Gebze Technical University
Computer Engineering

CSE 331 - 2018 Fall

ASSIGNMENT 2 REPORT

HALİL ONUR ÇEÇEN

161044057

Making of ALU

To achieve an ALU which can select an operation in a list of 8 I used 8x1 mux to select which operation to return. Before that I calculated all outcomes and directed them to 8x1 mux.

ALU Modules (Explained)

Mux

To assemble a 8x1 mux I needed 2 4x1 mux and 1 2x1 mux. And to construct a 4x1 mux I needed 3 more 2x1 mux. So, I started this assignment by building a 2x1 mux in Verilog. After first successful 2x1 mux I needed 32 bit input and output version of this mux to create 32 bit 4x1 and 8x1 muxes. In the way I used 32 times 2x1 muxes. Besides, in the meantime I started to learn Verilog Hdl syntax.

```
module mux_2x1(out, in1, in2, select);
input in1, in2;
input select;
output out;
wire a,b, notSelect;

and(a, in2, select);
not(notSelect, select);
and(b, notSelect, in1);
or(out, a, b);
endmodule
```

Simple Gates

After successful 1 bit and 32-bit 2x1 mux, I wrote 4x1 mux in bot 1 bit and 32 bits. After them finally wrote the 32-bit 8x1 mux to create 32-bit ALU.

After mux I wrote 32 bit and operation using and gate 32 times. Also, I wrote the or, xor and nor gates using the same way.

```
module and_32(out, in1, in2);
input [31:0] in1, in2;
output [31:0] out;

and and1(out[31], in1[31], in2[31]),
    and2(out[30], in1[30], in2[30]),
    and3(out[29], in1[29], in2[29]),
    and4(out[28], in1[28], in2[28]),
```

Adder and Subtractor

To create adder and subtractor I used classic Full adder design which relies on an Half adder without carry in input. Also, I need to make these 32-bit input and output. To achieve that I used similar method of 32-bit gate creation I used before.

```
module full_adder_32 (out, carry_out, in1, in2, carry_in);
input [31:0] in1, in2;
input carry_in;
output carry_out;
output [31:0] out;
wire [30:0] carry;

full_adder FA1(out[0], carry[0], in1[0], in2[0], carry_in),
   FA2(out[1], carry[1], in1[1], in2[1], carry[0]),
   FA3(out[2], carry[2], in1[2], in2[2], carry[1]),
   FA4(out[3], carry[3], in1[3], in2[3], carry[2]),
   FA5(out[4], carry[4], in1[4], in2[4], carry[3]),
```

Since our ALU output is single 32-bit output, I ignored carry out hence overflow situation.

To subtract two 32-bit input I used full adder again but with negative of the second number. To get negative of it I used 2's complement method. After inverting every bit I added 1 bit by carry in of full adder.

