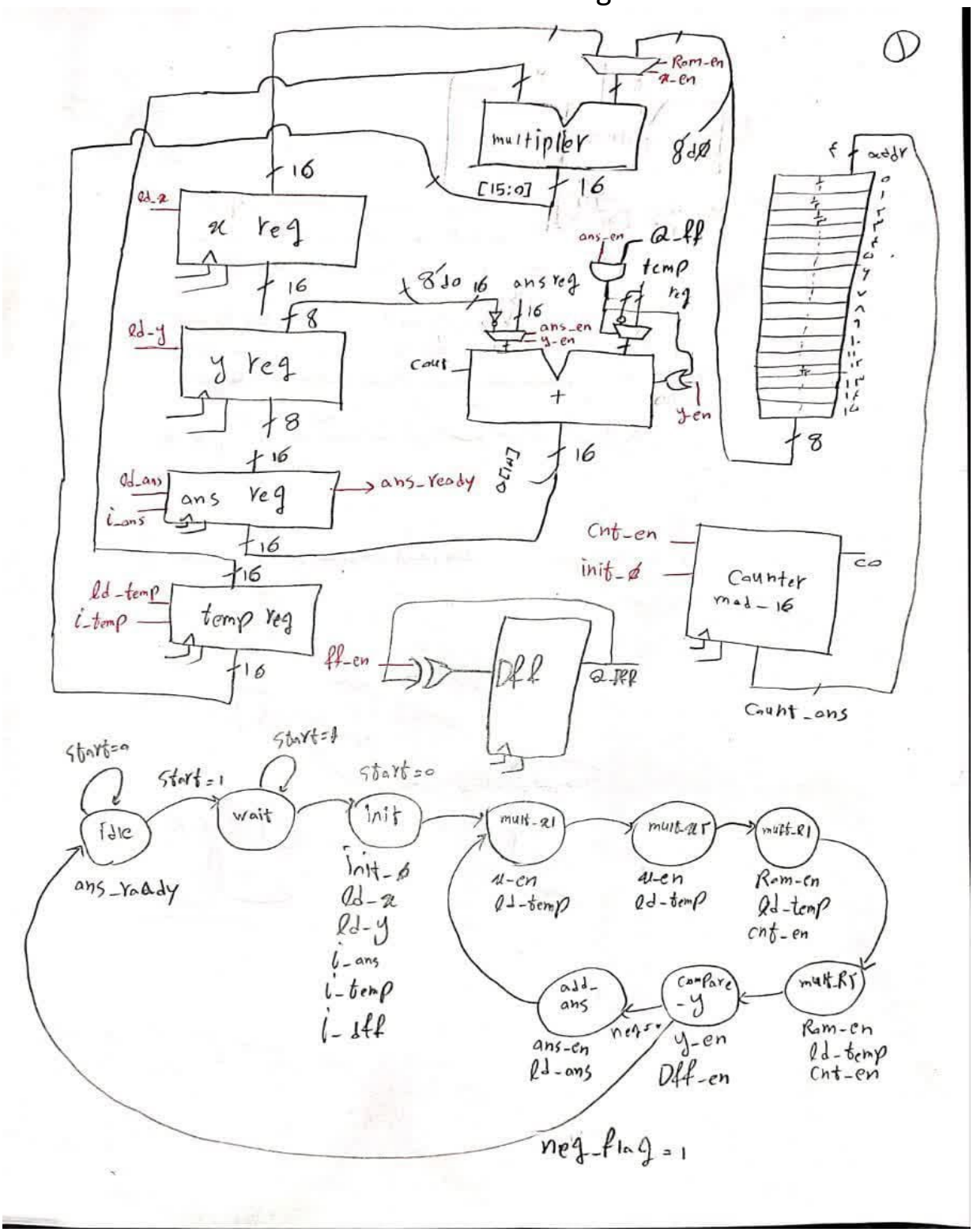
All parts are combined together and will be explained within the flow of design:

At first let's start with the hand written design:



Most of the data path is clear except for 3 parts:

First multiplier always multiplies temp with x_reg or rom value, and the choose signal is within controller's duties

