Overview

Goal: build a 4-bit CPU using logic gates, meaning MUX, adder, ALU, registers are all implemented using logic gates

Registers: 4-bits wide

Gate delays

and — 3ns or — 3ns nand — 2ns nor — 2ns not — 1ns

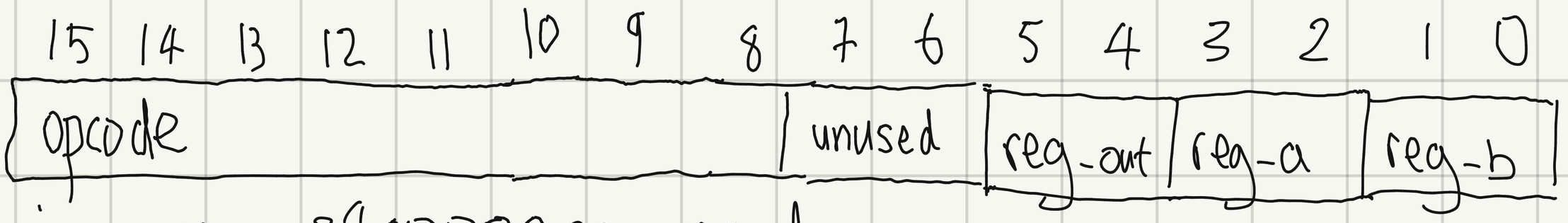
xor — 4ns xnor — 5 ns

Note: no need for a dependency tree, easier to just do a linear search through all ~23 modules to fix dependencies.

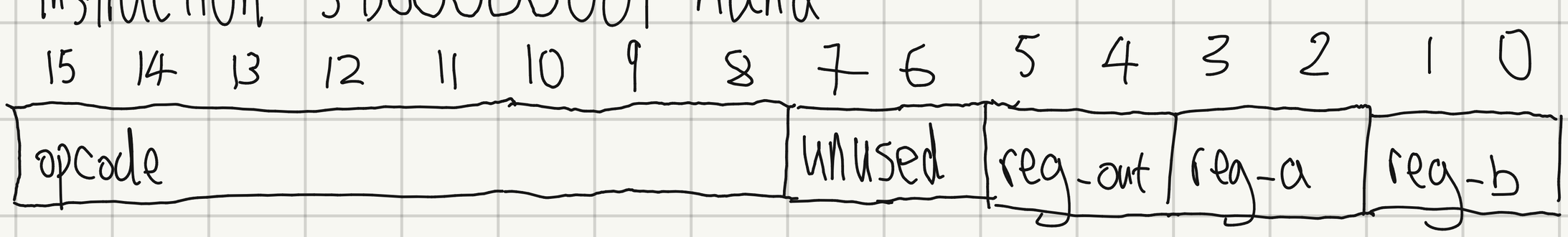
Instructions list

Instructions: memory for 16 instructions, each 11b wide

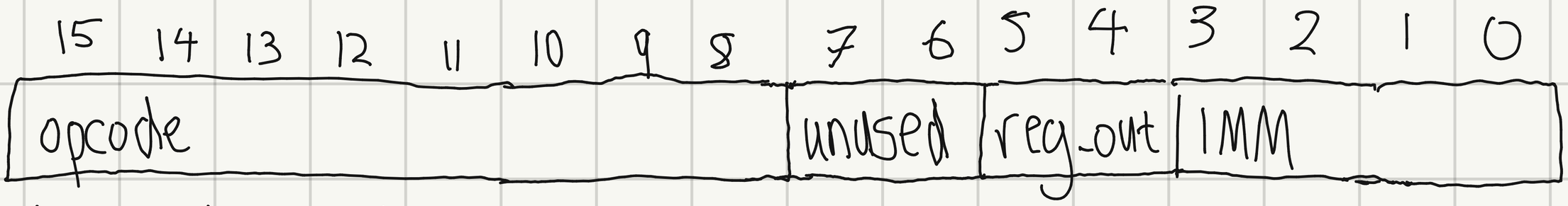
instruction 3'b00000000 add



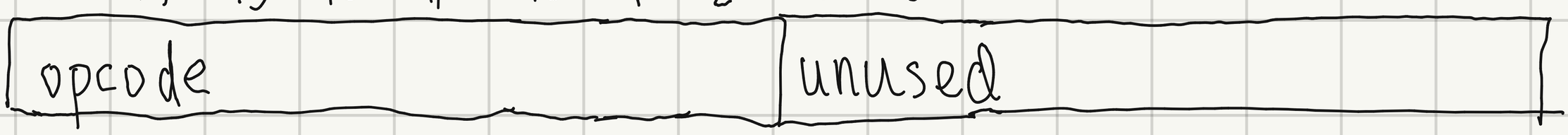
instruction 3'b00000001 nand



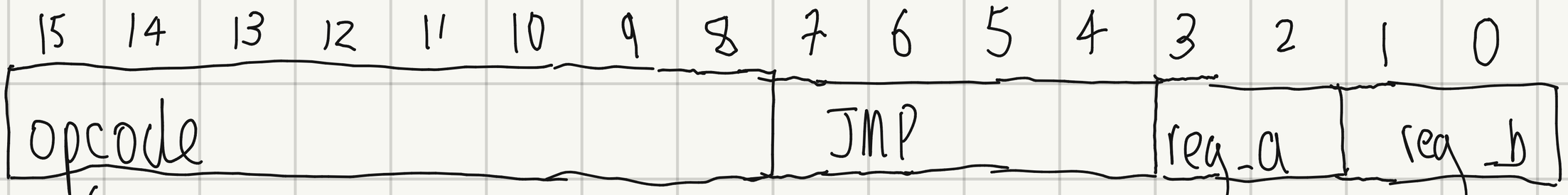
instruction 3'b00000010 move



instruction 3'b00000011 noop

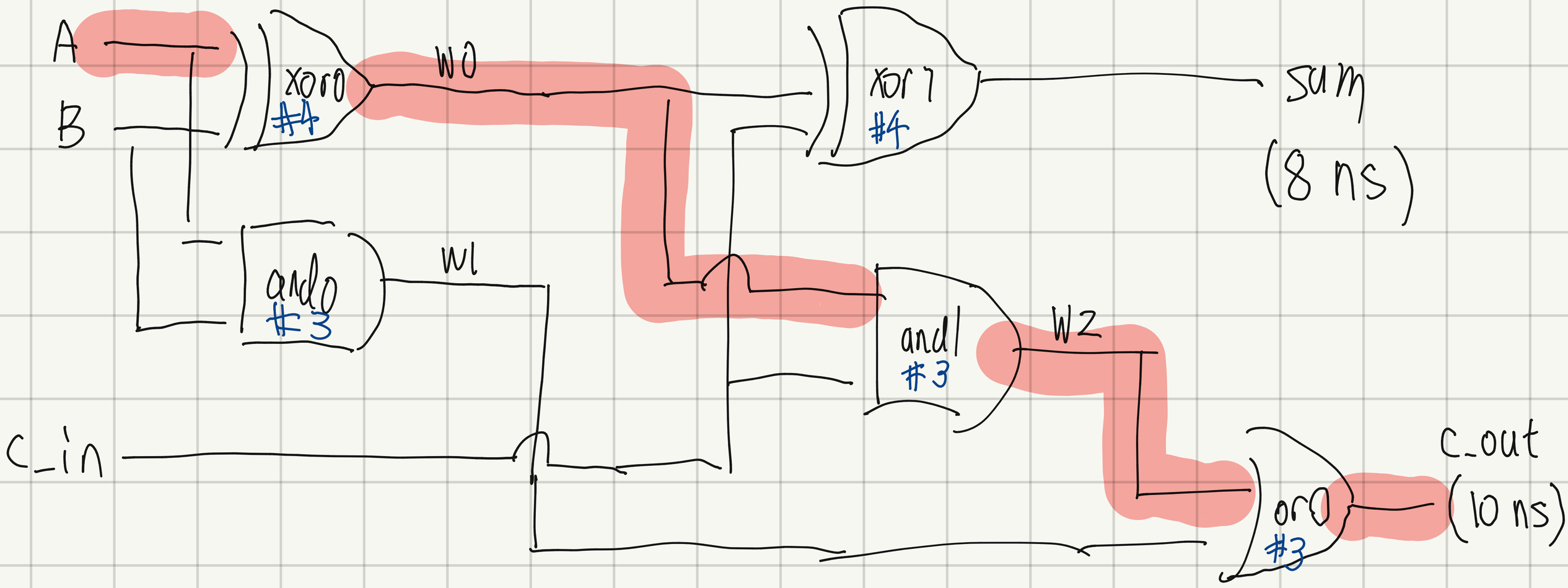


instruction 3'b00000100 branch



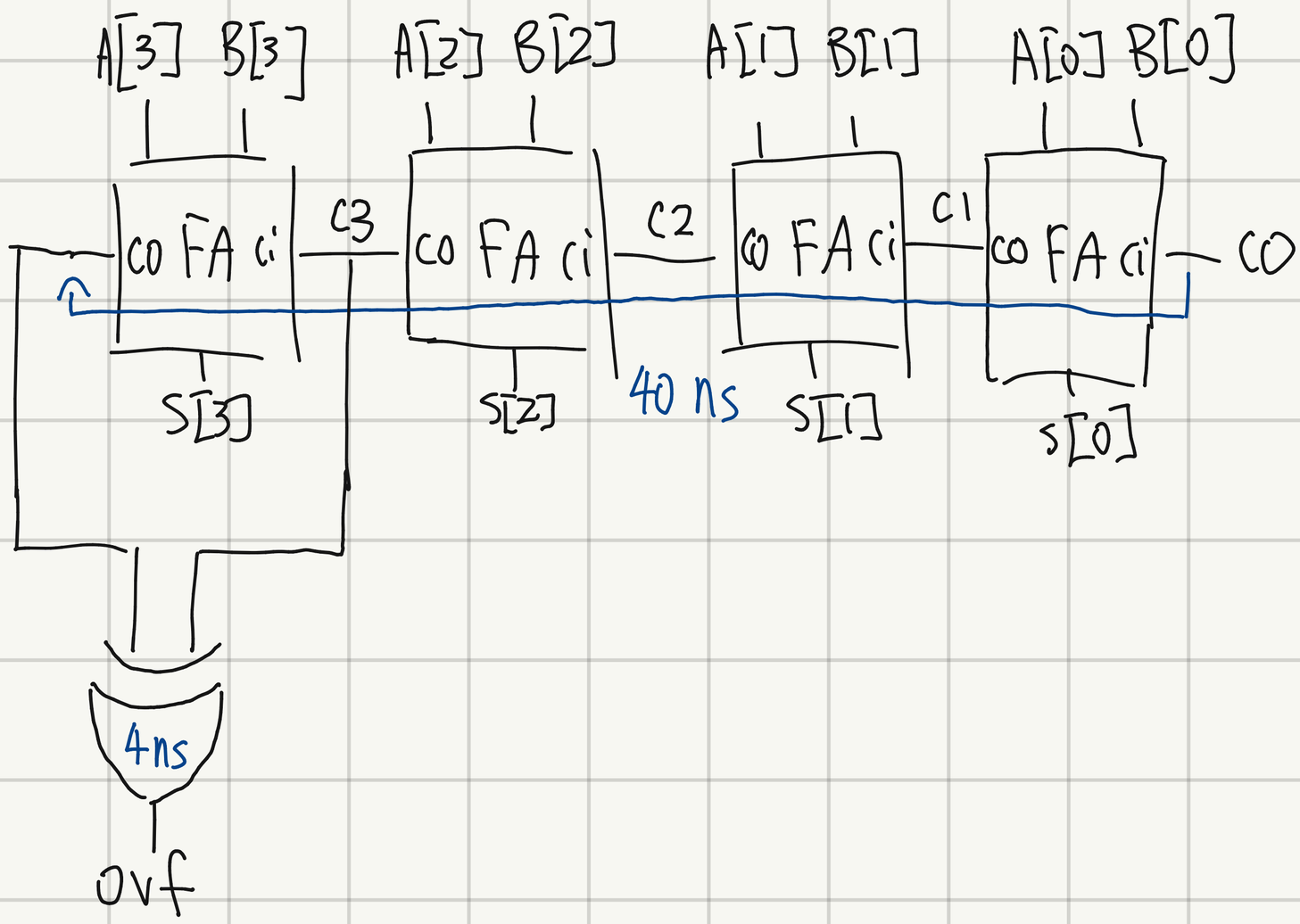

```
if(*reg_a == *reg_b)
        PC_NEXT = PC_CURR + JMP ;
else
        PC_NEXT = PC_CURR + 1;
```

FA



critical path delay: 10ns

RCA



critical path delay = 44ns

A

Overflow part 1

1) If sign(A) != sign(B) ⟶ no overflow

- Proof: A+B's valid range is limited by

max = $0 + (2^{N-1}-1)$ dec. 0 until extreme

min = $0 + -2^{N-1}$ inc. 0 until extreme

extreme: $(2^{N-1}-1) + (-2^{N-1}) = -1 \in$ range (for $N \geq 1$)

2) If sign(A) == sign(B) AND

a. sign(A+B) != sign(A) -> ovf bc result makes no sense

b. sign(A+B) == sign(A) -> no ovf, see proof

Prove 2b: If you exceed range you flip signs once

Two max pos. $\underbrace{0111\ldots}_{N} + \underbrace{0111\ldots}_{N} = \underbrace{1111\ldots10}_{N}$

Extreme cases (max flips)

Ex. 1111... ← carries

0111...

