Лабораторная работа №2	10	2023
Построение логических схем	Лихтар Анна Викторовна	

Инструментарий и требования к работе: работа выполняется в среде моделирования Logisim-evolution и Icarus Verilog 10 и новее. В работе используется Logisim-evolution v3.8.0 и Icarus Verilog version 12.0.

Ссылка на репозиторий: https://github.com/skkv-mkn/mkn-comp-arch-2023-circuit-likhhtar

Logisim

Была реализована версия easy.

Схема stack представлена на рис.1 (для удобства восприятия, разбила картинку на две).

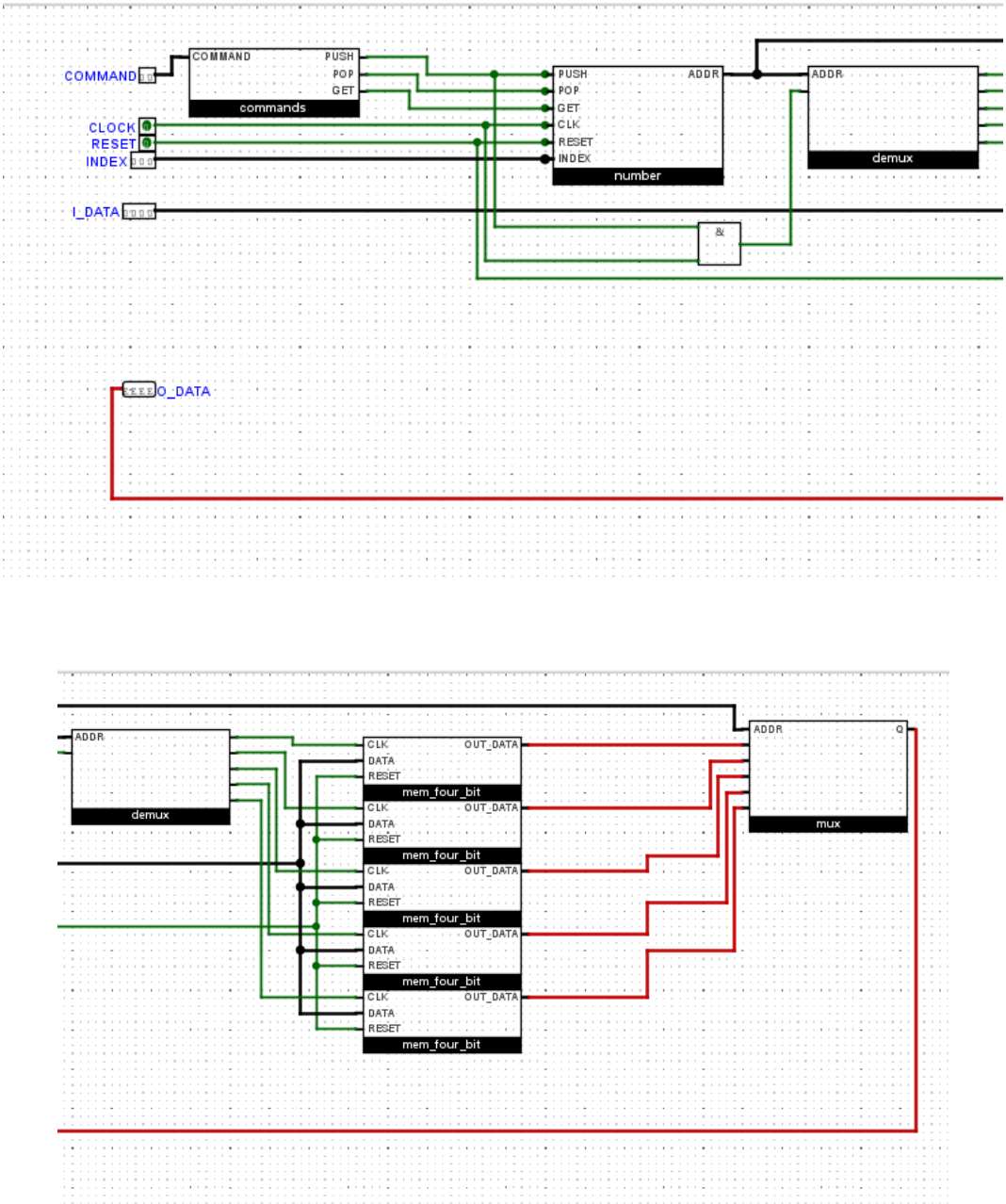

рис.1. Схема stack

Сначала вызывается парсер команды (рис. 2.), который выдает, какая из операций была включена: push, pop, get. Схема достаточно тривиальная, она просто проверяет, если на вход подается 01, то возвращает 1 для Push, если 10 – для Pop, если 11 – для Get.

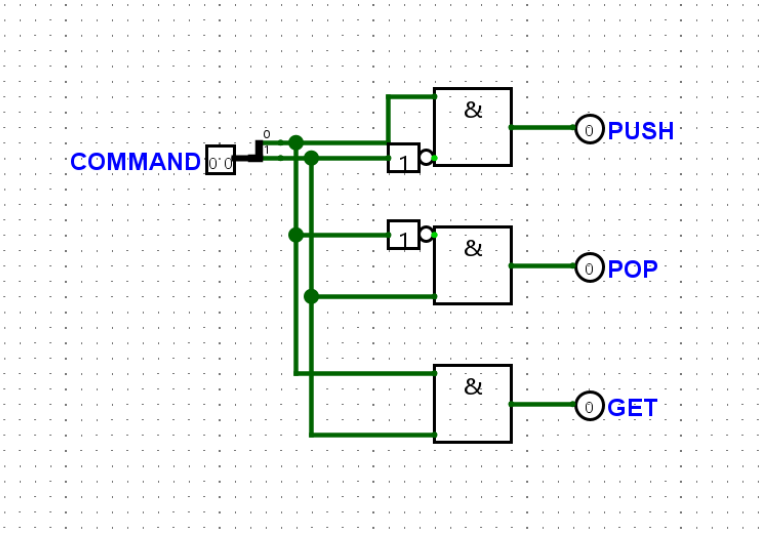

рис.2. Схема commands

На самом деле, вид оставшейся части stack очень похож на то, что было показано на паре, только теперь хранятся не однобитные значения, а четырехбитные, а также добавлена схема, которая поддерживает указатель на вершину стека.