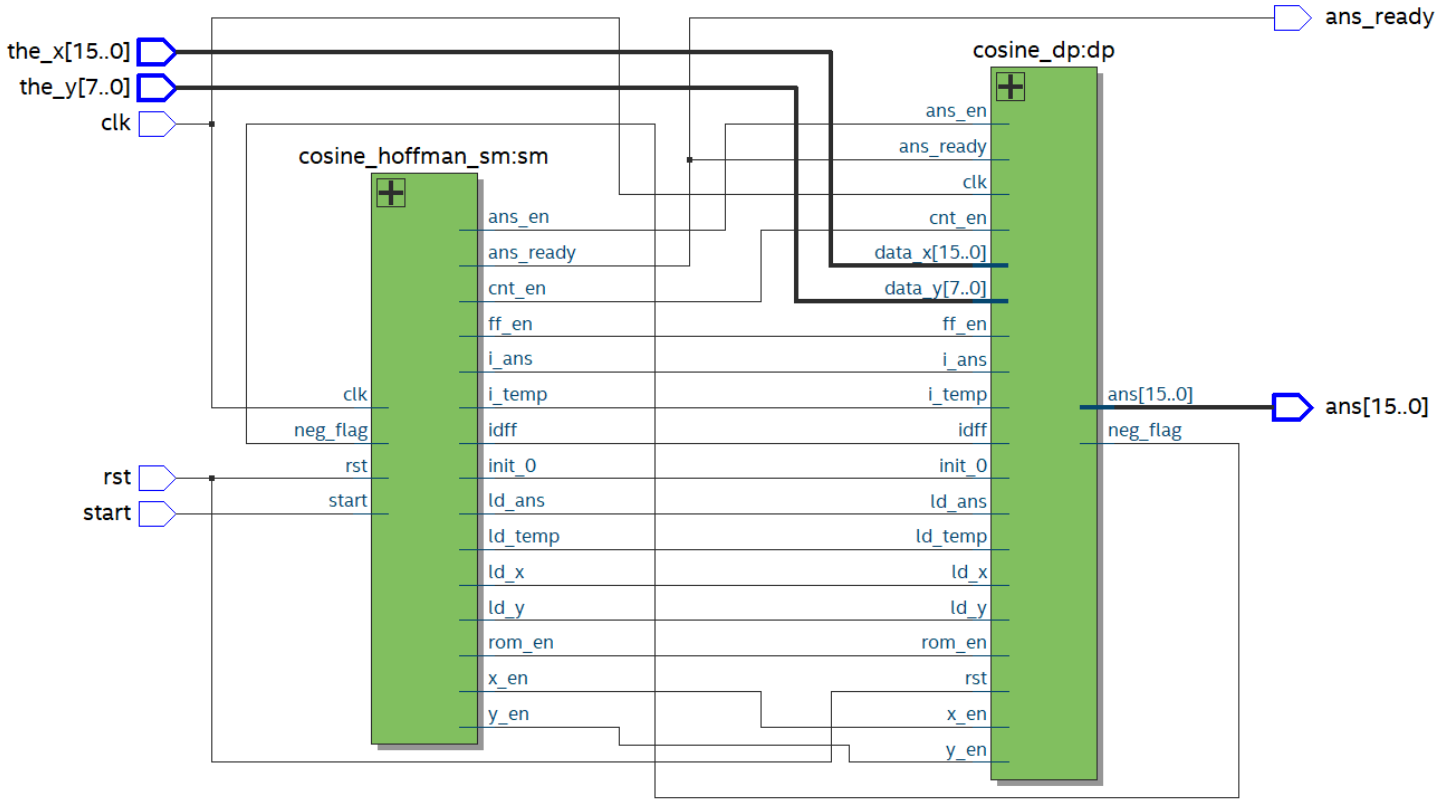
Second in add unit, sometimes it adds mines y with abs of term value to check if term is lesser than y or not, and the other times it adds answer value with mines or abs of term value, and the adder uses a dff and some other helping signals to choose between abs or mines, and the dff changes every adding cycle using controller's signals.