+ 0111...

1110

only 1 flip

Two -max neg. $\underbrace{1000\ldots}_{N} + \underbrace{1000\ldots}_{N} = \underbrace{10000\ldots}_{N+1}$

- Let P = sign(A) == sign(B)
- Let Q = sign(A+B) == sign(A)
- Let R = overflow has occurred

From 1) $\bar{P} \to \bar{R}$, so $R \to P$

From 2a) $P\bar{Q} \to R$

2b) $PQ \to \bar{R}$, so $R \to \overline{PQ}$ but $R \to P$ so $R \to P\bar{Q}$

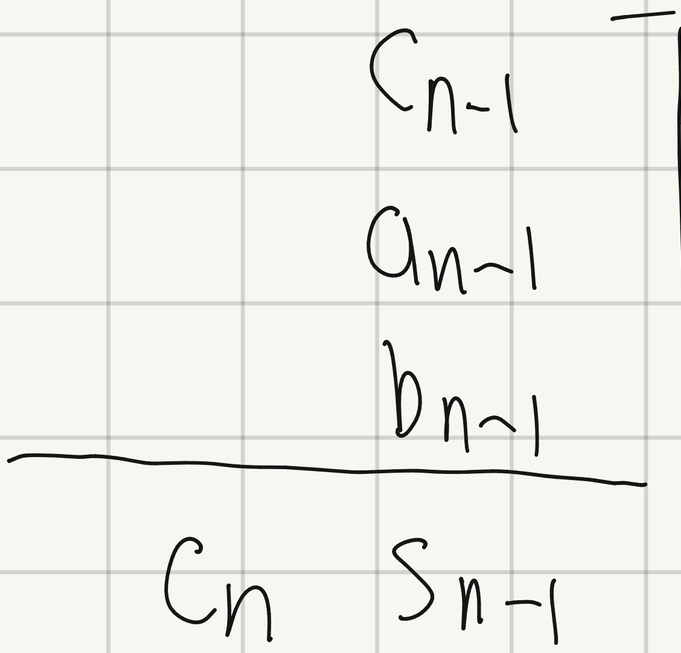
Then, $R \Leftrightarrow P\bar{Q}$, now we derive an eq. using TT

∴ If you didn't flip signs you didn't exceed range

"exceed range" means ovf!

Overflow part 2

We know $R \leftrightarrow P\bar{Q}$, i.e. overflow iff sign(A) == sign(B) and sign(A+B) != sign(A)



Inputs			outputs		
a_{n-1}	b_{n-1}	c_{n-1}	c_n	s_{n-1}	ovf
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

ovf iff $c_{n-1} \neq c_n$,
i.e. $ovf = c_n \oplus c_{n-1}$

Overflow examples:

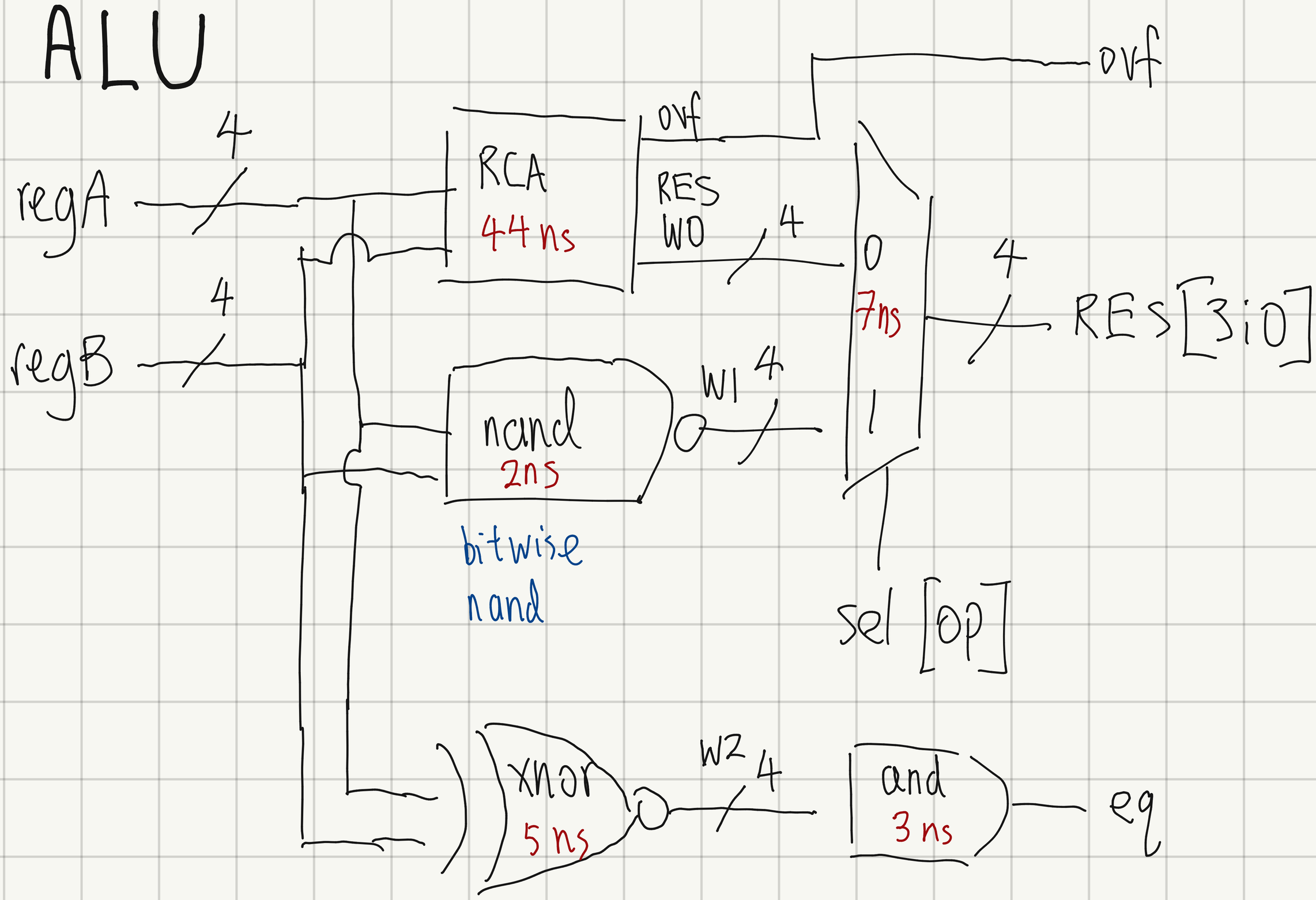
-8 + -3 = 5 (should be -11)

$$\begin{array}{cccc} & 1 & & \\ 1 & 0 & 0 & 0 \\ + & 1 & 1 & 0 & 1 \\ \hline 0 & 1 & 0 & 1 \end{array}$$

5+5 = -6 (should be 10)

$$\begin{array}{cccc} & 1 & & 1 \\ 0 & 1 & 0 & 1 \\ + & 0 & 1 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 \end{array}$$

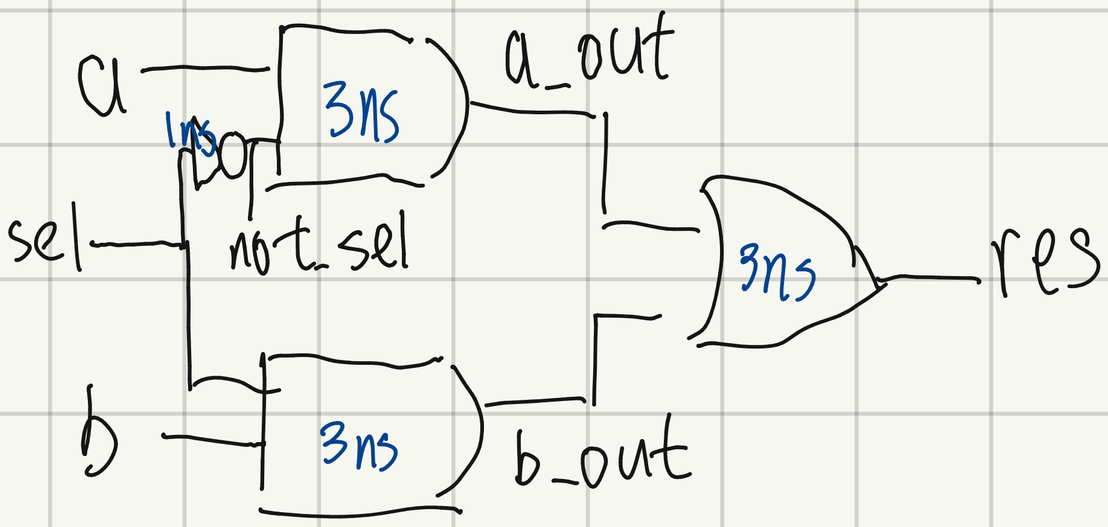
ALU



Critical path delay: 51 ns

MUX_2_1 (1b/4b)

1-bit multiplexer ☑



critical path delay = 7 ns

testbench ☑

a	b	sel	res	
0	0	0	0	// copy a
0	1	0	0	
1	0	0	1	
1	1	0	1	
0	0	1	0	// copy b
0	1	1	1	
1	0	1	0	
1	1	1	1	