Начнем со схемы, которая поддерживает указатель на вершину стека и возвращает адрес, по которому находится ячейка необходимая для соответствующей операции (рис. 3). В нее встроена схема totalizer (рис. 4), которая возвращает указатель на голову стека в данный момент, затем изменяет его в зависимости от операции (увеличивает при push, уменьшает при pop). Если была вызвана операция push, то необходимо вызвать схему plus_one (рис. 5), которая прибавит 1 к голове стека, если была вызвана операция pop, то голова стека и есть необходимая ячейка, если была вызвана операция get, то вызывается схема get_pos_for_get (рис. 6), которая отсчитает необходимое количество ячеек от головы и вернет необходимый индекс.

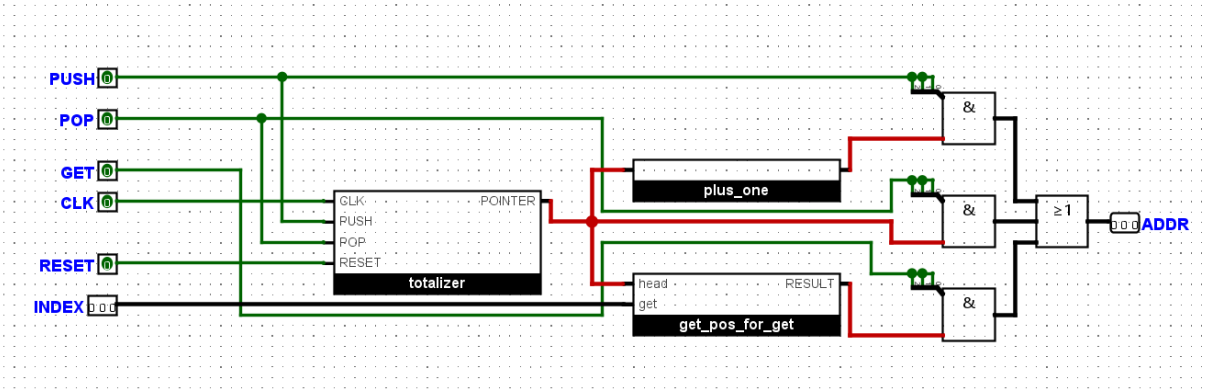

Рис. 3. Схема number

Теперь рассмотрим схему totalizer. Модуль change_number (рис. 7) возвращает номера следующей и предшествующей ячейки для вершины стека, которая хранится в mem_3_bit (рис. 8). После операций and и or в mem_3_bit отправляется номер нынешней вершины стека. Значение POINTER изменится только после выключения синхронизации. Это сделано таким образом, потому что при вызове pop должно сначала возвращаться значение головы, а только потом перемещаться указатель. Случаи для push и get в схеме number обрабатываются с помощью схем plus_one и get_pos_for_get.

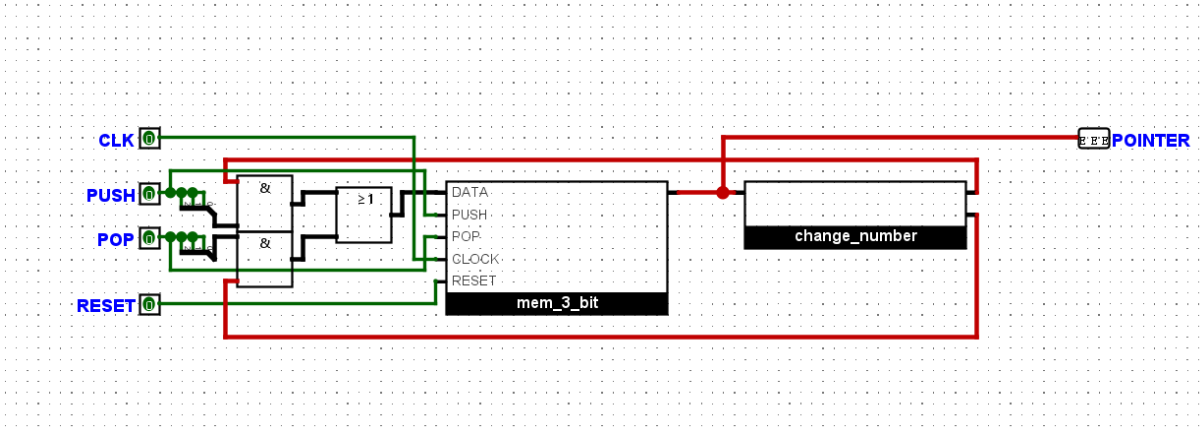

Рис. 4. Схема totalizer

В mem_3_bit DATA просто разбивается на три d_trigger (рис. 9), которые и будут нам гарантировать delay для POINTER. Последовательность логических операций, которая находится вначале, определяет, нужно ли нам вообще менять указатель. Его нужно изменить только при включенной синхронизации при push/pop.