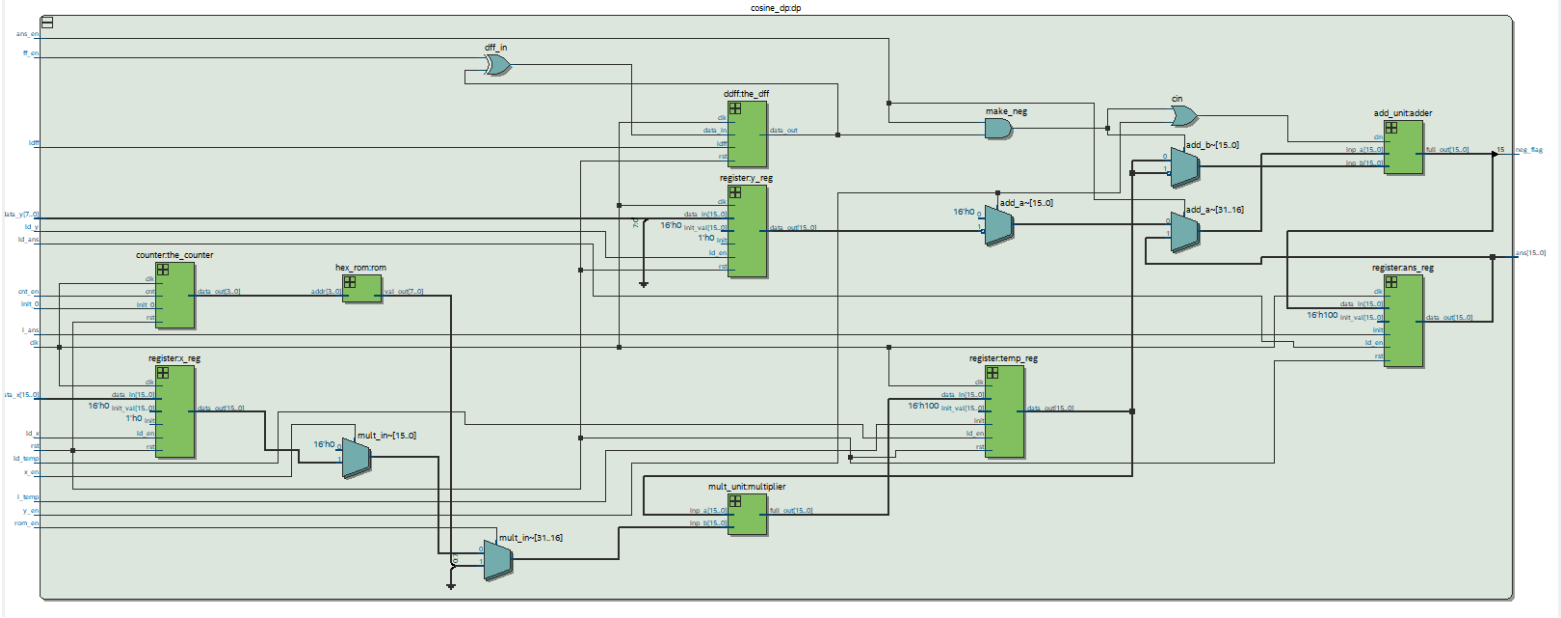
At last, our rom contains only 12 values, because ((pi)^12)/fact(12) is lesser than 8'b00000001 and is the biggest term with power of 12.

And now about the controller, most of the parts are clear but we must note that it could be implemented using x^2 or n * (n + 1) values in less clock cycles, but do to less importance of clock cycle counts than precision it has been chosen to be like this. Also the dff_en has the duty of adding negative or positive value of our term to ans, controlling the dff value.

