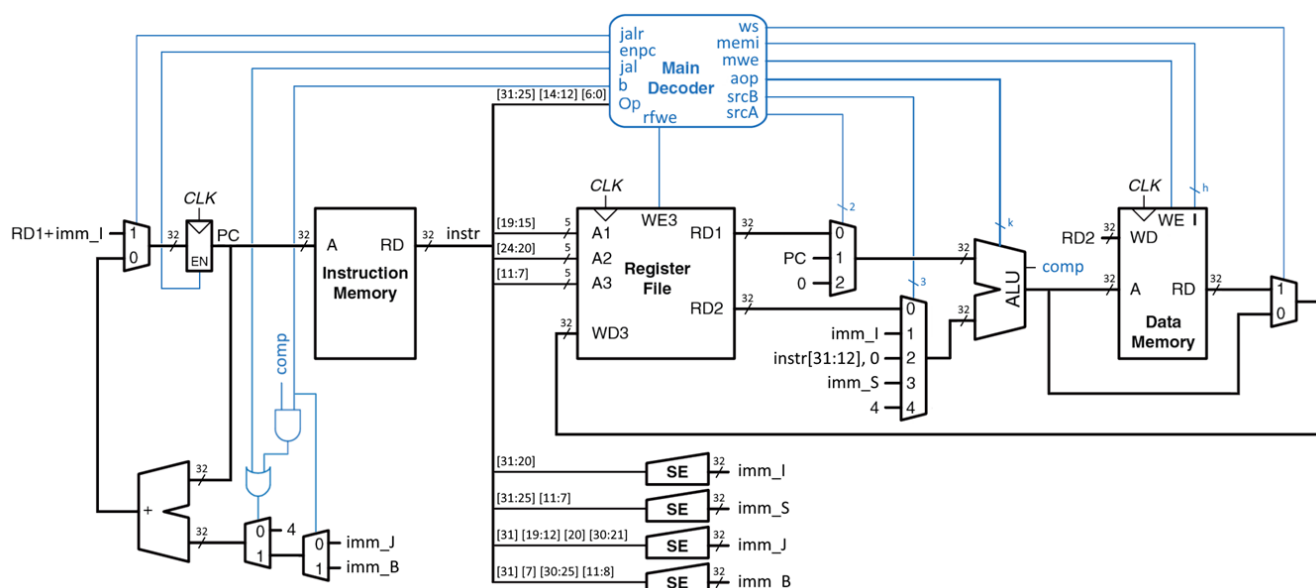
Кодирование инструкций RISC-V

Базовый набор целочисленных инструкций использует шесть форматов кодирования команд. Младшие 7 бит любой инструкции – это поле **opcode** (код операции), по коду в этом поле устройство управления (в частности декодер инструкций) определяет какая именно инструкция должна быть выполнена и каким из шести способов она закодирована.

Некоторые инструкции также имеют поля **funct3** и **funct7**, в которых кодируется уточняющая информация касательно выполняемой операции. Некоторые форматы инструкций кодируют адреса регистров: **rs1** и **rs2** – адреса регистров-операндов, **rd** – адрес регистра назначения.

| 31 | 27 | 26 | 25 | 24 | 20 | 19 | 15 | 14 | 12 | 11 | 7 | 6 | 0 | |
|-----------------------|----|----|----|-----|----|-----|----|--------|----|-------------|---|--------|---|--------|
| funct7 | | | | rs2 | | rs1 | | funct3 | | rd | | opcode | | R-type |
| imm[11:0] | | | | | | rs1 | | funct3 | | rd | | opcode | | I-type |
| imm[11:5] | | | | rs2 | | rs1 | | funct3 | | imm[4:0] | | opcode | | S-type |
| imm[12:10:5] | | | | rs2 | | rs1 | | funct3 | | imm[4:1:11] | | opcode | | B-type |
| imm[31:12] | | | | | | | | | | rd | | opcode | | U-type |
| imm[20:10:1:11:19:12] | | | | | | | | | | rd | | opcode | | J-type |

Микроархитектура процессора



Дешифратор команд RISC-V

Декодер инструкций в процессоре служит для преобразования инструкции в набор управляющих сигналов, необходимых для ее исполнения.

Например, для выполнения инструкции загрузки слова из памяти данных в регистровый файл `lw`, декодер должен направить в АЛУ два операнда и код операции АЛУ (сложение) для вычисления адреса. Для модуля загрузки-выгрузки из памяти (load-store unit – интерфейс взаимодействия с памятью) декодер передает запрос на считывание данных (`mem_req_o`) и размер слова (`mem_size_o`). Наконец, декодер решает, будут ли записаны данные в регистровый файл (`gpr_we_a_o`).

Управляющие сигналы на выходе декодера зависят от трех полей инструкции: **opcode**, **funct3** и **funct7**. Обратите внимание, что расположение этих полей одинаково для всех типов инструкций. Это сделано для удобства декодирования. При этом поля **funct3** и **funct7** могут отсутствовать для некоторых инструкций.