Shifters

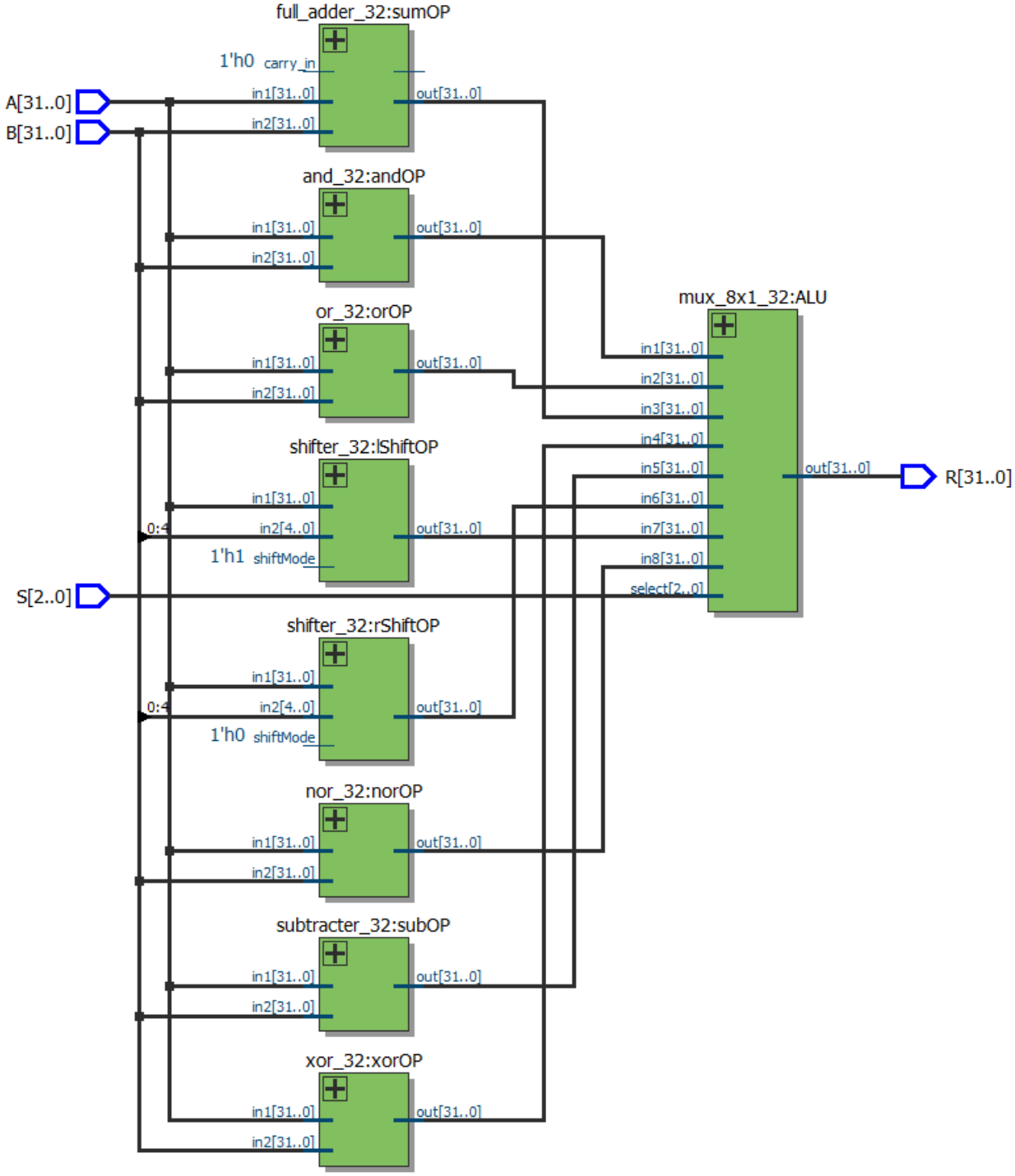
To write right and left shifters I used mux design which our lecturer shared with us in class. But to work both ways I used 4x1 mux instead of 2x1. I used most significant bit of select to determine if the shifter is going to work or just pass current values and other two inputs will take left and right bits respectively (left most significant bit of value or 0 respectively).

To determine what is most significant bit of the result, I used shifting direction and value of inputs most significant bit.

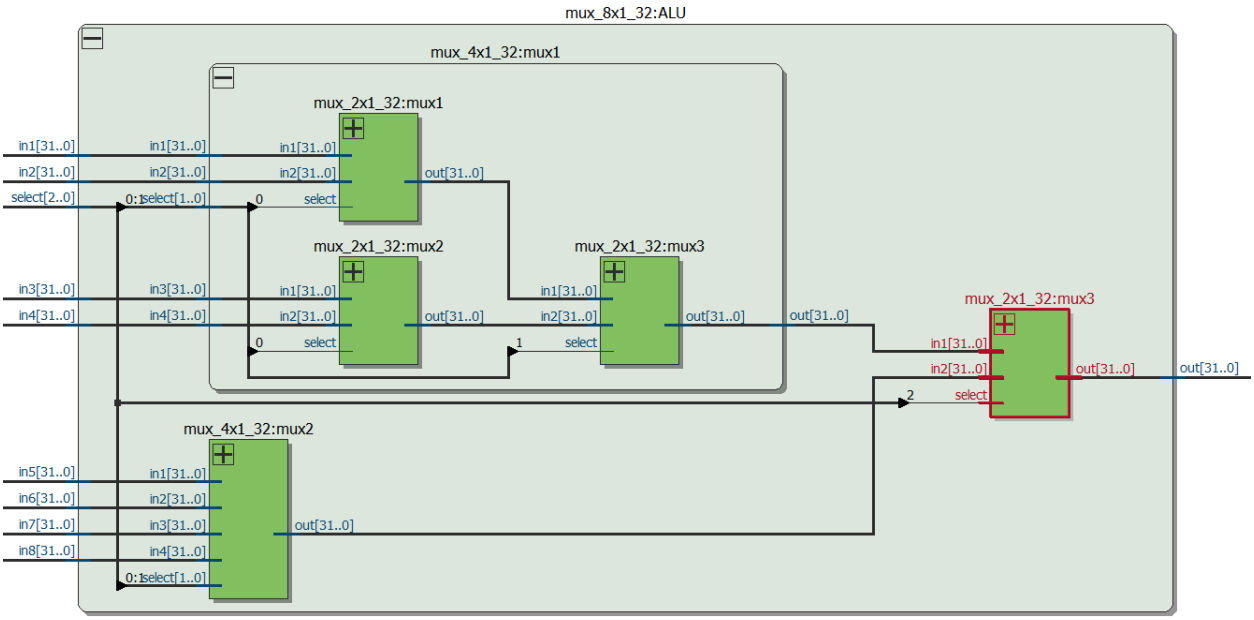
Also, my module gets another input for determination of shift direction as shiftMode.