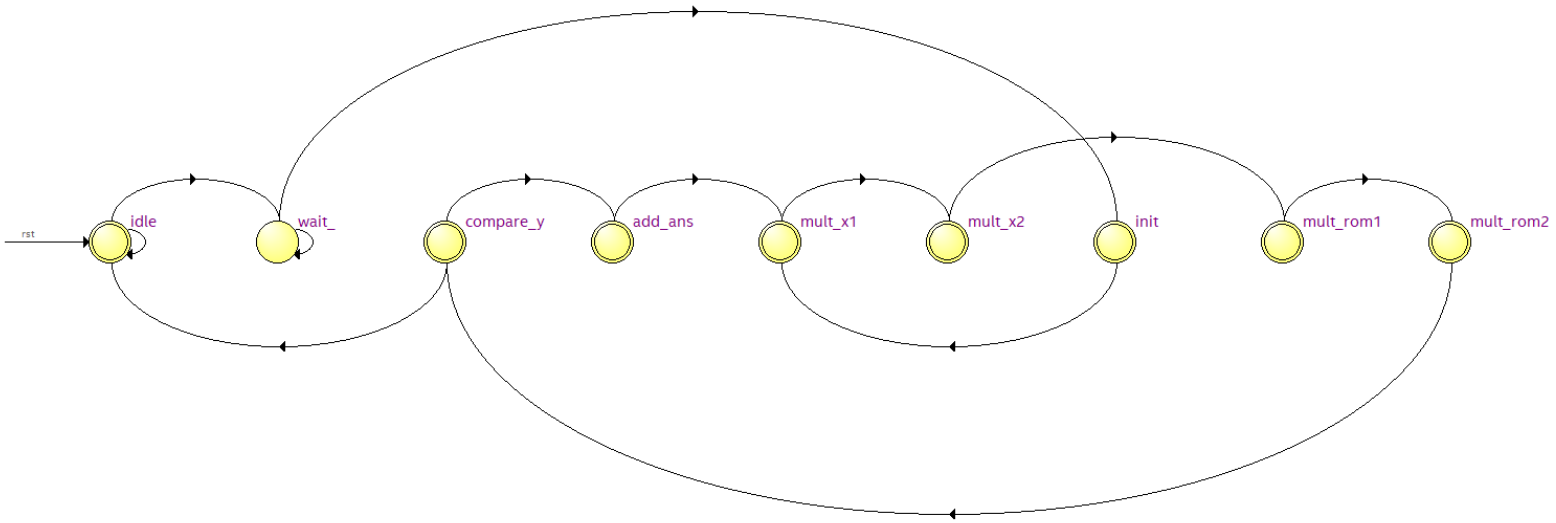
Please note that the red wires in data path are coming from controller.

Now that we have a better sense of this project, let's move to the synthesis parts to see how much our Verilog codes correspond to the hand written diagram: (Verilog codes can be found in "a" folder)


the_x[15..0]
the_y[7..0]
clk
rst
start
ans_ready
ans[15..0]
cosine_hoffman_sm:sm
ans_en
ans_ready
cnt_en
ff_en
i_ans
i_temp
idff
init_0
ld_ans
ld_temp
ld_x
ld_y
rom_en
x_en
y_en
clk
neg_flag
rst
start
cosine_dp:dp
ans_en
ans_ready
clk
cnt_en
data_x[15..0]
data_y[7..0]
ff_en
i_ans
i_temp
idff
init_0
ld_ans
ld_temp
ld_x
ld_y
rom_en
rst
x_en
y_en
ans[15..0]
neg_flag