4-bit multiplexer (4 1-bit muxes controlled by the same sel) ☑

Partial testbench: ☑ cpd = 7ns

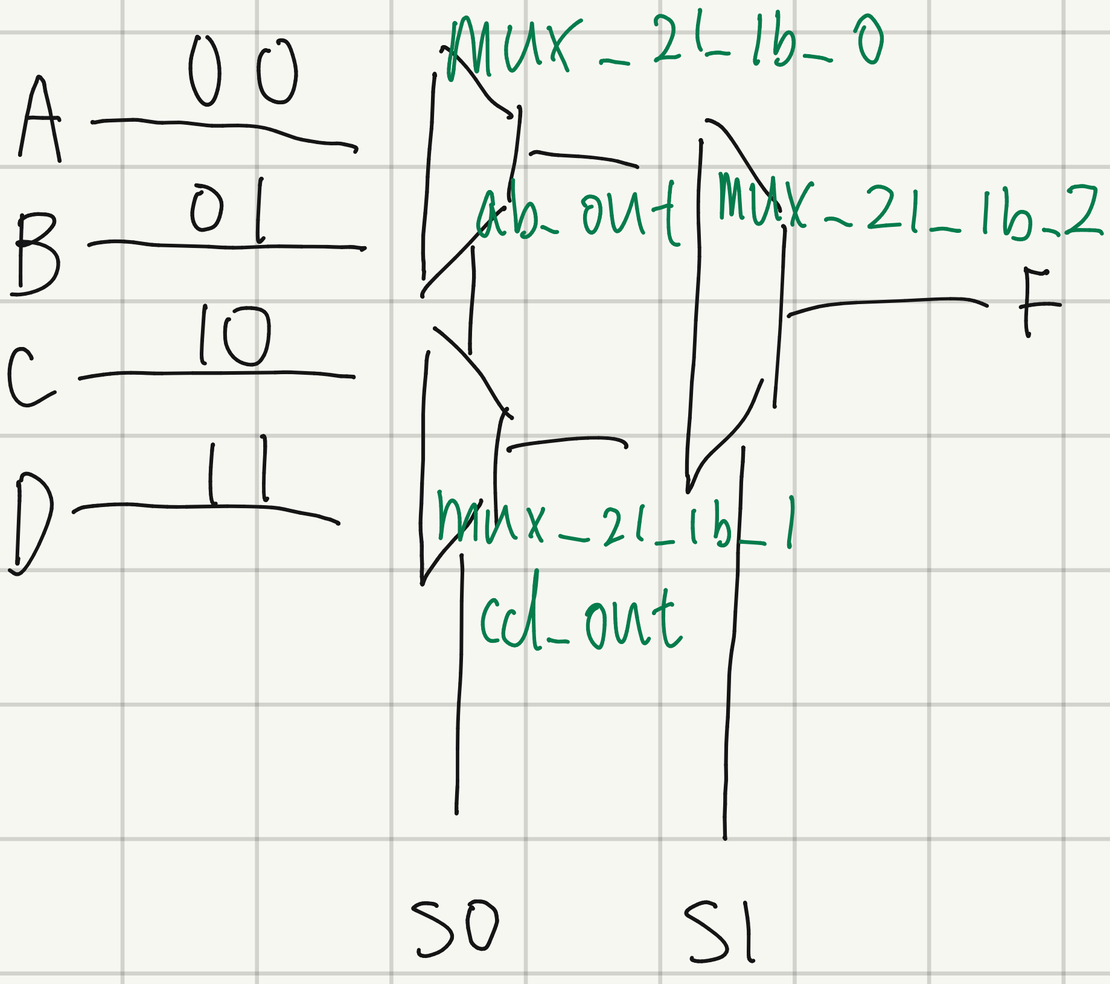
A[3] = 0 B[3] = 0 sel = 0 res[3] = 0

A[2] = 0 B[2] = 1 res[2] = 0

A[1] = 1 B[1] = 0 res[1] = 1

A[0] = 1 B[0] = 1 res[0] = 1

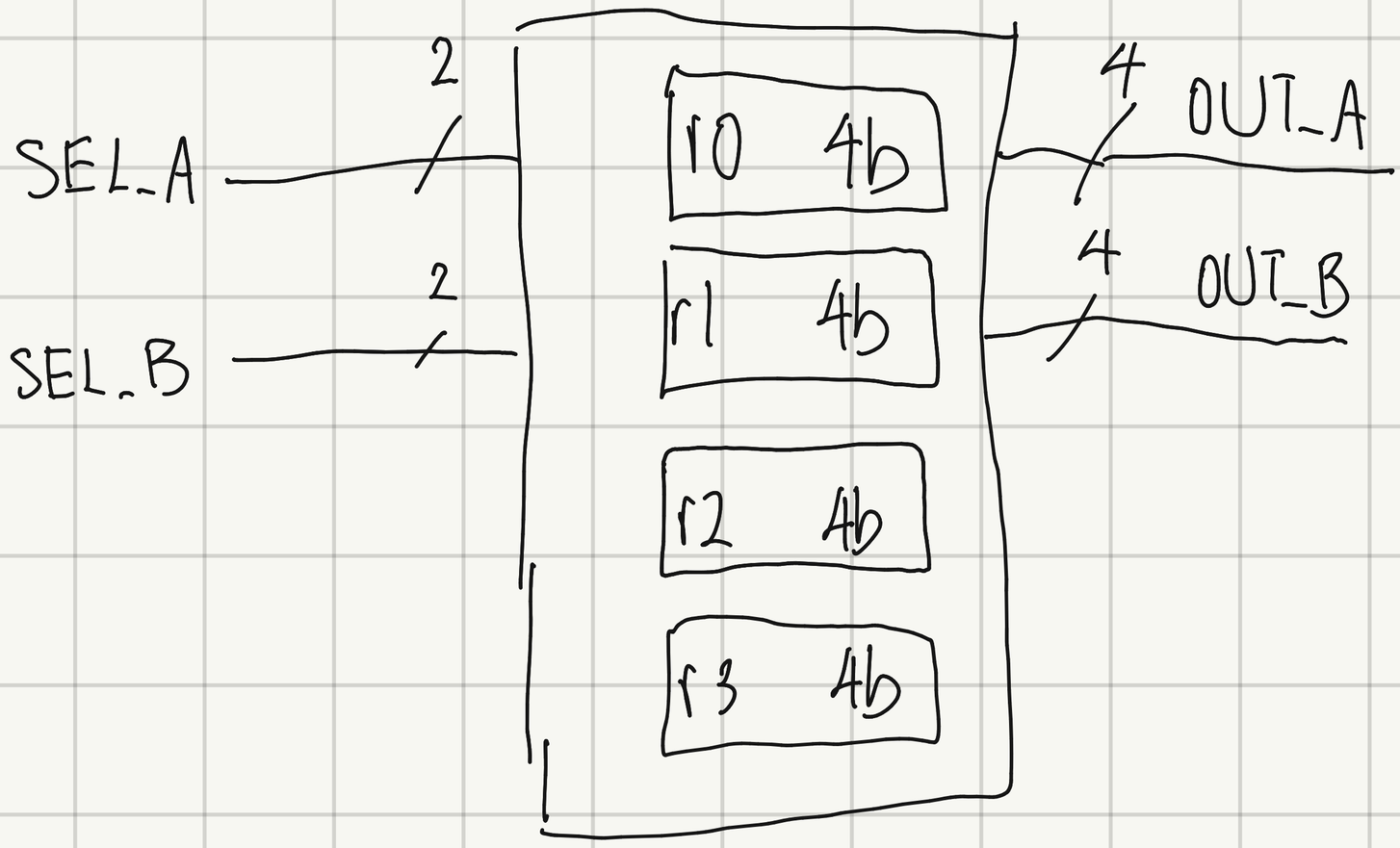
MUX_4_1_1b

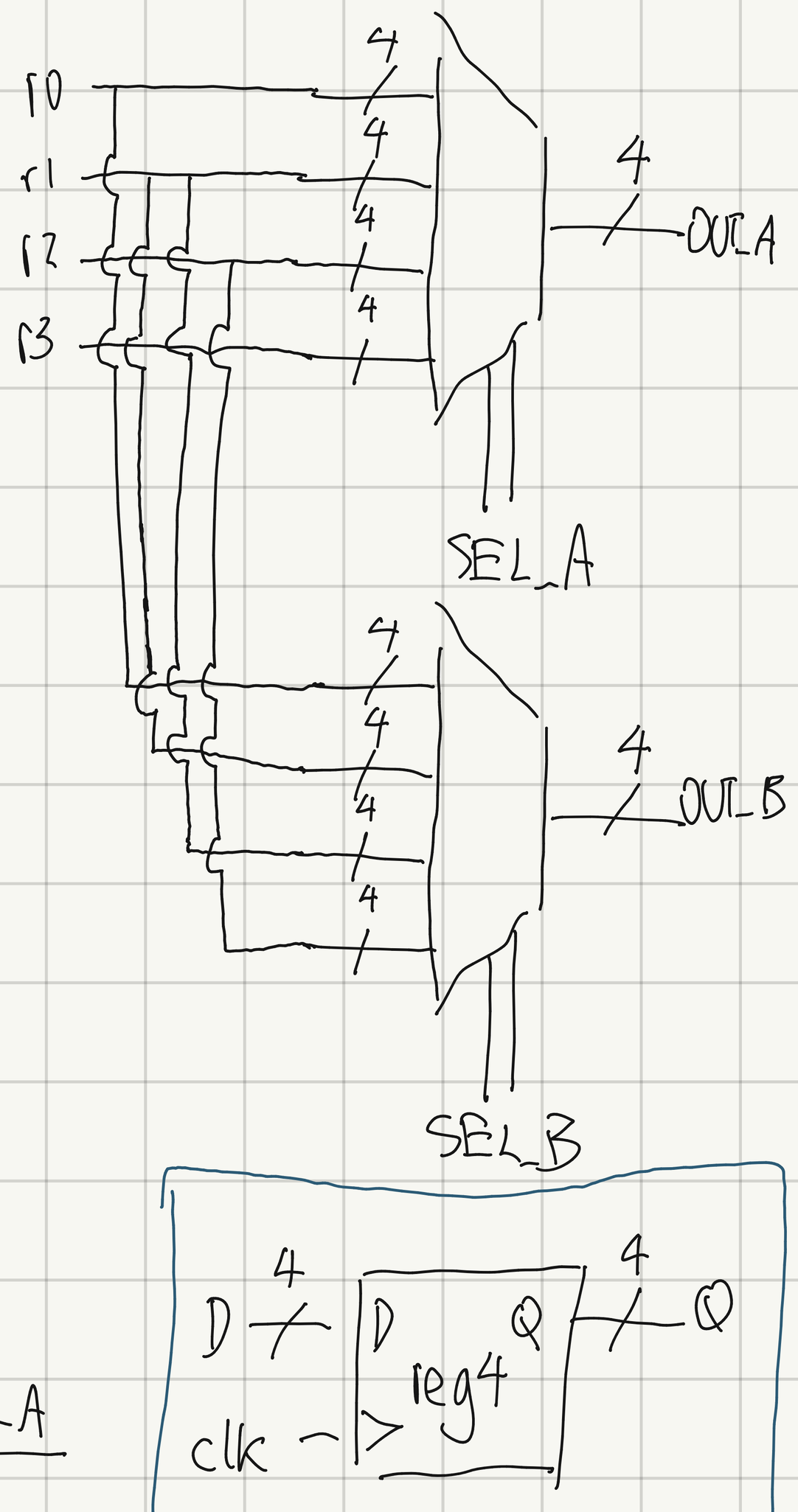


S1	S0	F
0	0	A
0	1	B
1	0	C
1	1	D

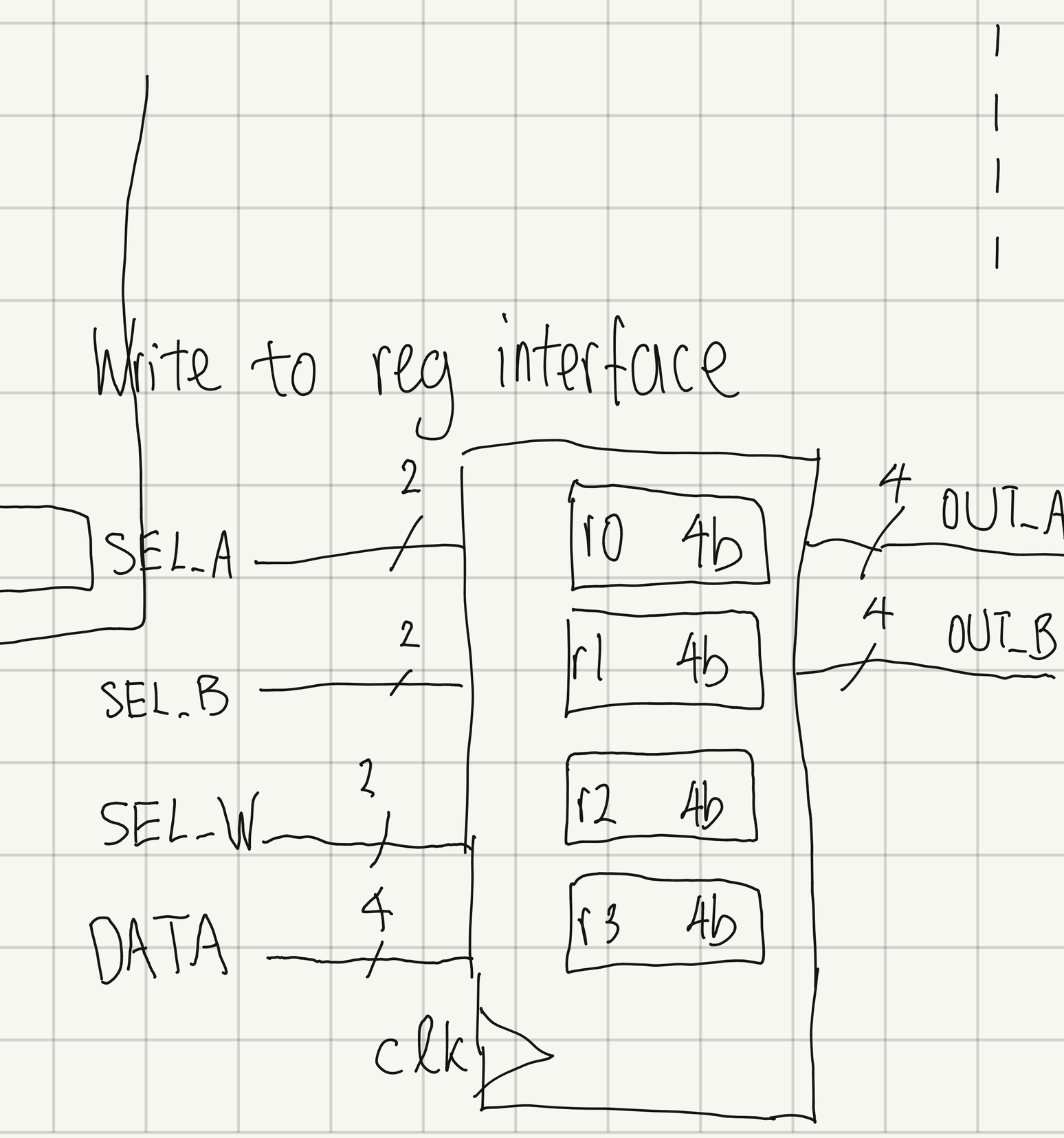
reg_file part 1

Let OUT_A return contents of SEL_A