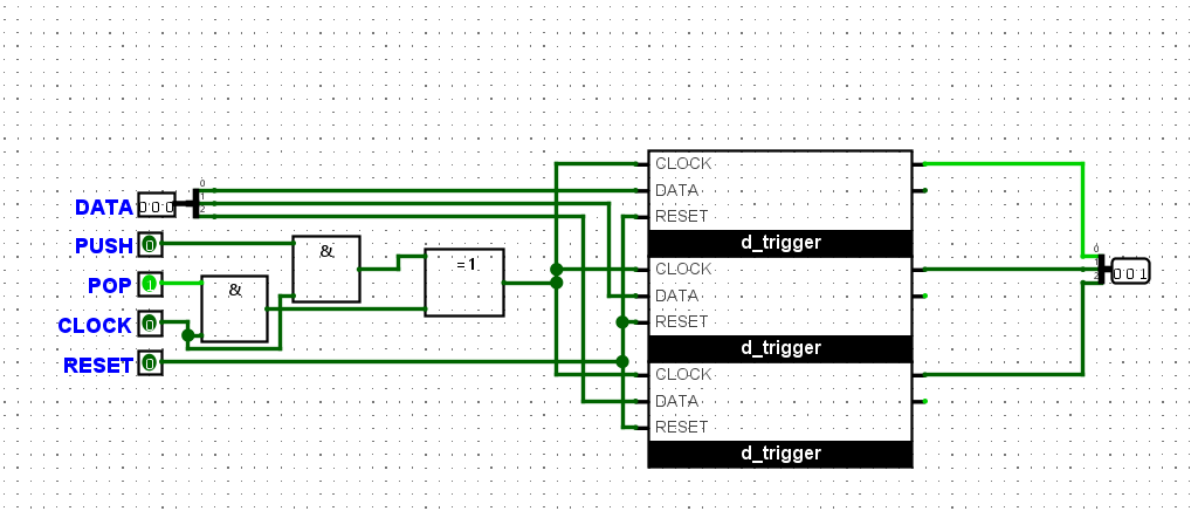

Рис. 8. Схема mem_3_bit

Чтобы гарантировать delay используется два синхронизированных d-триггера. Тогда, если включен RESET, то теперь будет хранится 0. При включении синхронизации значение DATA передастся в оба d-триггера (рис. 10), но во втором синхронизация будет равна 0, таким образом, пока CLOCK не станет 0, во втором d-триггере будет хранится предыдущее значение.

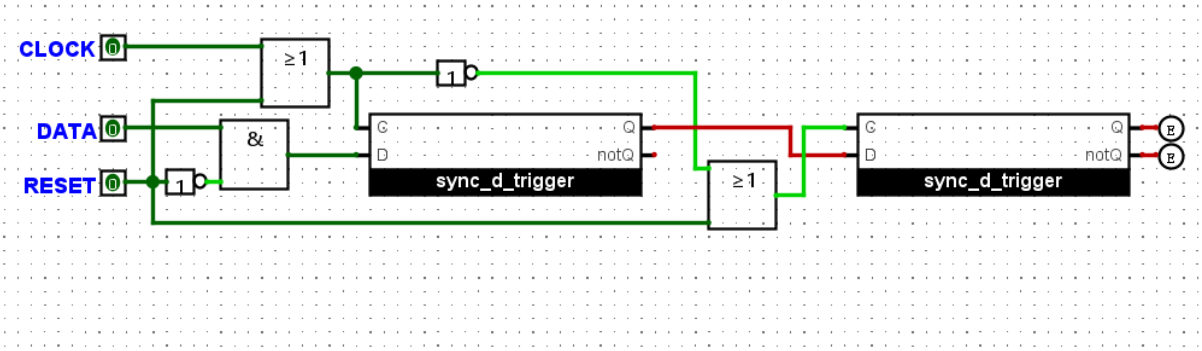

Рис. 9. Схема d_trigger

D-триггер при подаче на вход D 1 возвращает также 1, а при 0 – 0. Но это не все, иначе было бы неинтересно. Если синхронизация выключена, то он не читает вход, а сохраняет значение. Для этого мы используем RS-триггер (рис. 11).

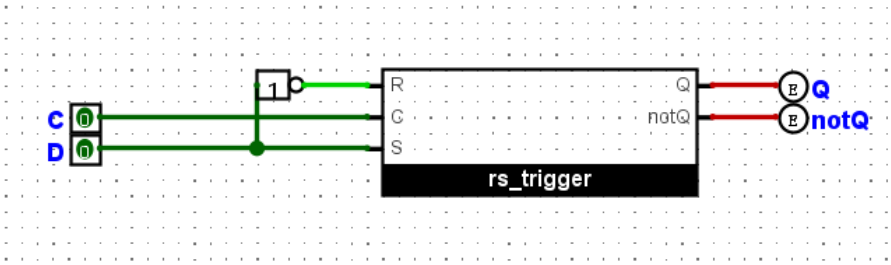

Рис. 10. Схема sync_d_trigger

В этой схеме представлен синхронизированный RS-триггер. Если убрать операторы and, то получится обычный RS-триггер, которому соответствует следующая таблица истинности.

a	b	q
0	0	сохранение того значения, которое было
0	1	0
1	0	1
1	1	не важно

Таким образом, если синхронизация выключена, то сохраняется значение, а если нет, то, если на S подается 1, выдается 1, а если 0 – 0. То, что нужно.

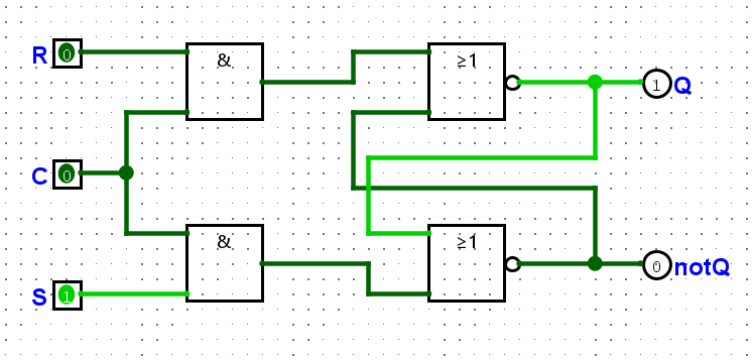

Рис. 11. Схема rs_trigger

Вернемся к получению номеров ячеек для POP, PUSH и GET. На вход схеме подается номер ячейки-головы, на выход – номера следующей и предыдущей ячеек соответственно.

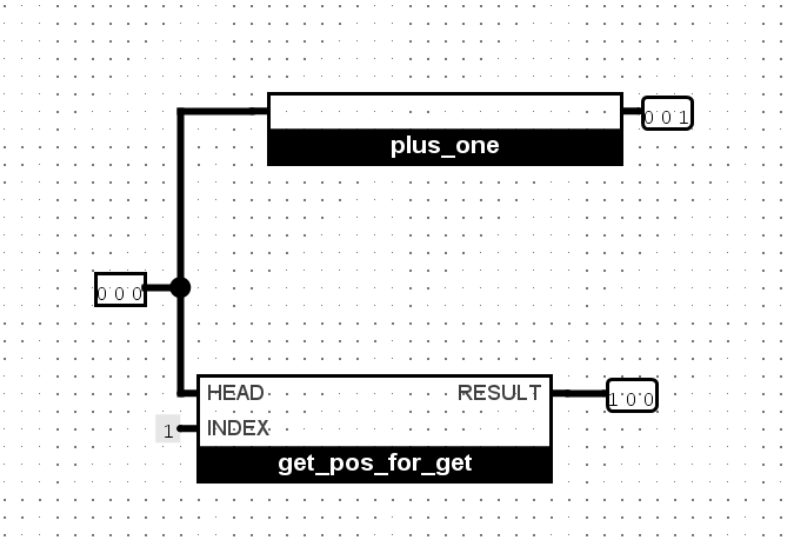

Рис.7. Схема Change_number

В этой схеме число дважды берется по модулю 5 (Рис. 12): до прибавления единицы (рис. 13) и после.