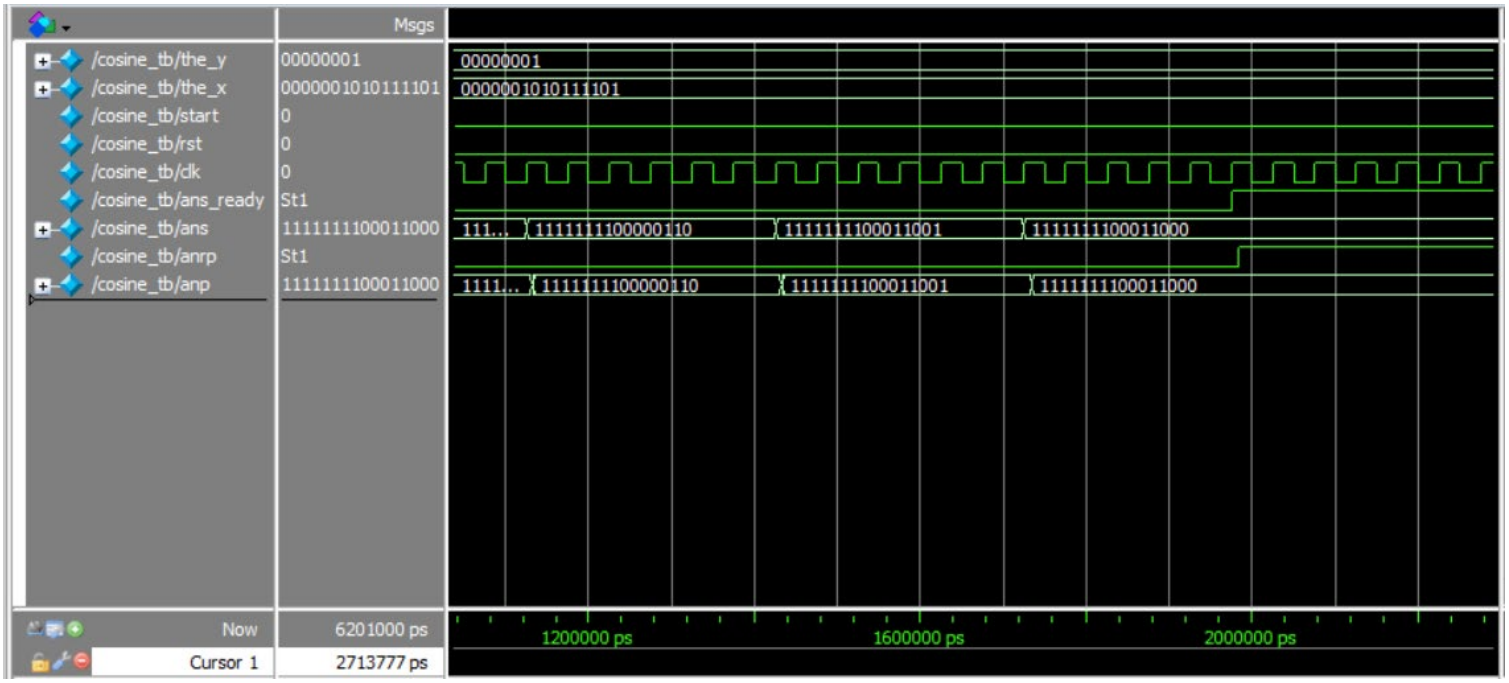

cosine_dp:dp
dff_in
ddff:the_dff
make_neg
cin
add_unitadder
add_b~[15..0]
add_a~[15..0]
add_a~[31..16]
registery_reg
counter:the_counter
hex_rom:rom
registerans_reg
registerx_reg
registertemp_reg
mult_in~[15..0]
mult_in~[31..16]
mult_unit:multiplier

