Условные переходы

Команды перехода

- Безусловный переход (unconditional branch/jump) — команда на изменение порядка выполнения инструкций
  - RV32I: `JAL`, `JALR`
- Условный переход (conditional branch/jump) — команда на изменение порядка выполнения инструкций в соответствии с результатом проверки условия
  - RV32I: `BEQ`, `BNE`, `BLT`, `BGE`, `BLTU`, `BGEU`

Команды условного перехода в RV32I

imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	**Type-SB**
simm[12|10:5]	rs2	rs1	000	simm[4:1|11]	1100011	BEQ rs1, rs2, offset
simm[12|10:5]	rs2	rs1	001	simm[4:1|11]	1100011	BNE rs1, rs2, offset
simm[12|10:5]	rs2	rs1	100	simm[4:1|11]	1100011	BLT rs1, rs2, offset
simm[12|10:5]	rs2	rs1	101	simm[4:1|11]	1100011	BGE rs1, rs2, offset
simm[12|10:5]	rs2	rs1	110	simm[4:1|11]	1100011	BLTU rs1, rs2, offset
simm[12|10:5]	rs2	rs1	111	simm[4:1|11]	1100011	BGEU rs1, rs2, offset

Если для 32-битных слов из регистров `rs1` и `rs2` выполнено соответствующее условие, то к счетчику команд `pc` прибавляется знаковое смещение `offset`.

- Поле `funct3` определяет условие
- Поле `imm` содержит знаковое смещение в 2-байтных словах
- Поля `rs1` и `rs2` содержат номера регистров-источников для сравнения

Машинный код	Код на языке ассемблера RISC-V
	`_loop:`
`00138393`	`addi x7, x7, 1`
`fe039ee3`	`bne x7, x0, _loop`

Устройство управления

```
module control(
    ..................
    output reg branch
);

always @(*) begin
    ..............
    branch = 1'b0;

    casez ({funct5, funct2, funct3, opcode})
        ....................................
        17'b?????_??_001_1100011: begin // BNE
            imm12 = {instr[31], instr[31], instr[7], instr[30:25], instr[11:9]};
            alu_op = 3'b100;
            branch = 1'b1;
        end
endcase
```

Ядро

```
.............................................................
wire [31:0]pc_target = branch_taken ? branch_target : (pc + 1);
wire [31:0]pc_next = (pc == last_pc) ? pc : pc_target;

.............................

wire cmp_res = alu_result != 0;
wire branch_taken = branch & cmp_res;
wire [31:0]branch_target = pc + imm32;
wire branch;

control control(
    ..............
    .branch(branch)
);
```

Числа Фибоначчи

```
.text
.globl _start
.globl _finish

_start:
    li      t0, 0   # fib(0)
    li      t1, 1   # fib(1)

    li      t3, 1   # n counter
    li      t4, 12  # max n counter

    ...........

_finish:
    nop
.rept 20
    nop
.endr
```

```
_next:
    # fib(n) = fib(n - 1) + fib(n - 2)
    add     t2, t1,     t0

    sw      t2, 0x20(zero) # display fib(n)

    mv      t0, t1
    mv      t1, t2

    addi    t3, t3,     1  # increment counter
    bne     t3, t4,     _next   # next iteration
```

Симуляция в Icarus Verilog

```
           ...........
[pc = 00000005] 005303b3
taken = 0
[pc = 00000006] 02702023
taken = 0
(SW) funct3 = 2, opcode = 23
[00000020] <- 00000003
[pc = 00000007] 00030293
taken = 0
[pc = 00000008] 00038313
taken = 0
[pc = 00000009] 001e0e13
taken = 0
[pc = 0000000a] ffde16e3
taken = 1 target = 00000005
(BNE) funct3 = 1, opcode = 63
```