let OUT_B return contents of SEL_B

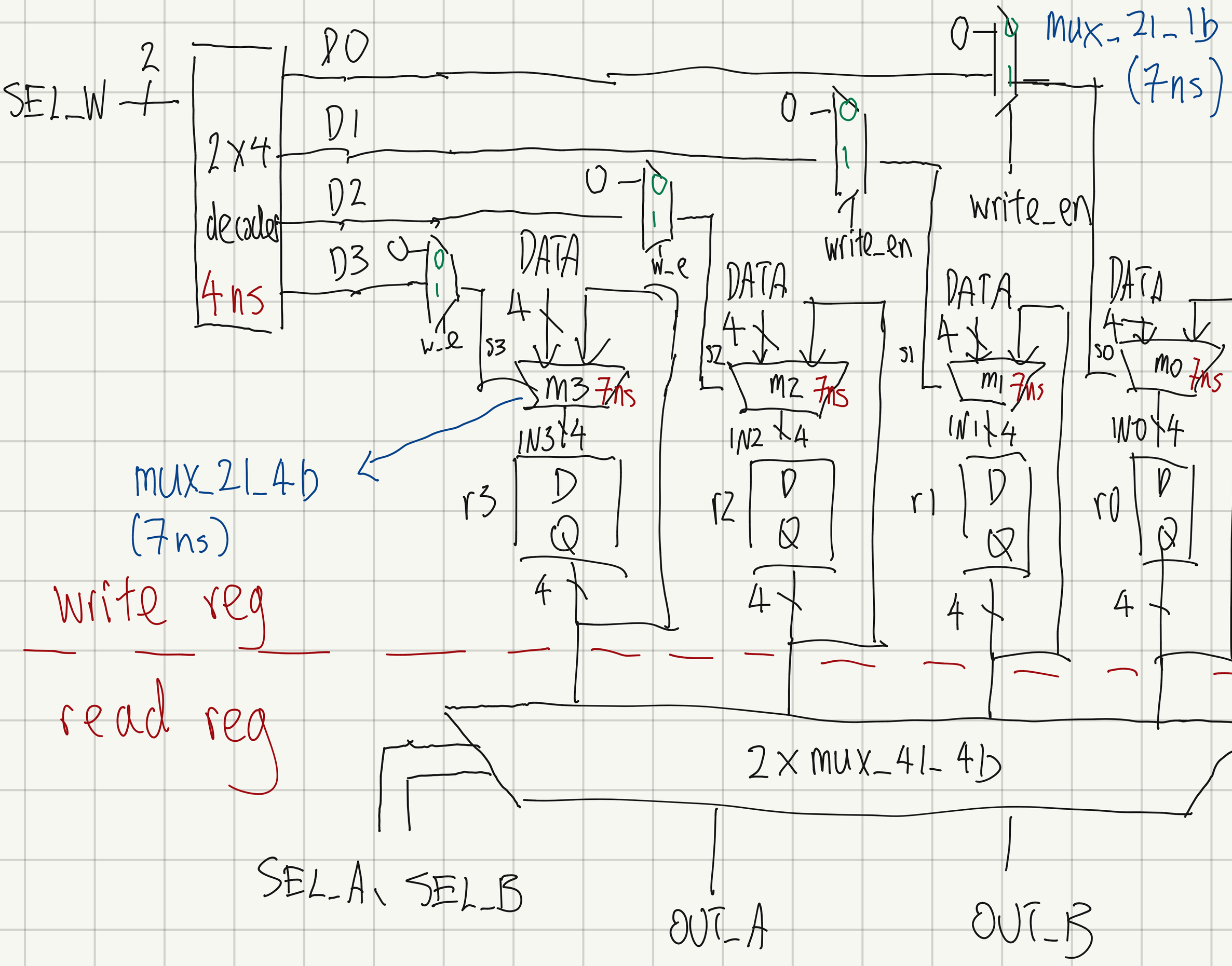




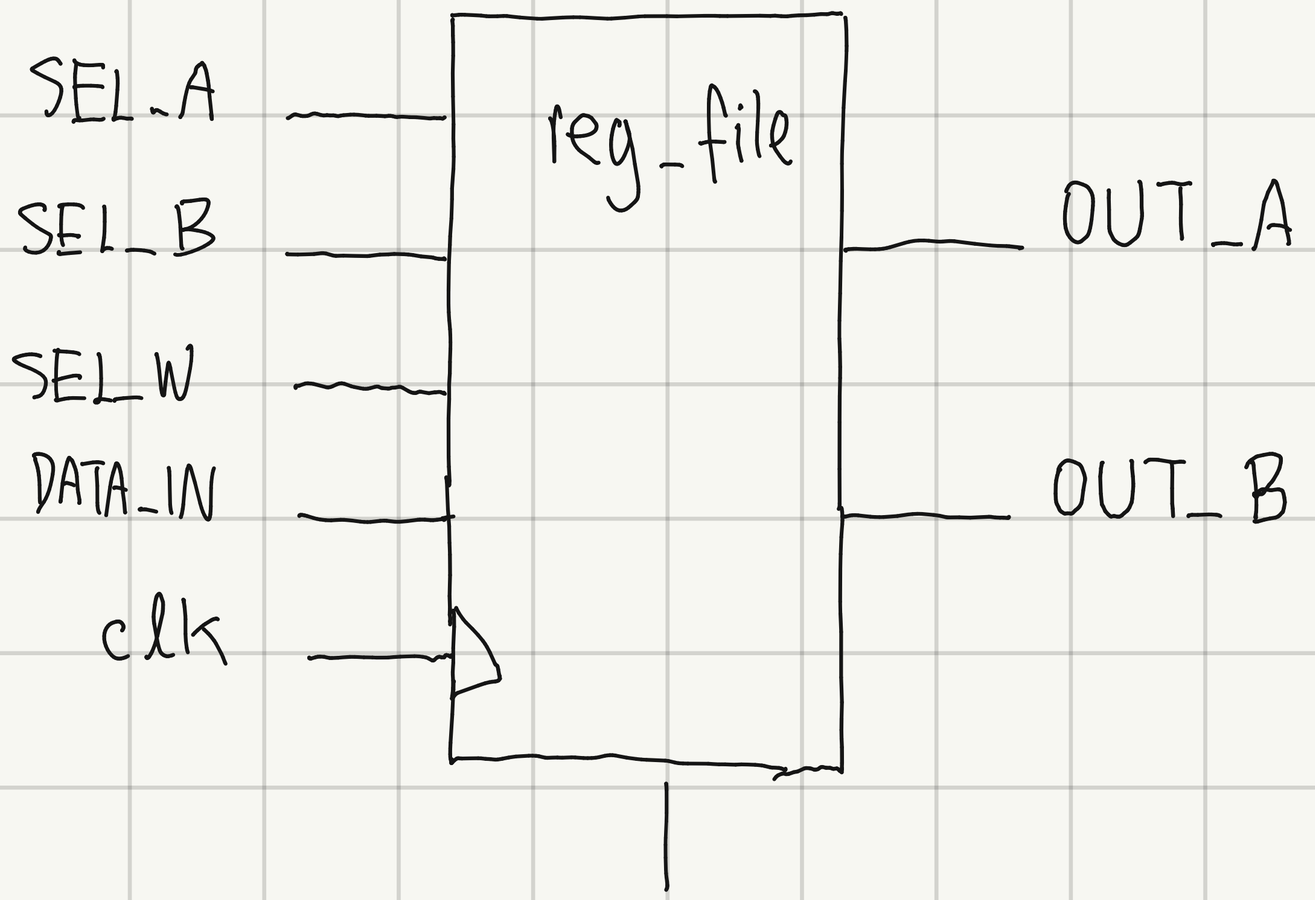
Write to reg interface



reg_file part 2



reg_file part 3



DATA_IN = 4'b0101 →

decoder → mux_2_1_1b → mux_2_1_4b → reg4 setup → clk 0→1 →

(4ns) (7ns) (7ns) (17 ns)

reg write setup (35 ns)

reg4 hold → mux_4_1_4b → OUT_A = 4'b0101

(17 ns) (14ns)

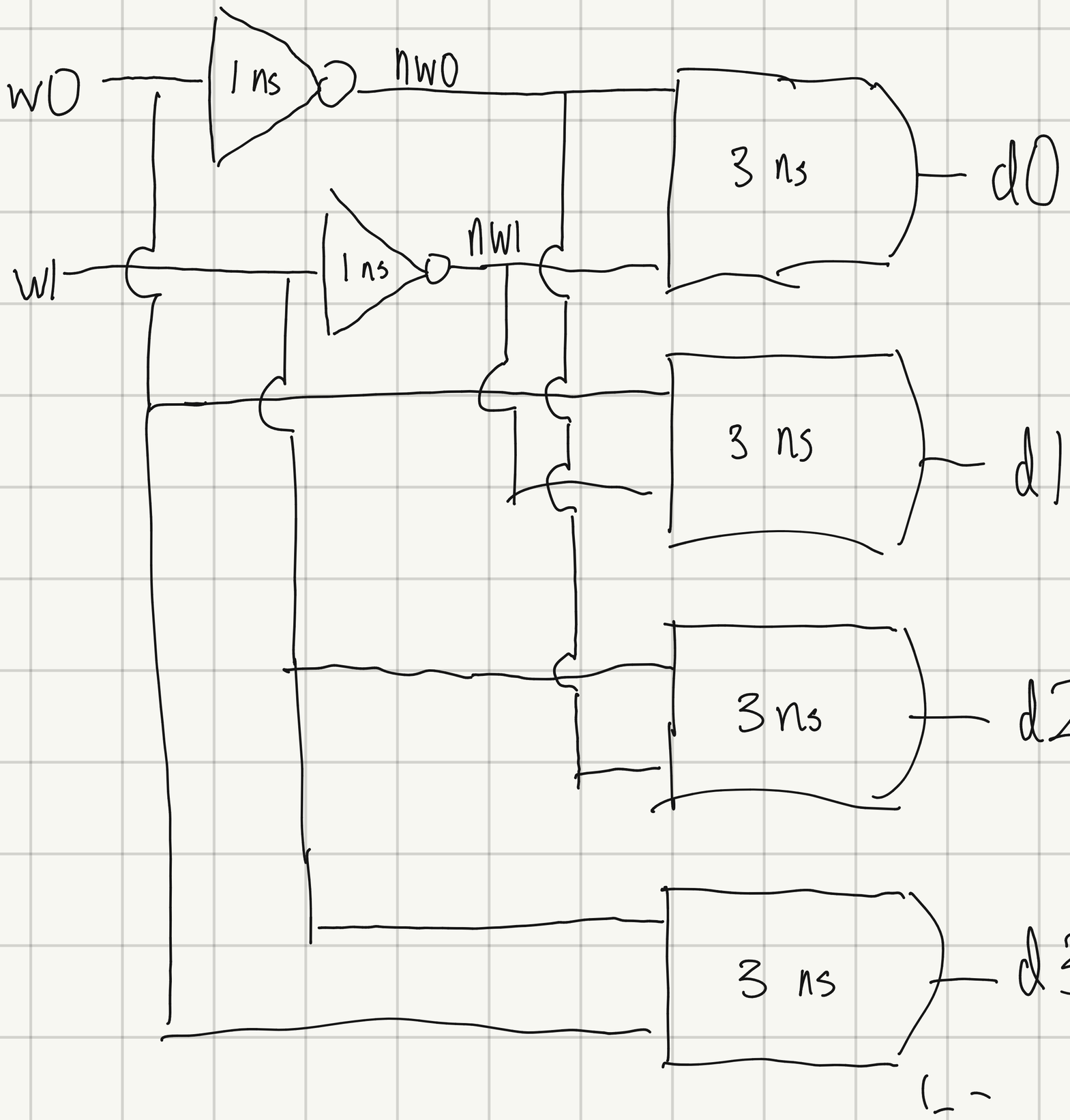
reg write hold (17 ns) reg read (14 ns)