Прототип разрабатываемого устройства и описание его сигналов представлены ниже.

```
module riscv_decode
(
    input    [31:0]    fetched_instr_i,
    output   [1:0]     ex_op_a_sel_o,
    output   [2:0]     ex_op_b_sel_o,
    output   [`ALU_OP_WIDTH-1:0] alu_op_o,
    output   mem_req_o,
    output   mem_we_o,
    output   [2:0]     mem_size_o,
    output   gpr_we_a_o,
    output   wb_src_sel_o,
    output   illegal_instr_o,
    output   branch_o,
    output   jal_o,
    output   jalr_o
);
```

| Название | Пояснение |
|------------------------------|--|
| <code>fetched_instr_i</code> | Инструкция для декодирования, считанная из памяти инструкций |
| <code>ex_op_a_sel_o</code> | Управляющий сигнал мультиплексора для выбора первого операнда АЛУ |
| <code>ex_op_b_sel_o</code> | Управляющий сигнал мультиплексора для выбора второго операнда АЛУ |
| <code>alu_op_o</code> | Операция АЛУ |
| <code>mem_req_o</code> | Запрос на доступ к памяти (часть интерфейса памяти) |
| <code>mem_we_o</code> | Сигнал разрешения записи в память, «write enable» (при равенстве нулю происходит чтение) |
| <code>mem_size_o</code> | Управляющий сигнал для выбора размера слова при чтении-записи в память (часть интерфейса памяти) |
| <code>gpr_we_a_o</code> | Сигнал разрешения записи в регистровый файл |
| <code>wb_src_sel_o</code> | Управляющий сигнал мультиплексора для выбора данных, записываемых в регистровый файл |
| <code>illegal_instr_o</code> | Сигнал о некорректной инструкции (на схеме не отмечен) |
| <code>branch_o</code> | Сигнал об инструкции условного перехода |
| <code>jal_o</code> | Сигнал об инструкции безусловного перехода <code>jal</code> |
| <code>jalr_o</code> | Сигнал об инструкции безусловного перехода <code>jalr</code> |

Управляющие сигналы мультиплексоров, которые формирует декодер, всегда должны иметь валидные (действительные) значения, то есть те, которые однозначно определяют выход мультиплексора.

В системе команд RV32I два младших бита поля **opcode** всегда равны 2'b11, таким образом декодер понимает, что будут исполняться именно 32-битные инструкции, а не 16-битные, например. Декодер должен поднять сигнал `illegal_instr_o` в случае:

- неравенства двух младших битов opcode значению 2'b11;
- некорректного значения **funct3** или **funct7** для данной операции;
- если значение **opcode** не совпадает ни с одним из известных и следовательно операция не определена.

Декодер удобнее описывать разбив все инструкции на однотипные группы, как это сделано ниже.

| Операция | Opcode | Описание операции | Краткая запись |
|----------|--------|--|------------------------------------|
| LOAD | 00000 | Записать в rd данные из памяти по адресу rs1+imm | $rd = Mem[rs1 + imm]$ |
| MISC_MEM | 00011 | Не производить операцию <code>illegal_instr_o = 0</code> | |
| OP_IMM | 00100 | Записать в rd результат вычисления АЛУ над rs1 и imm | $rd = alu_op(rs1, imm)$ |
| AUIPC | 00101 | Записать в rd результат сложения непосредственного операнда U-типа (imm_u) и счетчика команд | $rd = PC + (imm \ll 12)$ |
| STORE | 01000 | Записать в память по адресу rs1+imm данные из rs2 | $Mem[rs1 + imm] = rs2$ |
| OP | 01100 | Записать в rd результат вычисления АЛУ над rs1 и rs2 | $rd = alu_op(rs1, rs2)$ |
| LUI | 01101 | Записать в rd значение непосредственного операнда U-типа (imm_u) | $rd = imm \ll 12$ |
| BRANCH | 11000 | Увеличить счетчик команд на значение imm, если верен результат сравнения rs1 и rs2 | $if\ cmp_op(rs1, rs2)\ PC += imm$ |
| JALR | 11001 | Записать в rd следующий адрес счетчика команд, в счетчик команд записать rs1+imm | $rd = PC + 4; PC = rs1 + imm$ |
| JAL | 11011 | Записать в rd следующий адрес счетчика команд, увеличить счетчик команд на значение imm | $rd = PC + 4; PC += imm$ |
| SYSTEM | 11100 | Не производить операцию <code>illegal_instr_o = 0</code> | |

Интерфейс памяти

Интерфейс памяти использует несколько сигналов для взаимодействия с памятью: `mem_req_o` (этот выход должен быть выставлен в 1 каждый раз, когда необходимо обратиться к памяти – считать или записать), `mem_we_o` (выставляется в 1, если необходимо записать данные в память, и 0 – если считать из памяти) и `mem_size_o` (указывающий размер порции данных необходимых для передачи; возможные значения указаны ниже).

| Название | Значение | Пояснение |
|----------|----------|--------------------------------|
| LDST_B | 3'd0 | Знаковое 8-битное значение |
| LDST_H | 3'd1 | Знаковое 16-битное значение |
| LDST_W | 3'd2 | 32-битное значение |
| LDST_BU | 3'd4 | Беззнаковое 8-битное значение |
| LDST_HU | 3'd5 | Беззнаковое 16-битное значение |