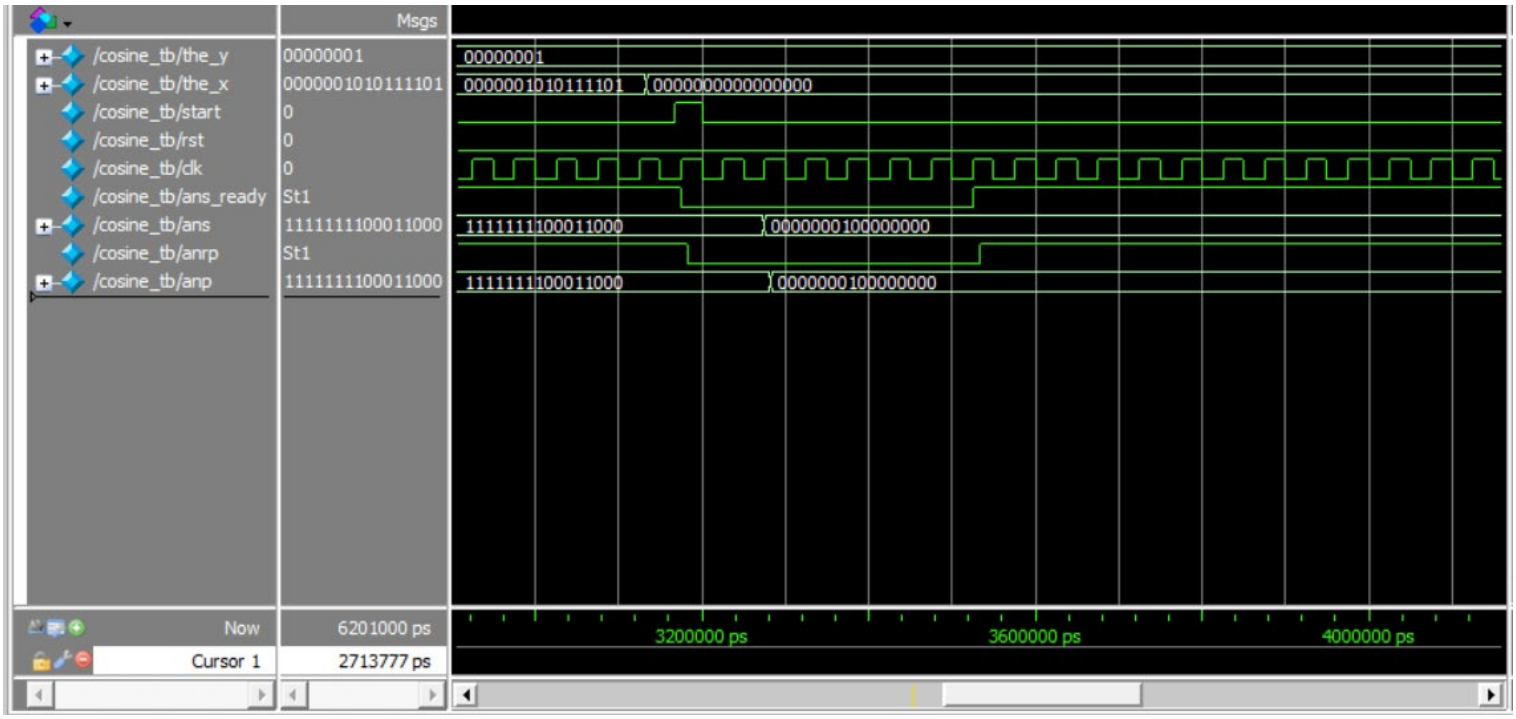
\



As you can see the designs are completely corresponding the hand written designs.

Now this is the best time to show you the test benches:





The first photo is cos(7pi/8), which it's true cosines value is -0.92, and our circuit calculated -0.91 which is a decent and close value. The second photo is also a means of reusability of our circuit among with preciseness, which will wait a while after the first input, and then when the start signal turns to 1, it starts to calculate the new input again, which as you can see gave us the answer of 1 which is completely true. Not that y is 0.00000001 in this test bench. As a means of controller and data path working fine and seprately, we printed some states, flags and values. Below you can see the ending prints of first test case and starting and ending print of the second one. As you can see the states are changing and when they recive a signal from data path, they stop the operation:

```
# this is ans_out in dp: 1111111100011001
# this is the current state:  3
# this is the neg_flag: 0
#
# this is ans_out in dp: 1111111100011000
# this is the current state:  4
# this is the neg_flag: 0
#
# this is ans_out in dp: 1111111100011000
# this is the current state:  5
# this is the neg_flag: 0
#
# this is ans_out in dp: 1111111100011000
# this is the current state:  6
# this is the neg_flag: 0
#
# this is ans_out in dp: 1111111100011000
# this is the current state:  7
# this is the neg_flag: 1
#
# this is ans_out in dp: 1111111100011000
# this is the current state:  0
# this is the neg_flag: 0
#
# this is ans_out in dp: 1111111100011000
# this is the current state:  0
# this is the neg_flag: 0
#
```

```
# this is ans_out in dp: 1111111100011000
# this is the current state:  2
# this is the neg_flag: 0
#
# this is ans_out in dp: 1111111100011000
# this is the current state:  3
# this is the neg_flag: 0
#
# this is ans_out in dp: 0000000100000000
# this is the current state:  4
# this is the neg_flag: 0
#
# this is ans_out in dp: 0000000100000000
# this is the current state:  5
# this is the neg_flag: 0
#
# this is ans_out in dp: 0000000100000000
# this is the current state:  6
# this is the neg_flag: 0
#
# this is ans_out in dp: 0000000100000000
# this is the current state:  7
# this is the neg_flag: 1
#
# this is ans_out in dp: 0000000100000000
# this is the current state:  0
# this is the neg_flag: 0
```