Total delay = 66 ns

写入的时候等setup+hold，读取的时候立刻取值。

可以同时读写 → write_en

decoder_2_4_1b



w1	w0	d3	d2	d1	d0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

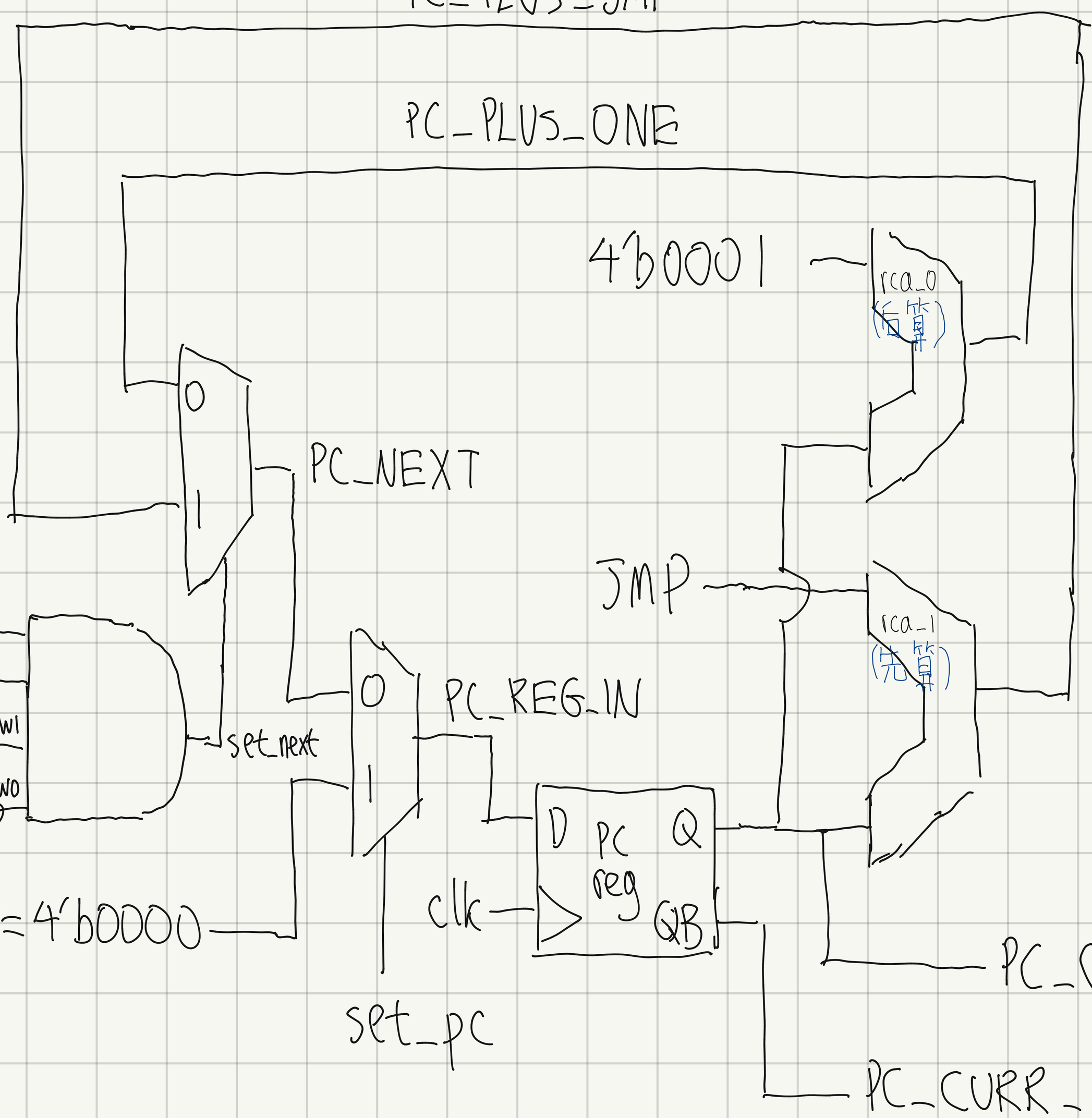
critical path delay = 4 ns

PC part 1

PC init and branch implementation

PC keep increasing 0-15, then repeat

Calculate both RCAs in paralle1, before clk 0→1



cycle 0 (set_pc=1):

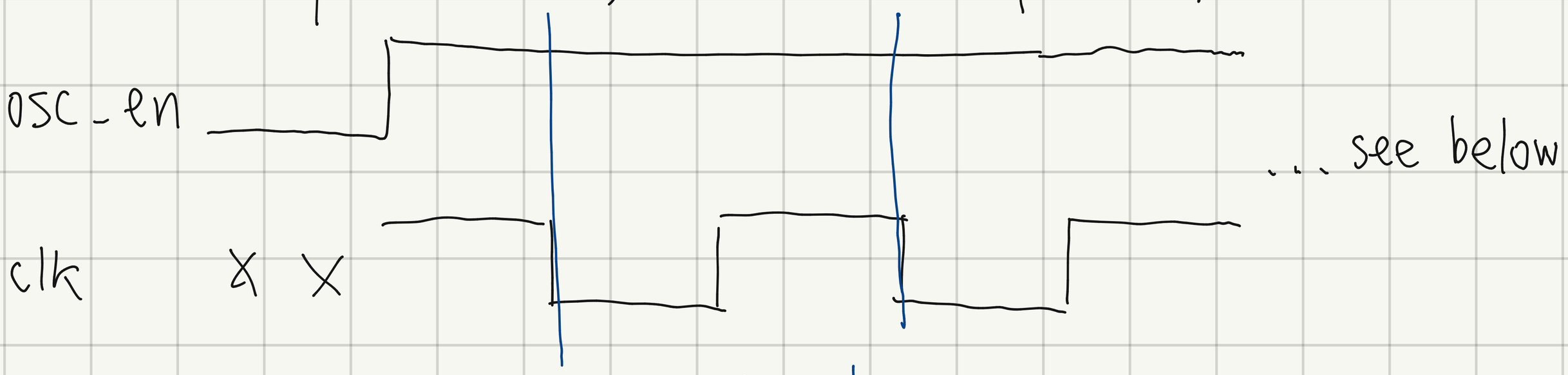
max(not + and, rca) + mux_2_1_4b + mux_2_1_4b + reg_set

1ns 3ns 44ns 7ns 7ns 17ns

reg_hold 17ns → clk 0→1 → setup: 75 ns, hold: 17 ns

PC part 2

Test PC loops around, osc shown for ref



set_pc=1 test 0
PC_CURR=4'bxxx
INS=>noop
11'b01100000000

set_pc=0 test 1
PC_CURR=4'b000

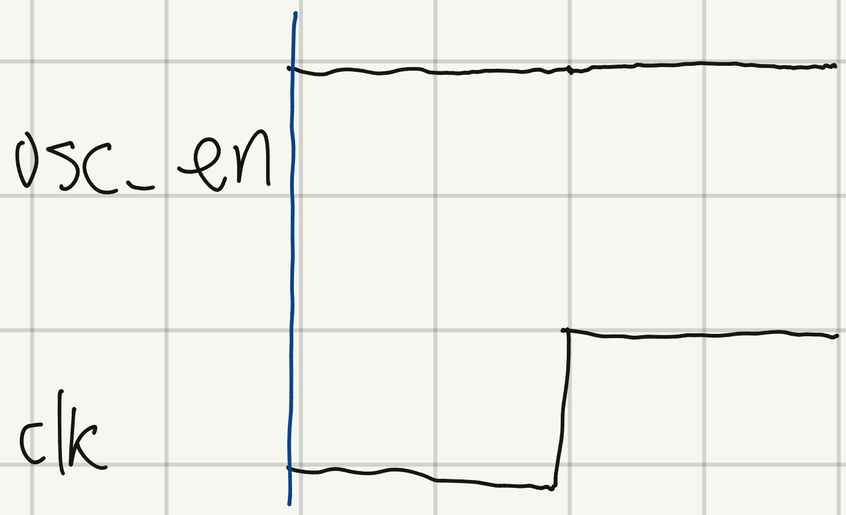
test 16
PC_CURR=4'b1111

test 17
PC_CURR=4'b0000
INS=>branch+7
11'b10001110000

pre-set
alu_eq=1

test 18
PC_CURR=4'b0111
ins=>branch-2
11'b10011100000

test 19
PC_CURR=4'b0101
ins=>branch+0
11'b10000000000



test 20
PC_CURR=4'b0101

branch.tass (program to test branching)

Pseudocode:

```
let stdout = r0, initialized as 0
move  r1, #1
move  r2, #2
move  r3, #3
if (r1 + r2 == r3)
    print 1
else
    print 2
```

TomAssembly:

```
0.  move  r0, 4'b0000
1.  move  r1, 4'b0001
2.  move  r2, 4'b0010
3.  move  r3, 4'b0011
4.  add   r3, r1, r2
5.  branch 4'b0011, r1, r3
6.  move  r0, 4'b0010
7.  branch 4'b0010, r0, r0
8.  move  r0, 4'b0001
9.  noop  // end of program
```

also try w/ nand

add:

```
  0001
+ 0010
------
  0011
```

nand:

```
  0001
↑ 0010
------
  1111
```

back.tass (backward branching)

Pseudocode:

```
let stdout = r0, initialized as 0
move r1, #-1
move r2, #2
move r3, #3
if (r1 == r2) // first time not taken
    branch to end program
r1 = r1 + r3
branch back
```

Tom Assembly:

```
0. move r0, 4'b0000
1. move r1, 4'b1111
2. move r2, 4'b0010
3. move r3, 4'b0011
4. branch 4'b0011, r1, r2
5. add r1, r1, r3
6. branch 4'b1110, r0, r0
7. noop // end of program
```

add:

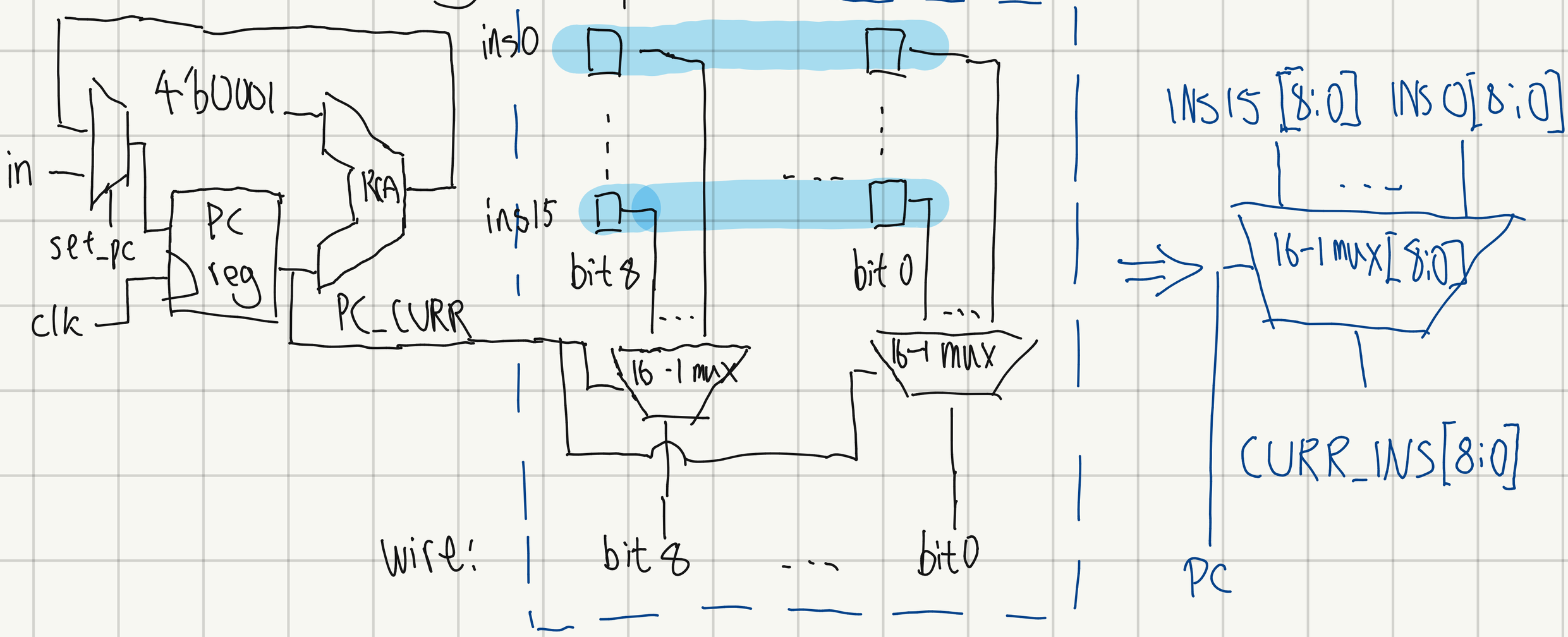
```
  0001
+ 0010
------
  0011
```

nand:

```
  0001
↑ 0010
------
  1111
```

ins_mem

Instruction memory (support 16 instructions, each 9b wide)



ins_dec part 1

instruction	op2 INS[10]	op1 INS[9]	op0 INS[8]	sel_data	alu_op
add	0	0	0	0	0
nand	0	0	1	0	1
push	0	1	0	1	X
noop	0	1	1	X	X
branch	1	0	0	X	X

sel_data = op1 = INS[9]

write_en = see next page

alu_op = op0 = INS[8]

SEL_A = INS[3:2]

SEL_B = INS[1:0]

SEL_W = INS[5:4]

IMM = INS[3:0]

JMP = INS[7:4]

ins_dec part 2

instruction	op2	op1	op0	write_en
add	0	0	0	1
nand	0	0	1	1
push	0	1	0	1
noop	0	1	1	0
branch	1	0	0	0