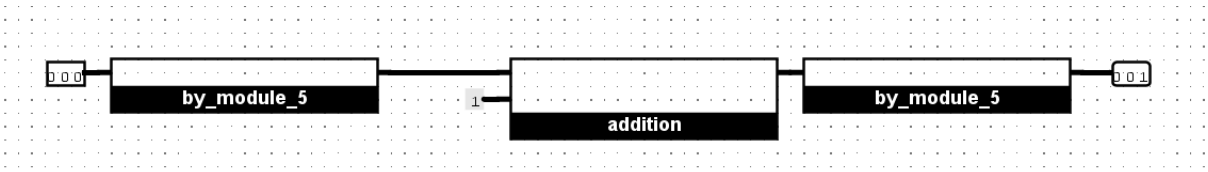

Рис. 5. Схема Plus_one

Модуль three_to_five (рис. 14) это переделанная схема 3to8 из лекции (в лекции было 8 ячеек памяти, нам дано 5). А модуль five_to_three (рис. 15) переводит число обратно в трехбитное число.

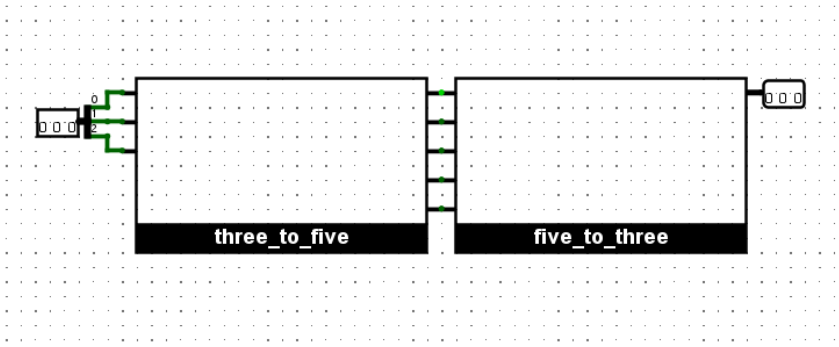

Рис. 12. Схема By_module_5

На первый взгляд выглядит страшно. Это связано с тем, что это была одна из первых схем, которые я реализовала в logisim. На самом деле, страшного здесь ничего нет. Эта схема возвращает 1 для ячейки, к которой мы должно обратиться, исходя из входных значений. То есть, если на вход подается 000 или 101, то 1 будет для первого выхода и т.д.

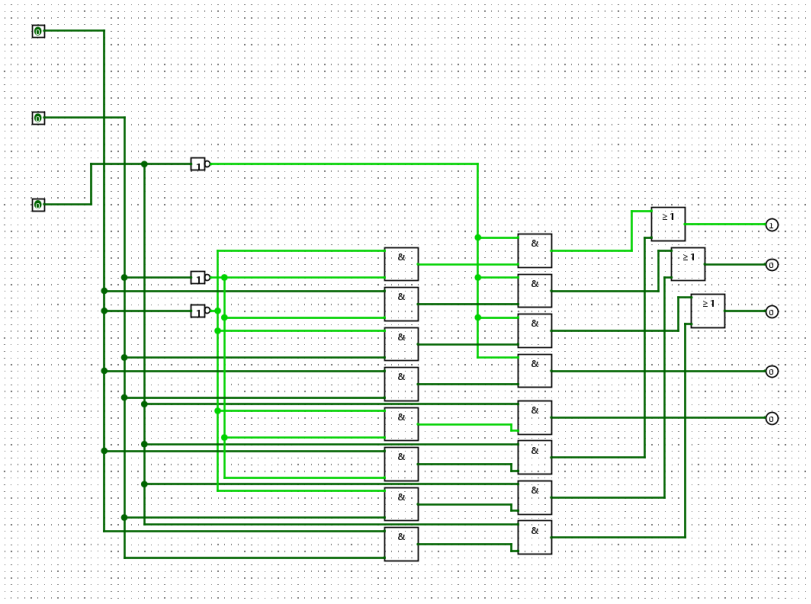

Рис. 12. Схема three_to_five

Эта схема уже имеет вид попроще, она по 5 битам возвращает трехбитное число.

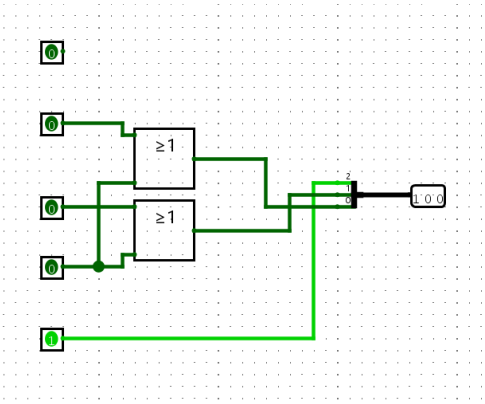

Рис. 14. Схема five_to_three

В этой схеме реализовано сложение в столбик. Summator (рис. 14) суммирует два однобитных числа, их сумму по модулю 2 записывает в соответствующую ячейку ответа, а остаток прибавляет к следующим битам.