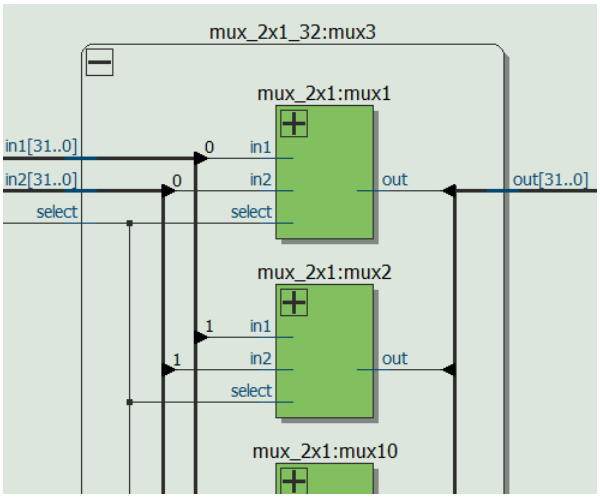
Schematic Design

ALU



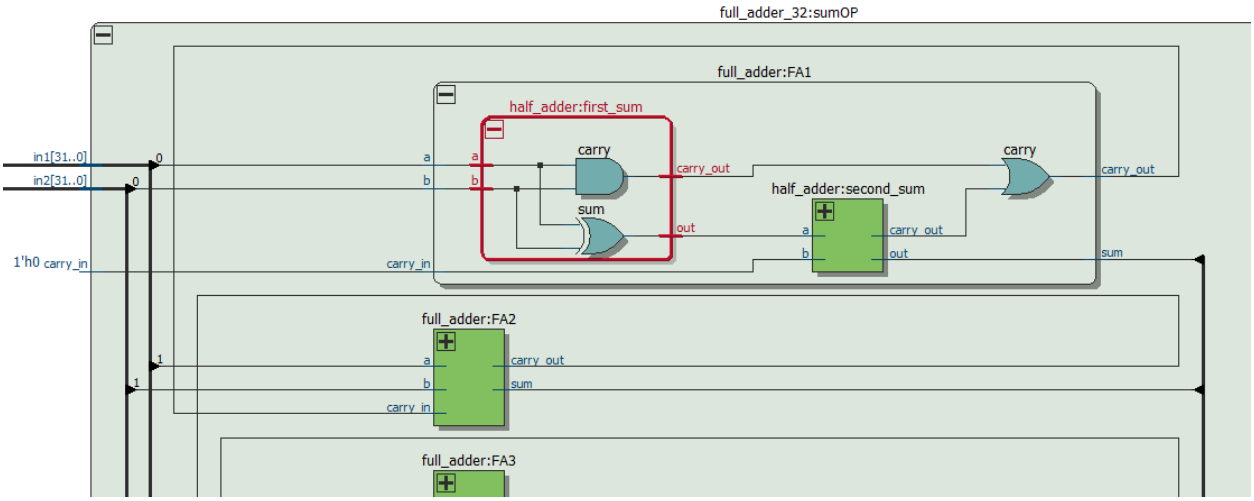
Mux



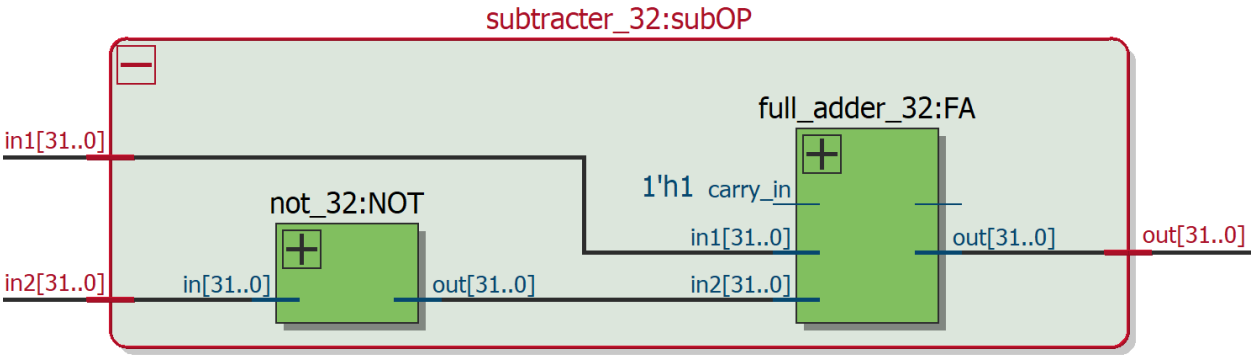


2x1 mux goes on like that to make 32 bit selection.