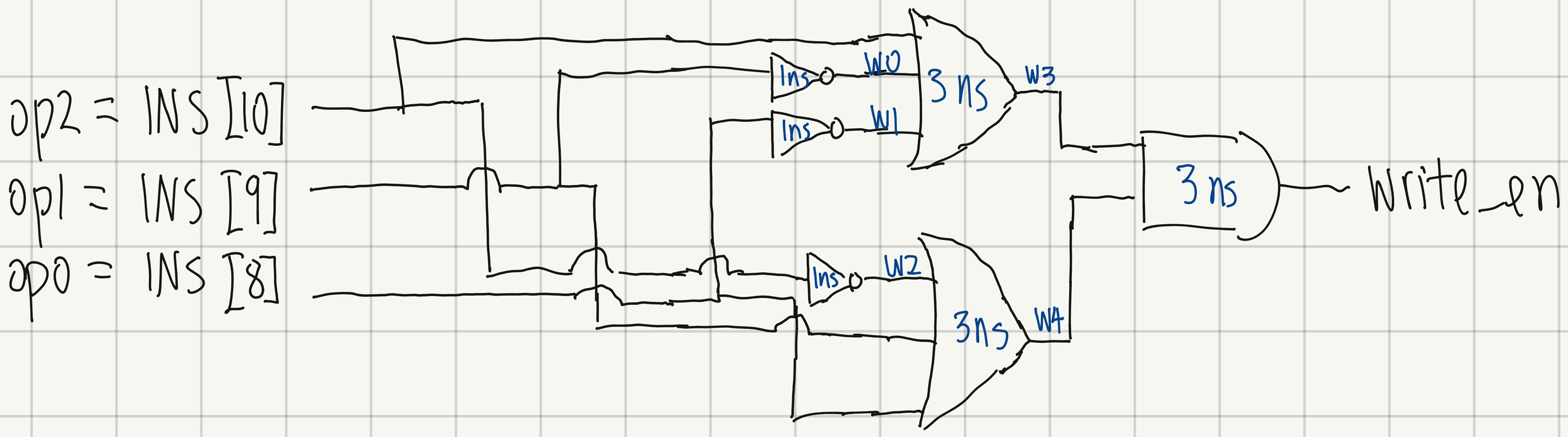
write_en sum of minterms = $\overline{op2}\ \overline{op1}\ \overline{op0} + \overline{op2}\ \overline{op1}\ op0 + \overline{op2}\ op1\ \overline{op0}$

product of maxterms = $\neg(\overline{op2}\ op1\ op0 + op2\ \overline{op1}\ \overline{op0})$

$= \neg(\overline{op2}\ op1\ op0) \cdot \neg(op2\ \overline{op1}\ \overline{op0})$

$= (op2 + \overline{op1} + \overline{op0}) \cdot (\overline{op2} + op1 + op0)$

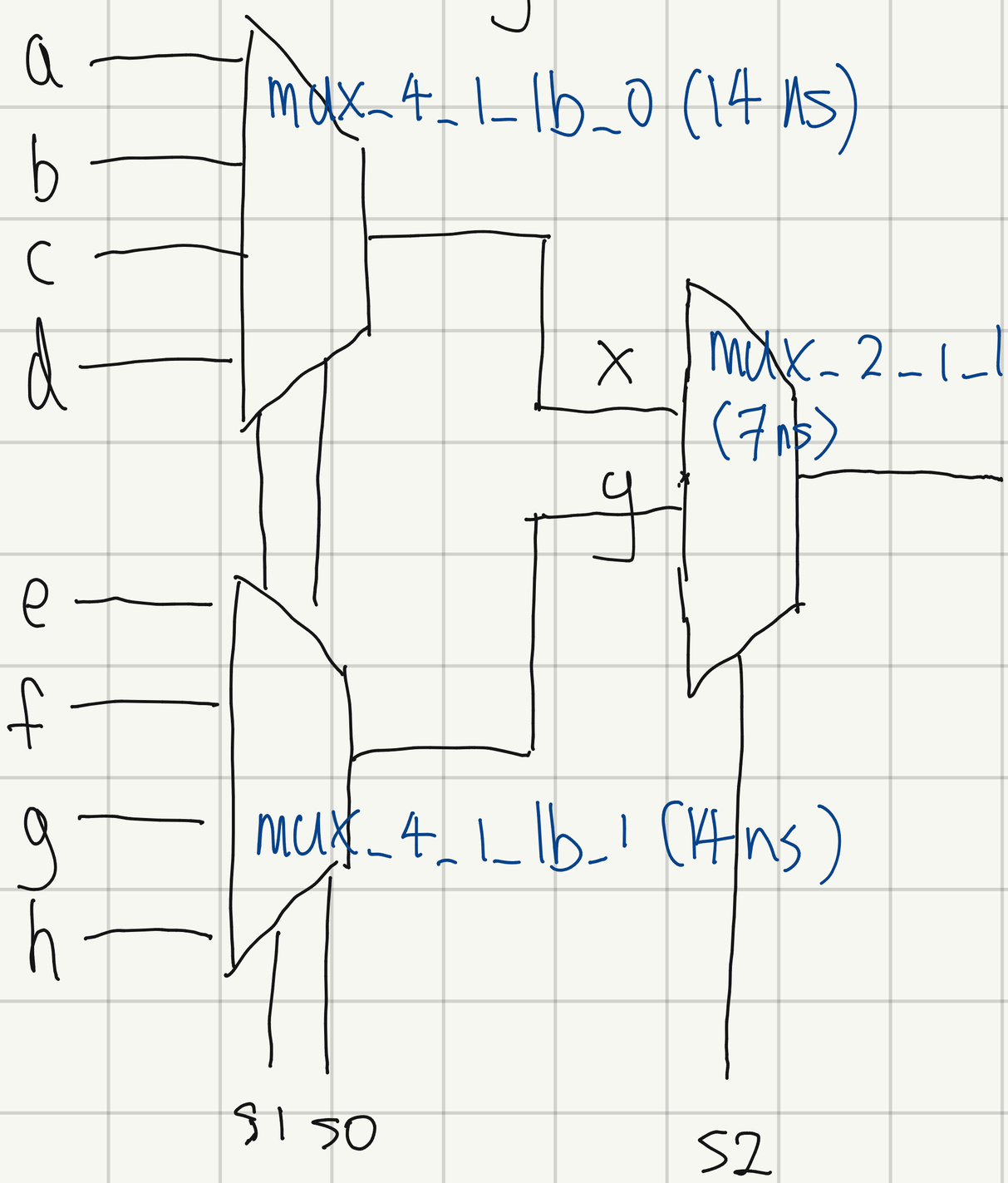
less terms



Critical path delay = 7 ns

MUX_8_1_1b

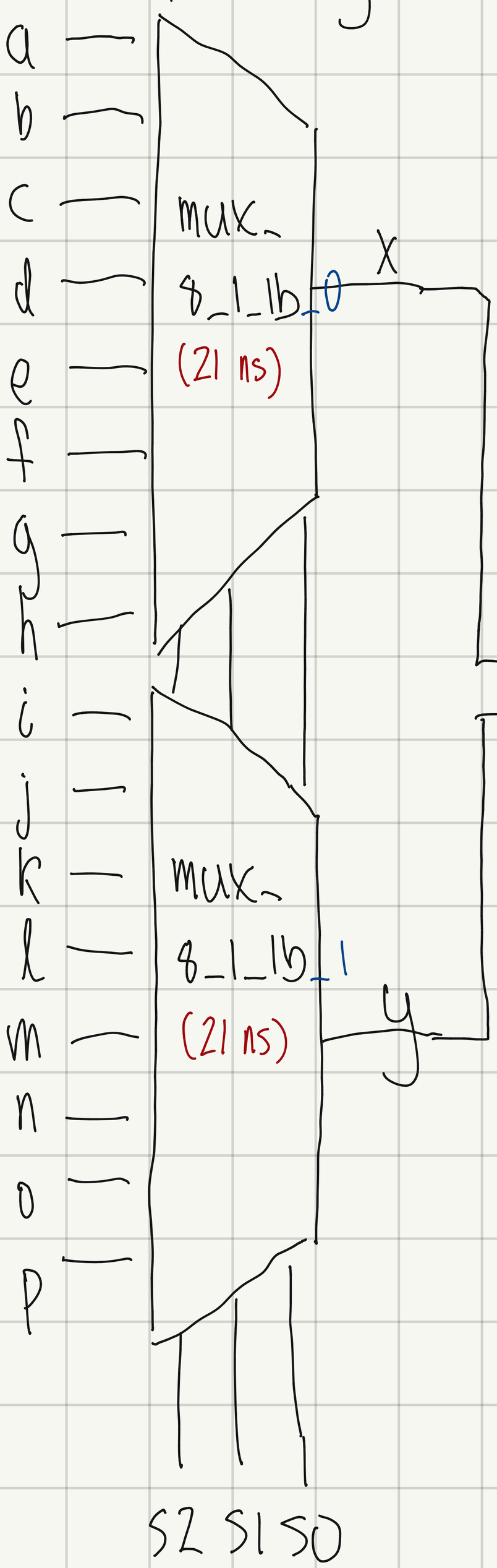
Critical path delay = 21 ns



S2	S1	S0	x	y	f
0	0	0	a	e	a
0	0	1	b	f	b
0	1	0	c	g	c
0	1	1	d	h	d
1	0	0	a	e	e
1	0	1	b	f	f
1	1	0	c	g	g
1	1	1	d	h	h

MUX_16_1_16

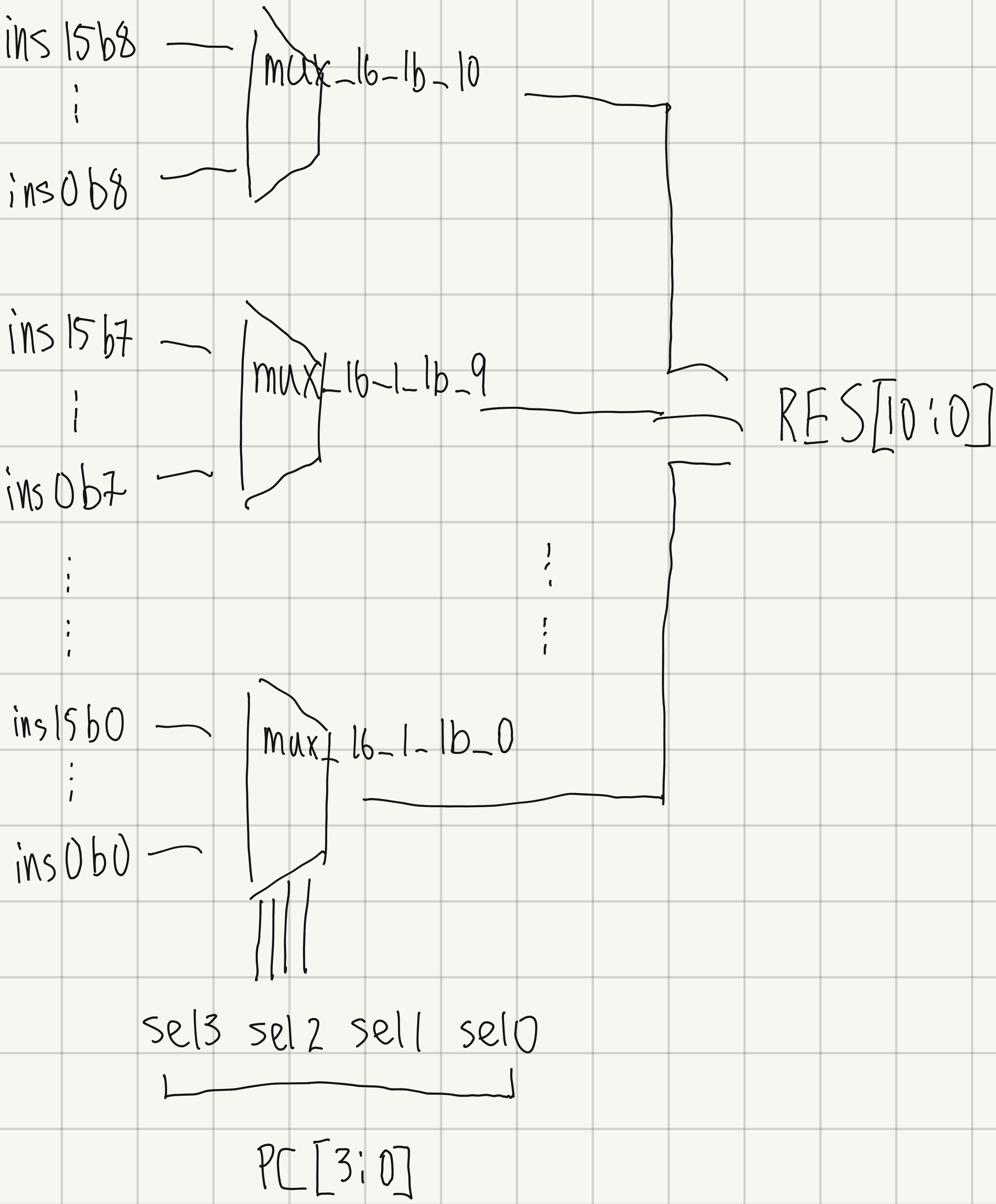
Critical path delay = 28 ns



S3	S2	S1	S0	x	y	res
0	0	0	0	a	i	a
0	0	0	1	b	j	b
0	0	1	0	c	k	c
0	0	1	1	d	l	d
0	1	0	0	e	m	e
0	1	0	1	f	n	f
0	1	1	0	g	o	g
0	1	1	1	h	p	h
1	0	0	0	a	i	i
1	0	0	1	b	j	j
1	0	1	0	c	k	k
1	0	1	1	d	l	l
1	1	0	0	e	m	m
1	1	0	1	f	n	n
1	1	1	0	g	o	o
1	1	1	1	h	p	p