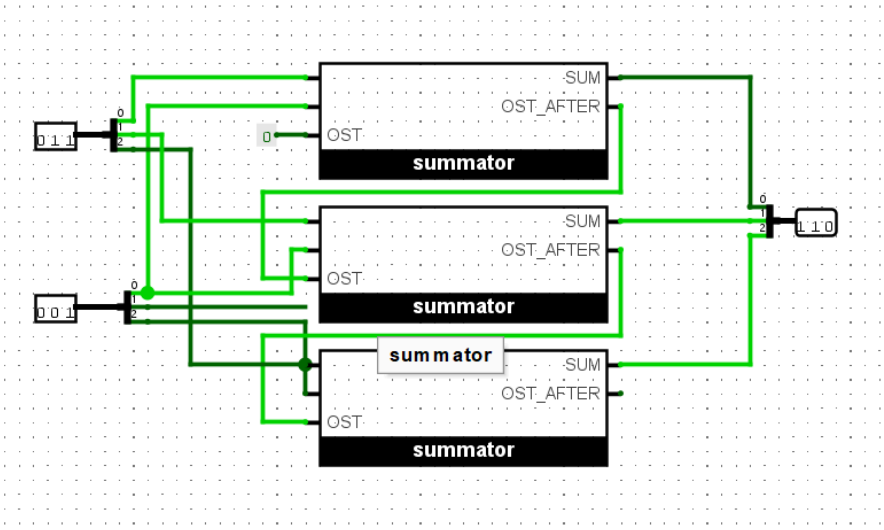

Рис. 13. Схема Addition

Эта схема как раз суммирует три однобитных числа (сами числа и остаток от прошлого вычисления).

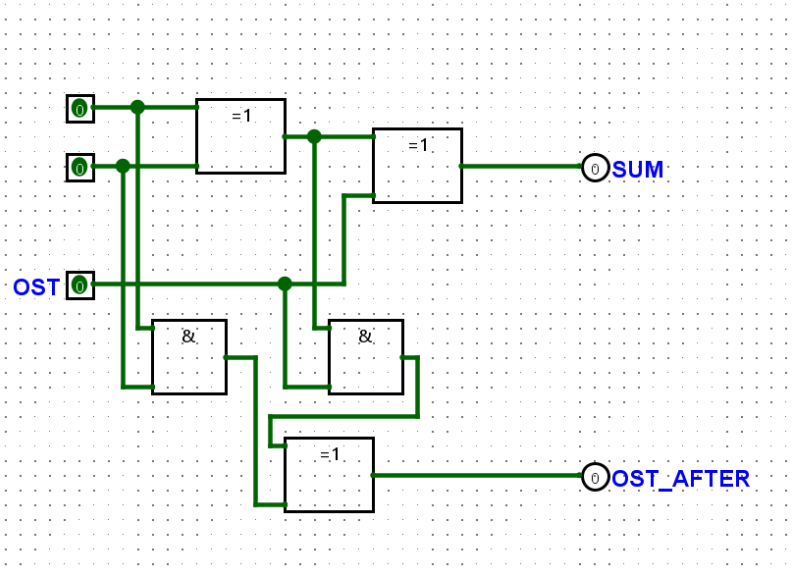

Рис. 15. Схема Summator

Эта схема для нахождения номера ячейки (head – get). Для этого сначала оба числа берутся по модулю 5, затем сравниваются (рис. 16), если head больше, то от него просто отнимается (рис. 17) get, если get больше, то считается следующее выражение (head + 5 – get).

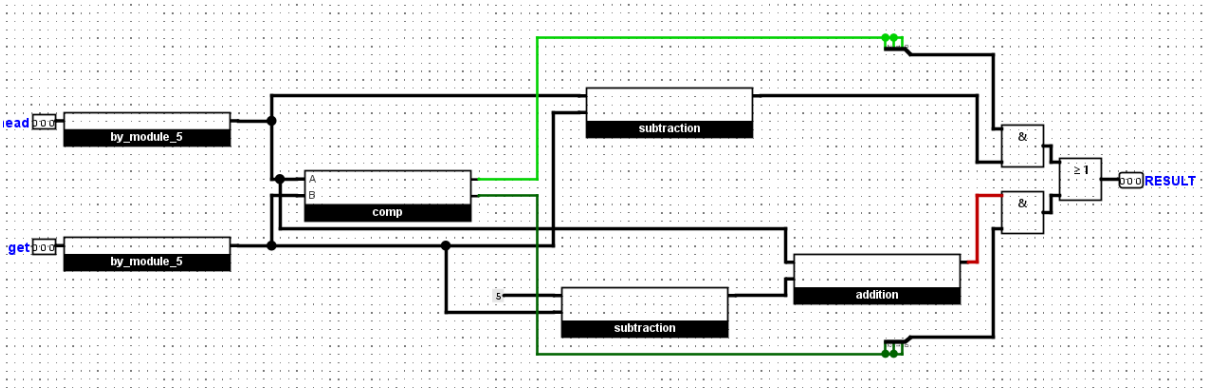

Рис. 6. Схема get_pos_for_get

Происходит последовательное сравнение битов, и если A >= B, то первое выходное значение будет 1, иначе второе будет 1.