Adder – Subtractor

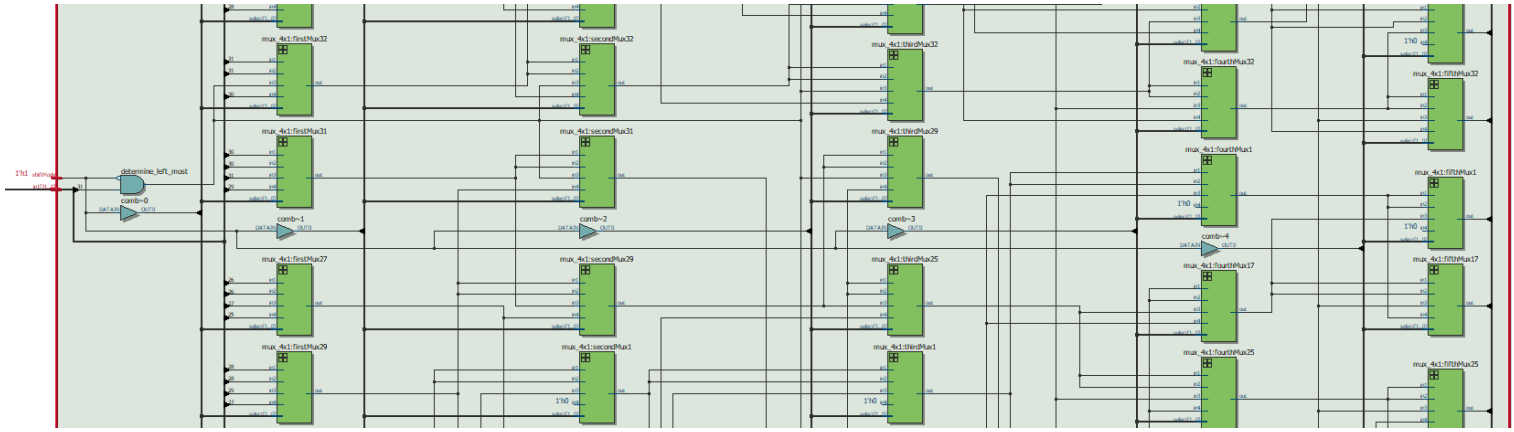


This module has 32 single bit Full Adders for summation operation. Also, a full adder and half adder schematics can be seen in this schematic.