MUX_16_1_96 part 1

critical path delay = 28 ns

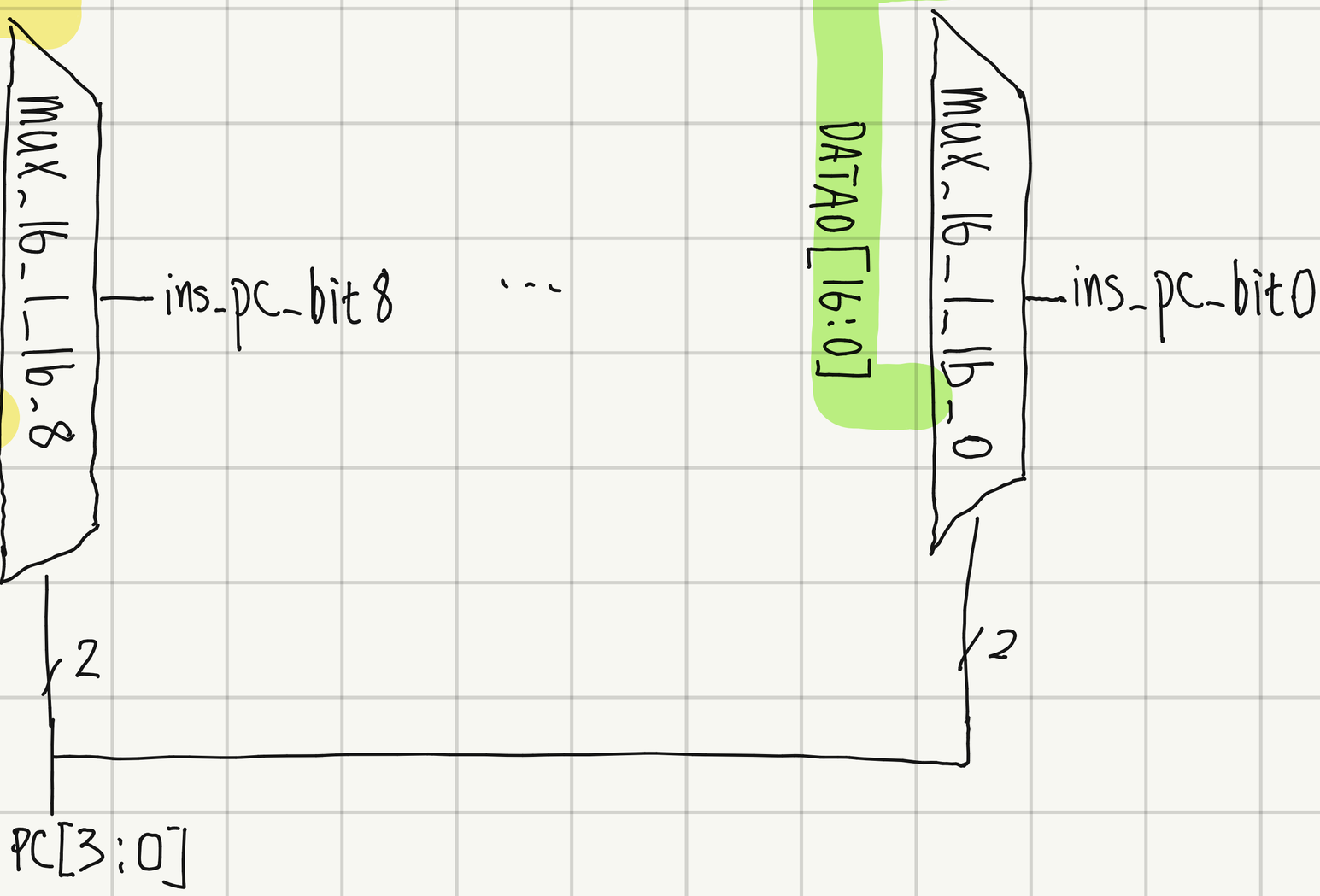


MUX_16-1_9b part 2

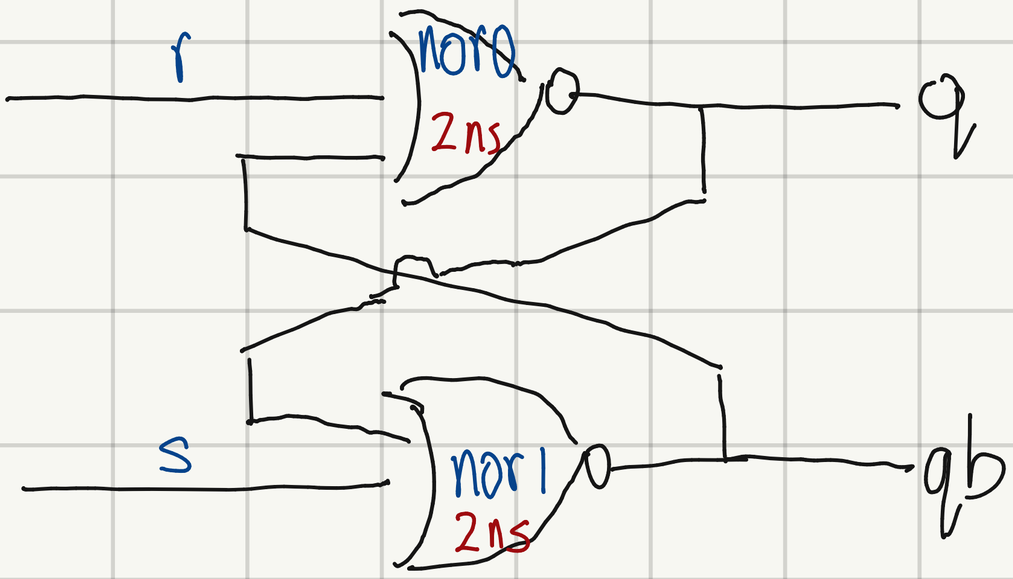
	bit8	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
INS 0	0	0	0	0	0	0	0	0	1
INS 1	0	0	0	0	0	0	0	1	0
INS 2	0	0	0	0	0	0	1	0	0
INS 3	0	0	0	0	0	1	0	0	0
INS 4	0	0	0	0	1	0	0	0	0
INS 5	0	0	0	1	0	0	0	0	0
INS 6	0	0	1	0	0	0	0	0	0
INS 7	0	1	0	0	0	0	0	0	0
INS 8	1	0	0	0	0	0	0	0	0
INS 9	1	1	1	1	1	1	1	1	0
INS 10	1	1	1	1	1	1	1	0	1
INS 11	1	1	1	1	1	1	0	1	1
INS 12	1	1	1	1	1	0	1	1	1
INS 13	1	1	1	1	0	1	1	1	1
INS 14	1	1	1	0	1	1	1	1	1
INS 15	1	1	0	1	1	1	1	1	1

mux_16_1_9b testbench 代码

```
mux_16_1_9b mux_16_1_9b_0[8:0](
        .a(INS0)      ┐ individual
        .b(INS1)      │ inputs
          :           │
        .p(INS15)     ┘
        .sel3(sel3)   ┐
        .sel2(sel2)   │ shared
        .sel1(sel1)   │ inputs
        .sel0(sel0)   ┘
        .res(RES));
```

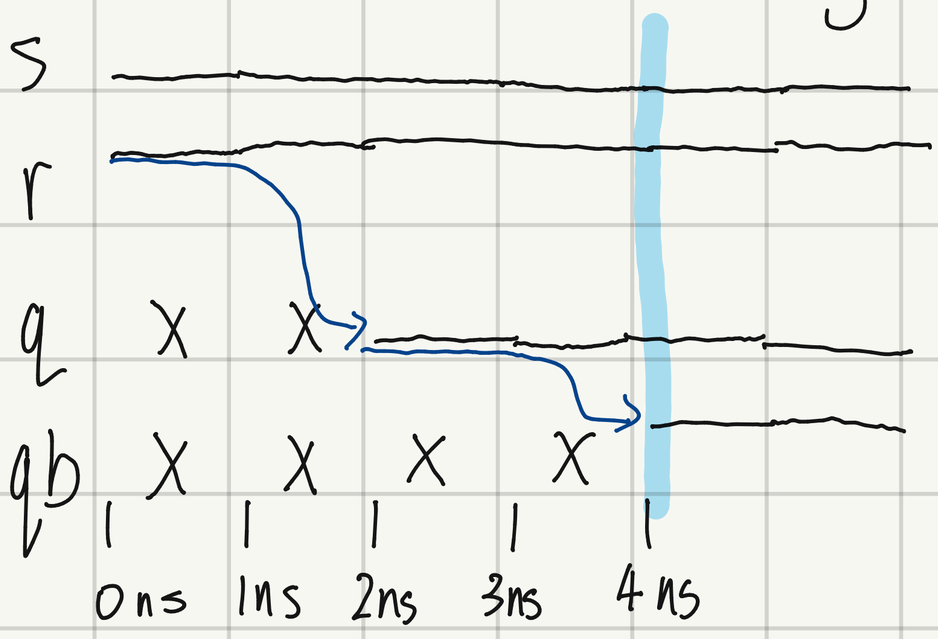


sr_latch

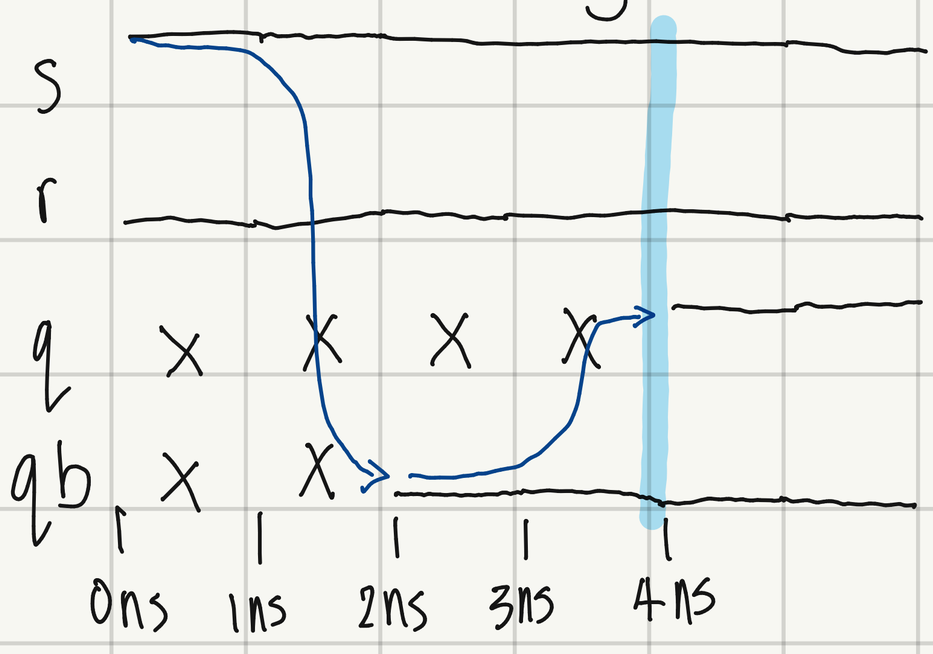


NOR gate:
iff 任何輸入是1，輸出就是0

case 0: reset (delay = 4 ns)

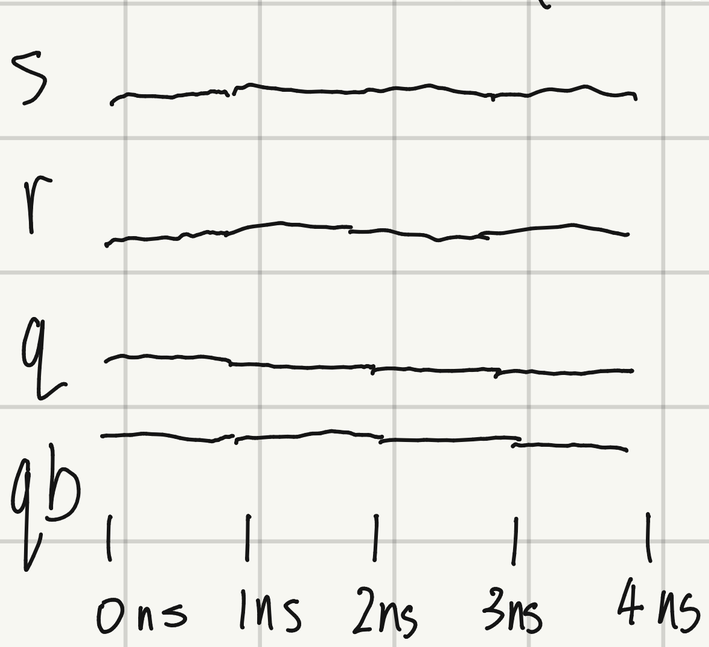


case 1: set (delay = 4 ns)