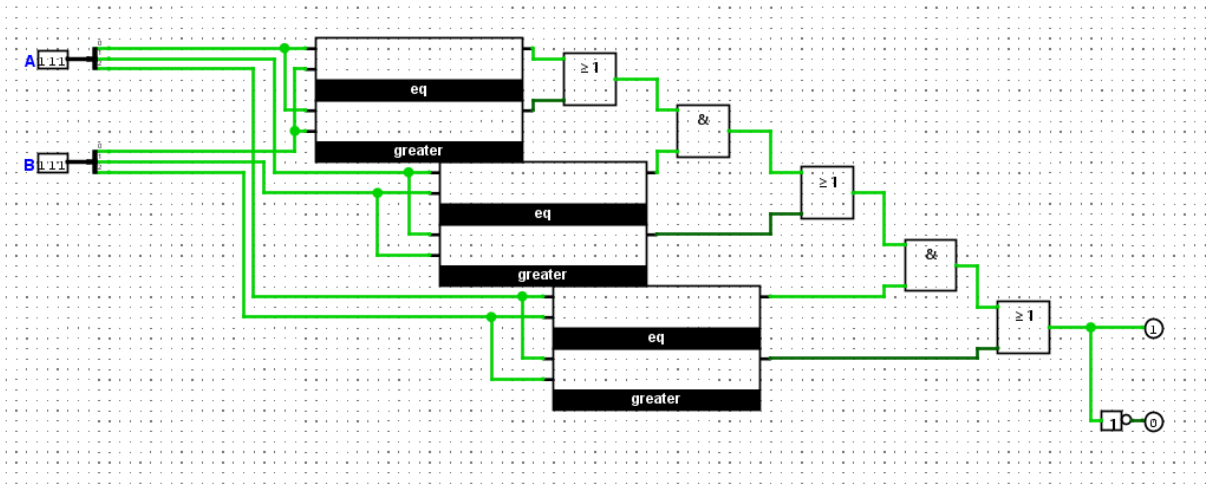

Рис. 16. Схема comp

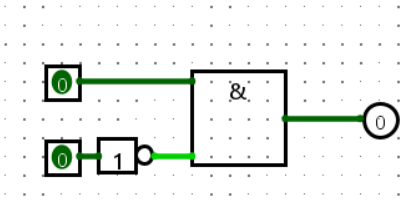

Рис. 18. Схема Greater

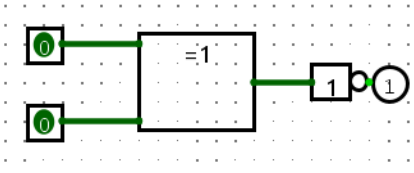

Рис.19. Схема EQ

Схема для вычитания трехбитных чисел. Не сильно хитрее суммирования.

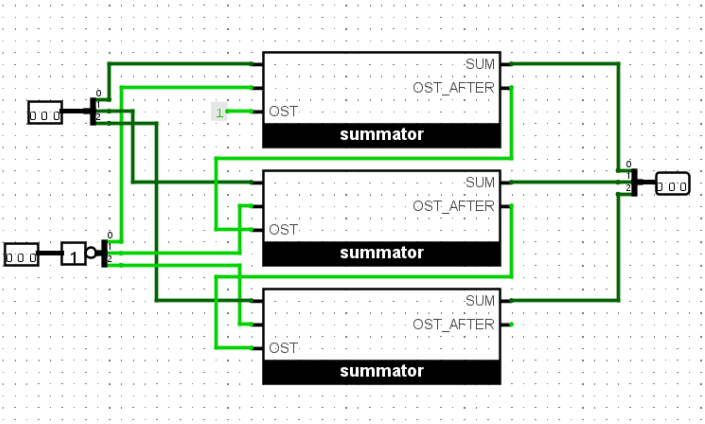

Рис. 17. Схема Subtraction

Вернемся к схеме Stack. Следующий шаг – demux. На вход подается адрес ячейки, к которой происходит запрос и бит, отвечающий за то, изменяем ли мы ячейку в этом такте. На выходе под номером ячейки, которая будет изменяться, будет стоять 1, а у всех остальных 0 (могут вообще ячейки не изменяться и все быть 0).

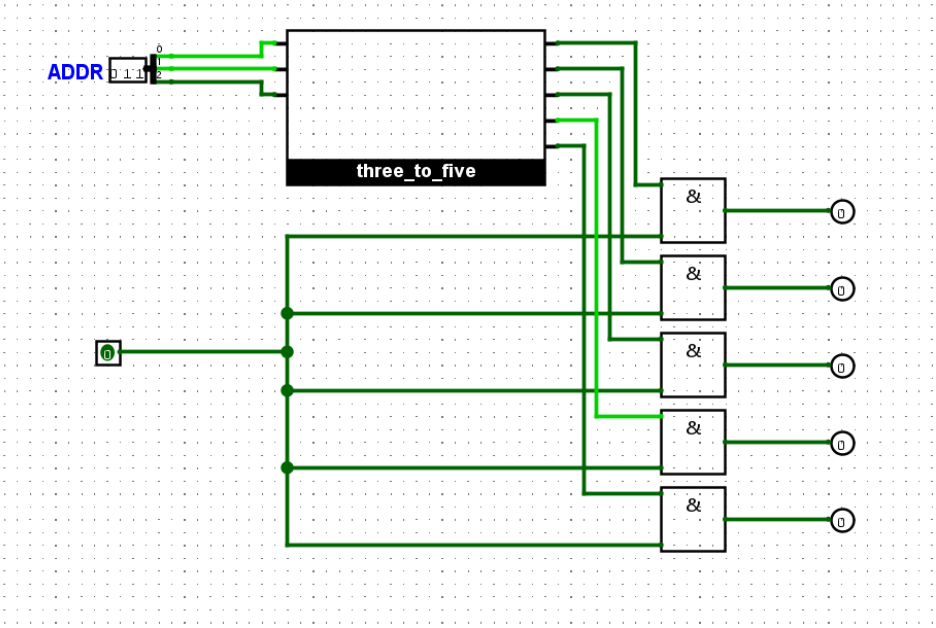

Рис. 20. Схема Demux

Затем в ячейки, в которых хранятся значения стека, подается значение I_DATA, RESET и соответствующий бит из схемы Demux как CLK. Четырехбитное число хранится с помощью четырех d_trigger_reset (рис. 22). На выход подается нынешнее значение ячейки.