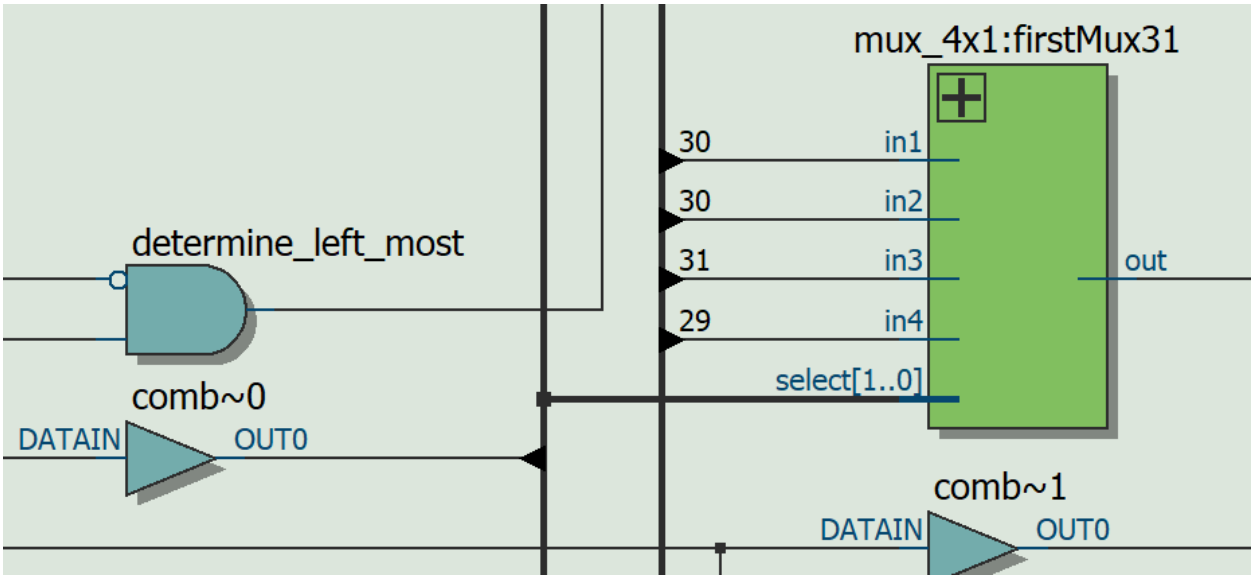


Subtractor uses an 32-bit full adder with inverted bits and 1 carry.

Shifter



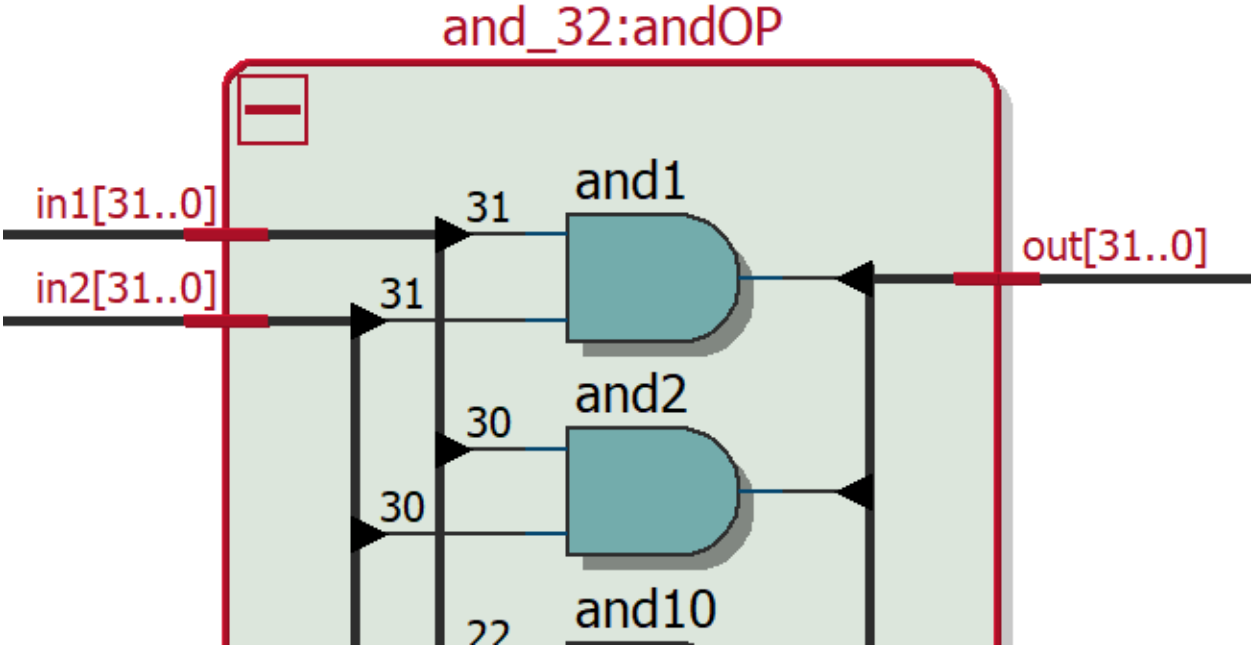
Shifter unit uses 32 of 4x1 1-bit muxes in 5 rows.