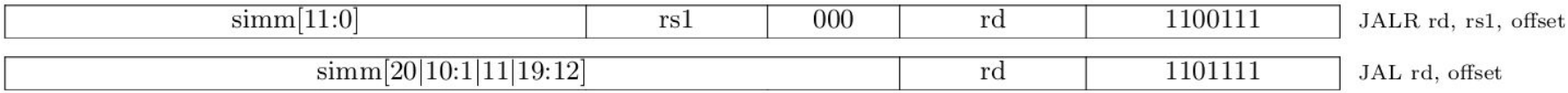
```
[pc = 00000005] 005303b3
taken = 0
[pc = 00000006] 02702023
taken = 0
(SW) funct3 = 2, opcode = 23
[00000020] <- 00000005
[pc = 00000007] 00030293
taken = 0
[pc = 00000008] 00038313
taken = 0
[pc = 00000009] 001e0e13
taken = 0
[pc = 0000000a] ffde16e3
taken = 1 target = 00000005
(BNE) funct3 = 1, opcode = 63
```

```
[pc = 00000005] 005303b3
taken = 0
[pc = 00000006] 02702023
taken = 0
(SW) funct3 = 2, opcode = 23
[00000020] <- 00000008
[pc = 00000007] 00030293
taken = 0
[pc = 00000008] 00038313
taken = 0
[pc = 00000009] 001e0e13
taken = 0
[pc = 0000000a] ffde16e3
taken = 1 target = 00000005
(BNE) funct3 = 1, opcode = 63
           ...........
```

Задержка

```
    .............
    li      t5, 0   # init delay counter

_next:
    add     t2, t1,     t0  # fib(n) = fib(n - 1) + fib(n - 2)
    sw      t2, 0x20(zero)  # display fib(n)

_delay:
    addi    t5, t5,     128     # increment delay counter, assume t5 = 0
    bne     t5, zero,   _delay  # next delay loop (~700ms)

    mv      t0, t1
    mv      t1, t2
    addi    t3, t3,     1       # increment counter
    bne     t3, t4,     _next   # next iteration

    ........................
```

Команды безусловного перехода в RV32I



simm[11:0]	rs1	000	rd	1100111	JALR rd, rs1, offset
simm[20|10:1|11|19:12]			rd	1101111	JAL rd, offset

Инструкция `JAL` сохраняет адрес следующей инструкции (`pc + 4`) в регистр `rd` и передает управление на адрес `pc + offset`.

- Поле `simm` содержит знаковое смещение, рассчитанное в 2-байтных словах

Инструкция `JALR` сохраняет адрес следующей инструкции (`pc + 4`) в регистр `rd` и передает управление на адрес из регистра `rs1` со смещением `offset`.

- Поле `simm` содержит знаковое смещение, рассчитанное в 2-байтных словах
- Целевой адрес (`rs1 + offset`) выравнивается на границу 2-байтного слова в меньшую сторону

Числа Фибоначчи на С

Файл fib.c

```c
typedef unsigned int uint32_t;
typedef unsigned char uint8_t;

void main() {
    uint32_t first = 0, second = 1, next, i = 0;

    for (i = 0; i != 12; i++) {
        next = first + second;
        *(volatile uint32_t *)((uint8_t *)0x20) = next;
        first = second;
        second = next;
    }
}
```

Файл loader.s

```
.text
.globl _start
.globl _finish
.globl main

_start:
    call main

_finish:
    nop
```

```
$ riscv64-linux-gnu-gcc -nostdlib -fomit-frame-pointer -O1 -mabi=ilp32 -march=rv32i -c fib.c -o fib.o
$ riscv64-linux-gnu-as -march=rv32i -mabi=ilp32 -c loader.s -o loader.o
$ riscv64-linux-gnu-ld -Ttext=0x1000 -melf32lriscv loader.o fib_riscv.o -o fib.out
```

objdump

```
$ riscv64-linux-gnu-objdump -d fib.out
00001000 <_start>:
    1000:  008000ef              jal   ra, 1008 <main>

00001004 <_finish>:
    1004:  00000013              nop

00001008 <main>:
    1008:  00c00713              li    a4, 12
    100c:  00100793              li    a5, 1
    1010:  00000613              li    a2, 0
    1014:  00f606b3              add   a3, a2, a5
    1018:  02d02023              sw    a3, 32(zero) # 20 <_start-0xfe0>
    101c:  fff70713              addi  a4, a4, -1
    1020:  00078613              mv    a2, a5
    1024:  00068793              mv    a5, a3
    1028:  fe0716e3              bnez  a4, 1014 <main+0xc>
    102c:  00008067              ret
```

GitHub

github.com/viktor-prutyanov/drec-fpga-intro