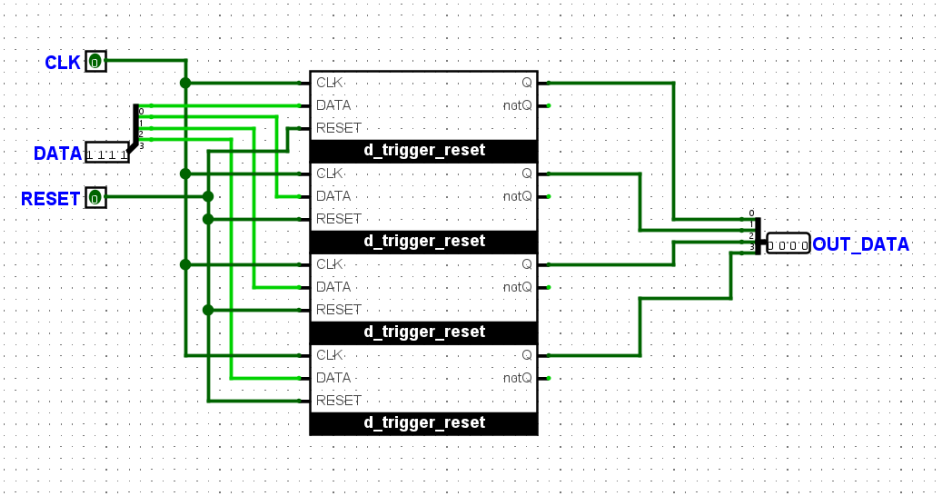

Рис. 21. Схема mem_four_bit

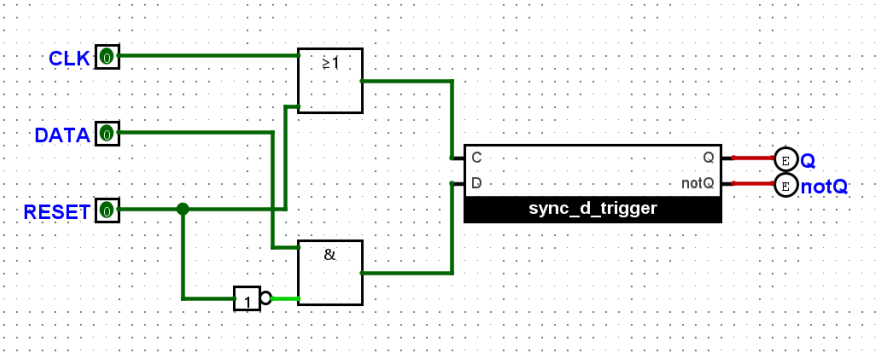

Рис. 22. Схема d_trigger_reset

Ну и в конце вызывается схема mux, которая по ADDR понимает, какую ячейку необходимо вернуть, и возвразат ее в качестве ответа.

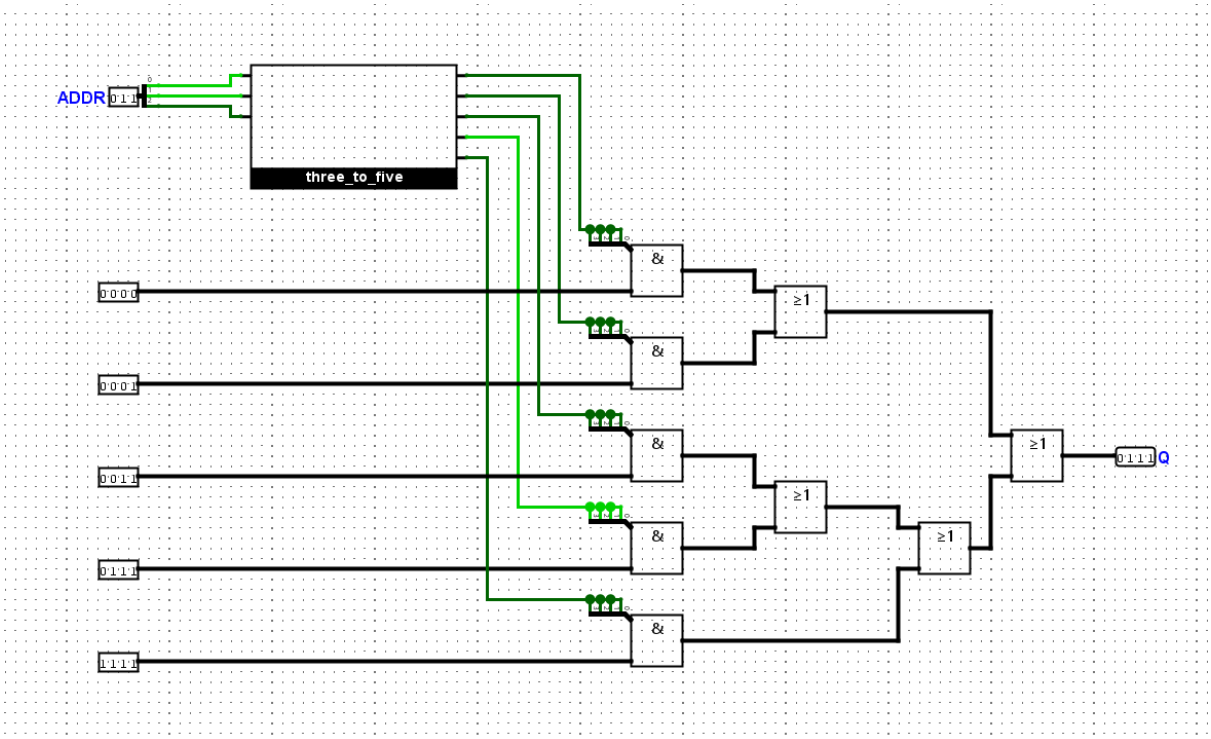

Рис. 23. Схема mux

Verilog

Structural

Все схемы, реализованные в Logisim, просто были перенесены в Verilog. Рассмотрим `module` stack_structural_easy.

```
module stack_structural_easy(
    output wire[3:0] O_DATA,
    input wire RESET,
    input wire CLK,
    input wire[1:0] COMMAND,
    input wire[2:0] INDEX,
    input wire[3:0] I_DATA
    );

    wire push, pop, get;
    commands f1(push, pop, get, COMMAND);

    wire [2:0] addr;
    number f2(addr, push, pop, get, CLK, RESET, INDEX);

    wire C;
    and (C, push, CLK);

    wire Q1, Q2, Q3, Q4, Q5;
    demux f3(Q1, Q2, Q3, Q4, Q5, addr, C);

    wire [3:0] D1, D2, D3, D4, D5;
    mem_four_bit f4(D1, Q1, I_DATA, RESET);
    mem_four_bit f5(D2, Q2, I_DATA, RESET);
    mem_four_bit f6(D3, Q3, I_DATA, RESET);
    mem_four_bit f7(D4, Q4, I_DATA, RESET);
    mem_four_bit f8(D5, Q5, I_DATA, RESET);

    mux f9(O_DATA, D1, D2, D3, D4, D5, addr);
endmodule
```

На вход подаются Reset, Clk, Command, Index, I_Data, а на выход – O_Data. Создаются провода push, pop, get, которые подключаются к выходу модуля commands. Создается провод addr, который будет выходом для модуля number, и будет содержать в себе адрес вершины, к которой мы сейчас будем обращаться. Дальше делаем все таким же образом. Выходом модуля mux будет O_Data.