Determines the most significant bit for right shift using shift direction input and input 1.

4x1 muxes take 2 of same input and one for each direction. If MSB of select is zero no shifting is done in that row.

Simple 32-Bit Gate Modules



All simple gates (and, nor, or, xor) goes like that 32 times for 32-bit input. 32-Bit not module used in shifter takes one input.

ModelSim Results

All modules

alu32	Module
alu32_testbench	Module
and_32	Module
full_adder	Module
full_adder_32	Module
half_adder	Module
mux_2x1	Module
mux_2x1_32	Module
mux_4x1	Module
mux_4x1_32	Module
mux_8x1_32	Module
nor_32	Module
not_32	Module
or_32	Module
shifter_32	Module
subtracter_32	Module
xor_32	Module

Simulation Results

```
# time =  0, A = 01111011111111111111111111111110, B=11111111111111111000011111111111, R=01111011111111111100001111111110, S=000
# time = 10, A = 01111011111111111111100011111110, B=11111111100011111100001111111111, R=01111011100011111100000011111110, S=000
# time = 20, A = 01111011111110000011111111111110, B=00100011100011111100001111111111, R=00100011100010000000001111111110, S=000
# time = 30, A = 01111011111111000001111111111110, B=11111111100011111100001111100011, R=11111111111111111111111111111111, S=001
# time = 40, A = 01111011111111000000000000000110, B=11111111100011111100001111111111, R=11111111111111111100001111111111, S=001
# time = 50, A = 01111011111111000001111111111110, B=11111111100011111100001111111111, R=11111111111111111111111111111111, S=001
# time = 60, A = 00000000000000000000000000000010, B=00000000000000000000000000000100, R=00000000000000000000000000000110, S=010
# time = 70, A = 00000000000000000000000000001100, B=00000000000000000000000000011001, R=00000000000000000000000000100101, S=010
# time = 80, A = 00000000100001001010001011110100, B=00000000000001001011000101001100, R=00000000100010010101010001000000, S=010
# time = 90, A = 01010101010101010101010101010101, B=10101010101010101010101010101010, R=11111111111111111111111111111111, S=011
# time = 100, A = 00011011100111111111000011100101 0, B=11001101100000001111110000000111, R=11010110000111110001110111001101, S=011
# time = 110, A = 01101110111111111111000011111111, B=01111111111111111111111111111111, R=00010001000000000000111100000000, S=011
# time = 120, A = 00000000000000000000000000000110, B=00000000000000000000000000000100, R=00000000000000000000000000000010, S=100
# time = 130, A = 00000000000000000000000000011001, B=00000000000000000000000000001100, R=00000000000000000000000000001101, S=100
# time = 140, A = 00000000100001001010001011110100, B=00000000000001010100100001110100, R=00000000011110100001001000000000, S=100
# time = 150, A = 00000000000000000000000000001000, B=00000000000000000000000000000001, R=00000000000000000000000000000100, S=101
# time = 160, A = 11111111111111111110000011111100, B=00000000000000000000000000011111, R=11111111111111111111111111111111, S=101
# time = 170, A = 11111111100000000011110000000000, B=00000000000000000000000000000011, R=11111111111100000000011110000000, S=101
# time = 180, A = 00000000000000000000000000001000, B=00000000000000000000000000000001, R=00000000000000000000000000010000, S=110
# time = 190, A = 11111111111111111110000011111101, B=00000000000000000000000000011110, R=01000000000000000000000000000000, S=110
# time = 200, A = 10000000000000000000000000000000, B=00000000000000000000000000000011, R=00000000000000000000000000000000, S=110
# time = 210, A = 11111100000000000000000000001000, B=00000000111111111100000000000001, R=00000011000000000011111111110110, S=111
# time = 220, A = 11111111111111111111100100111101, B=00000000000000011111100000011110, R=00000000000000000000000011000000, S=111
# time = 230, A = 10000000000000000000000001000000, B=00000000010000000000000000000011, R=01111111101111111111111110111100, S=111
```