Behaviour

```
module stack_behaviour_easy(
    output wire[3:0] O_DATA,
    input wire RESET,
    input wire CLK,
    input wire[1:0] COMMAND,
    input wire[2:0] INDEX,
    input wire[3:0] I_DATA
    );

    bit [4:0] [3:0] stack;
    reg [2:0] head;
    reg [3:0] output_;
    assign O_DATA = output_;

    initial begin
        head = 4;
    end

    always @(CLK) begin
        if (CLK == 1 && RESET == 0) begin
            if (COMMAND == 2'b01) begin
                head = (head + 1) % 5;
                stack[head] = I_DATA;
            end
            else if (COMMAND == 2'b10) begin
                output_ = stack[head];
                head = (head + 4) % 5;
            end
            else if (COMMAND == 2'b11) begin
                output_ = stack[(head - (INDEX) % 5 + 5) % 5];
            end
        end
    end

    always @(RESET) begin
        if (RESET == 1) begin
            head = 4;

            for(integer i = 0; i < 5; i += 1) begin
                stack[i] = 0;
            end

            output_ = stack[head];
        end
    end
endmodule
```

Если включена синхронизация и выключен reset, проверяем, какая операция сейчас вызывается. Если вызывается push, то сначала сдвигаем голову стека, потому что голова указывает на крайний вставленный элемент, а затем добавляем в соответствующую ячейку новый элемент. Если вызывается pop, то сначала присвоим ответу значение ячейки, а потом сдвинем голову. Если вызывается get, то ничего сдвигать не нужно, просто возвращаем значение нужной ячейки. При этом везде не забываем про взятие по модулю и возможность стать отрицательным.

Если включен reset, то зануляем все значения stack, вершиной стека станет последняя ячейка, ее же и отправим на вывод.

Результаты

При написании stack_structural я добавляла тесты ко всем функциям, чтобы не закопаться в конце в ошибках. Не закомментированы только тесты для самого стека. Продемонстрирую здесь их.

```
--- reset ---
                  0    CLK: x, COMMAND: x, INDEX: xxx, I_DATA: xxxx, RESET: x, O_DATA: xxxx
                  1    CLK: 0, COMMAND: 0, INDEX: 000, I_DATA: 0000, RESET: 1, O_DATA: 0000

--- push 1 2 3 ---
                  2    CLK: 0, COMMAND: 1, INDEX: 000, I_DATA: 0001, RESET: 0, O_DATA: 0000
                  3    CLK: 1, COMMAND: 1, INDEX: 000, I_DATA: 0001, RESET: 0, O_DATA: 0001
                  4    CLK: 0, COMMAND: 1, INDEX: 000, I_DATA: 0010, RESET: 0, O_DATA: 0000
                  5    CLK: 1, COMMAND: 1, INDEX: 000, I_DATA: 0010, RESET: 0, O_DATA: 0010
                  6    CLK: 0, COMMAND: 1, INDEX: 000, I_DATA: 0011, RESET: 0, O_DATA: 0000
                  7    CLK: 1, COMMAND: 1, INDEX: 000, I_DATA: 0011, RESET: 0, O_DATA: 0011

--- get 0 1 2 3 4 5 6 -- answers have to be 3 2 1 0 0 3 2 ---
                  8    CLK: 0, COMMAND: 3, INDEX: 000, I_DATA: 0011, RESET: 0, O_DATA: 0011
                  9    CLK: 1, COMMAND: 3, INDEX: 000, I_DATA: 0011, RESET: 0, O_DATA: 0011
                 10    CLK: 0, COMMAND: 3, INDEX: 001, I_DATA: 0011, RESET: 0, O_DATA: 0010
                 11    CLK: 1, COMMAND: 3, INDEX: 001, I_DATA: 0011, RESET: 0, O_DATA: 0010
                 12    CLK: 0, COMMAND: 3, INDEX: 010, I_DATA: 0011, RESET: 0, O_DATA: 0001
                 13    CLK: 1, COMMAND: 3, INDEX: 010, I_DATA: 0011, RESET: 0, O_DATA: 0001
                 14    CLK: 0, COMMAND: 3, INDEX: 011, I_DATA: 0011, RESET: 0, O_DATA: 0000
                 15    CLK: 1, COMMAND: 3, INDEX: 011, I_DATA: 0011, RESET: 0, O_DATA: 0000
                 16    CLK: 0, COMMAND: 3, INDEX: 100, I_DATA: 0011, RESET: 0, O_DATA: 0000
                 17    CLK: 1, COMMAND: 3, INDEX: 100, I_DATA: 0011, RESET: 0, O_DATA: 0000
                 18    CLK: 0, COMMAND: 3, INDEX: 101, I_DATA: 0011, RESET: 0, O_DATA: 0011
                 19    CLK: 1, COMMAND: 3, INDEX: 101, I_DATA: 0011, RESET: 0, O_DATA: 0011
                 20    CLK: 0, COMMAND: 3, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0010
                 21    CLK: 1, COMMAND: 3, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0010

--- pop * 7 -- answers have to be 3 2 1 0 0 3 2 ---
                 22    CLK: 0, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0011
                 23    CLK: 1, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0011
                 24    CLK: 0, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0010
                 25    CLK: 1, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0010
                 26    CLK: 0, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0001
                 27    CLK: 1, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0001
                 28    CLK: 0, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0000
                 29    CLK: 1, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0000
                 30    CLK: 0, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0000
                 31    CLK: 1, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0000
                 32    CLK: 0, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0011
                 33    CLK: 1, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0011

                  34    CLK: 0, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0010
                  35    CLK: 1, COMMAND: 2, INDEX: 110, I_DATA: 0011, RESET: 0, O_DATA: 0010
```

Сначала делаем reset, зануляя все элементы.

Затем делаем последовательное добавление чисел 1, 2 и 3.

Теперь проверяем работу get для индексов 0, 1, 2, 3, 4, 5, 6. Ответы должны быть: 3 2 1 0 0 3 2.

Теперь проверяем pop. Делаем его 7 раз. Вывод должен быть таким же, как для get. Все работает)