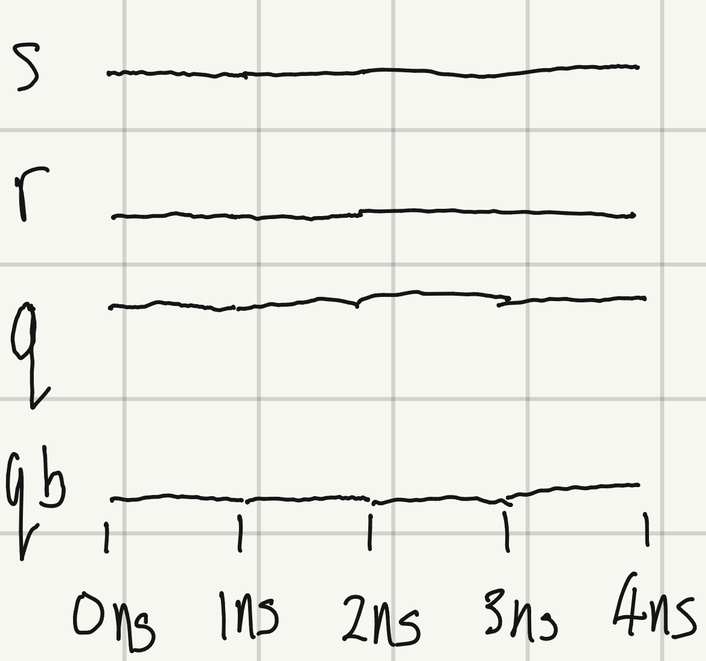


∠ set/reset w/ X means most waiting ⌒

case 2: hold 0 (delay = 0 ns)

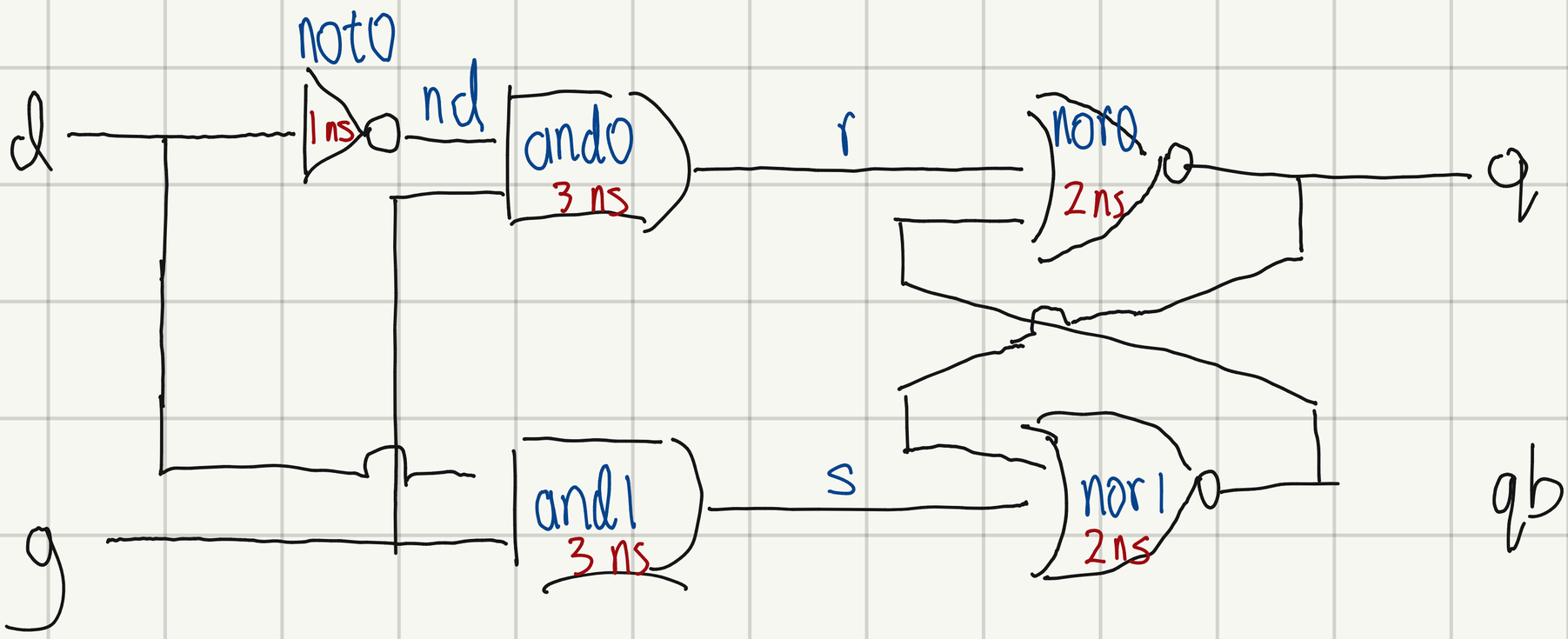


case 3: hold 1 (delay = 0 ns)



∴ Critical path delay = 4ns

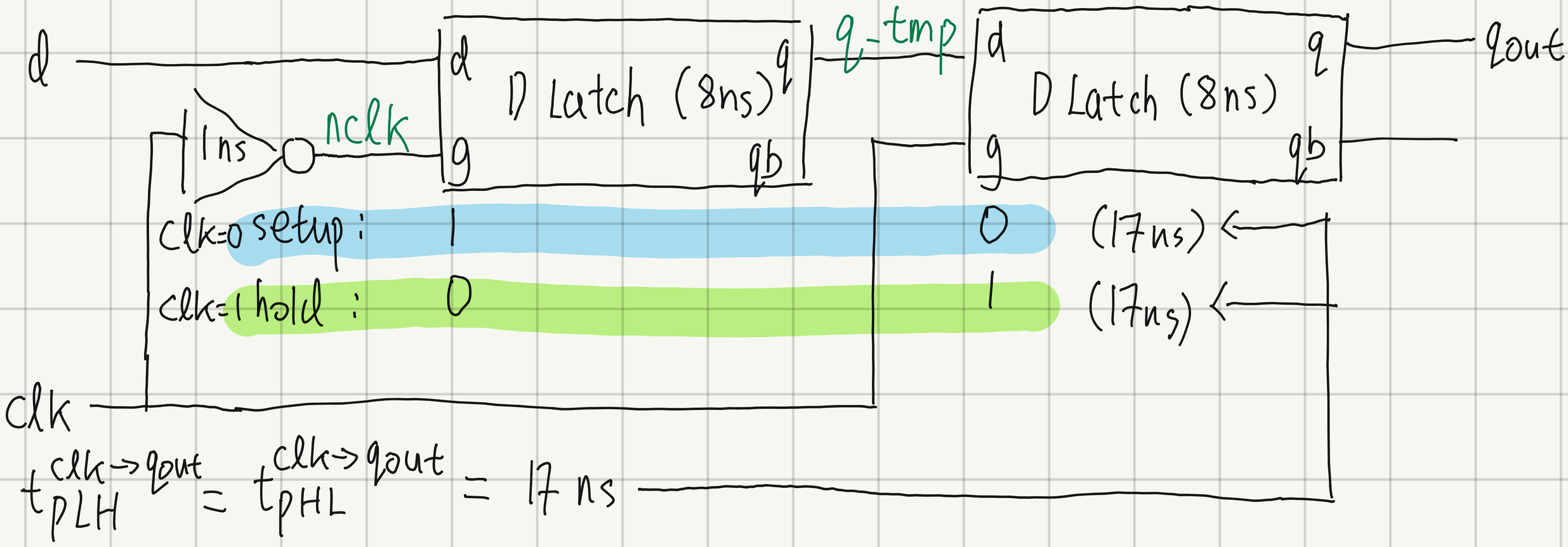
d_latch



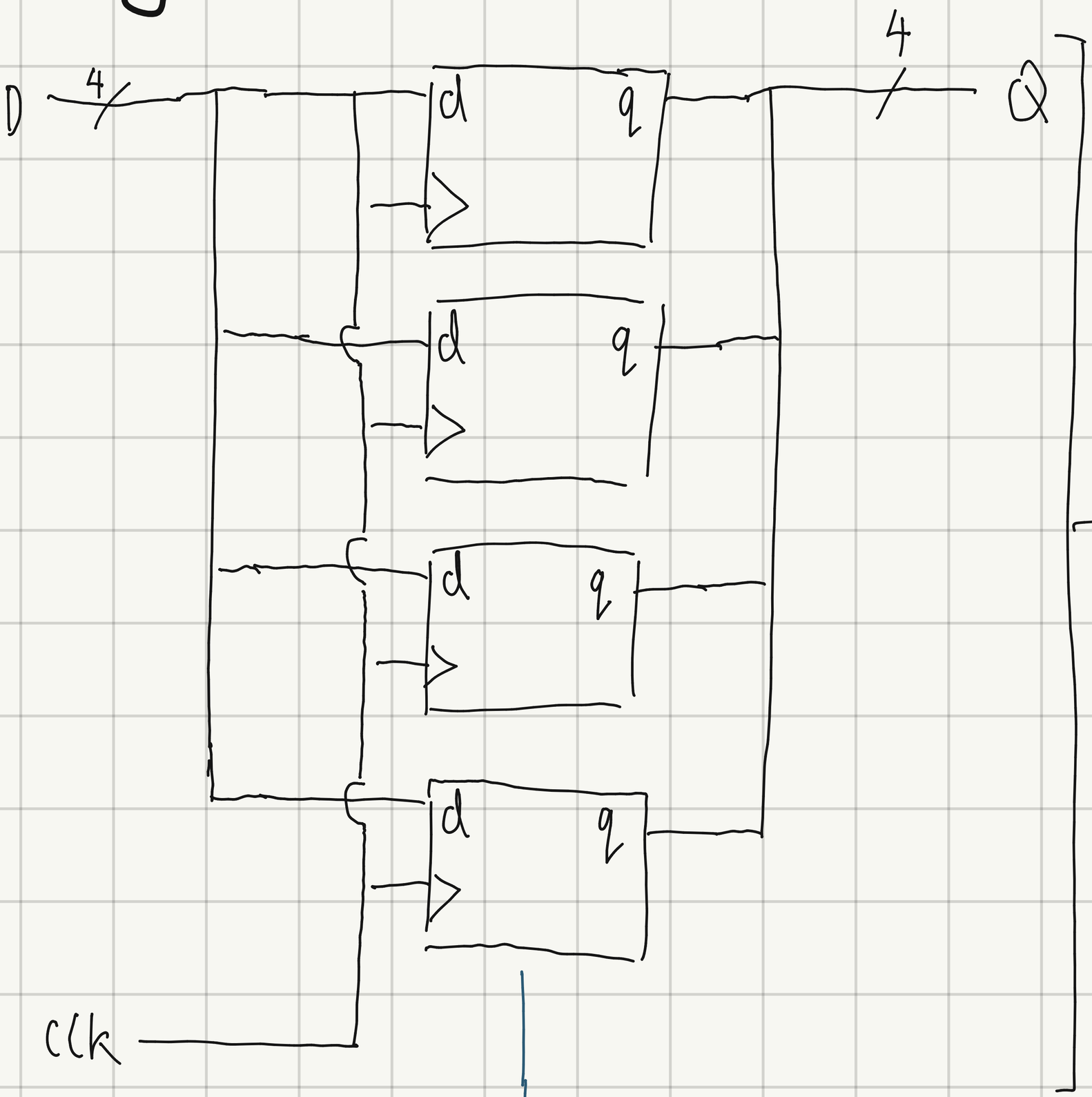
d	g	nd	s	r	action
0	0	1	0	0	hold
0	1	1	0	1	reset
1	0	0	0	0	hold
1	1	0	1	0	set

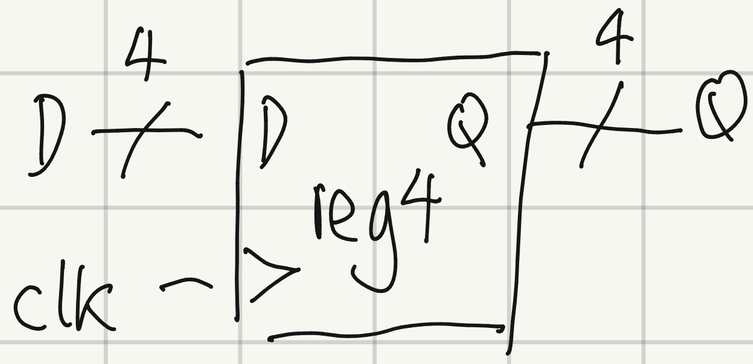
D-latch critical path = 8ns because SR-Latch critical path delay = 4ns

dffp (rising edge)



reg4



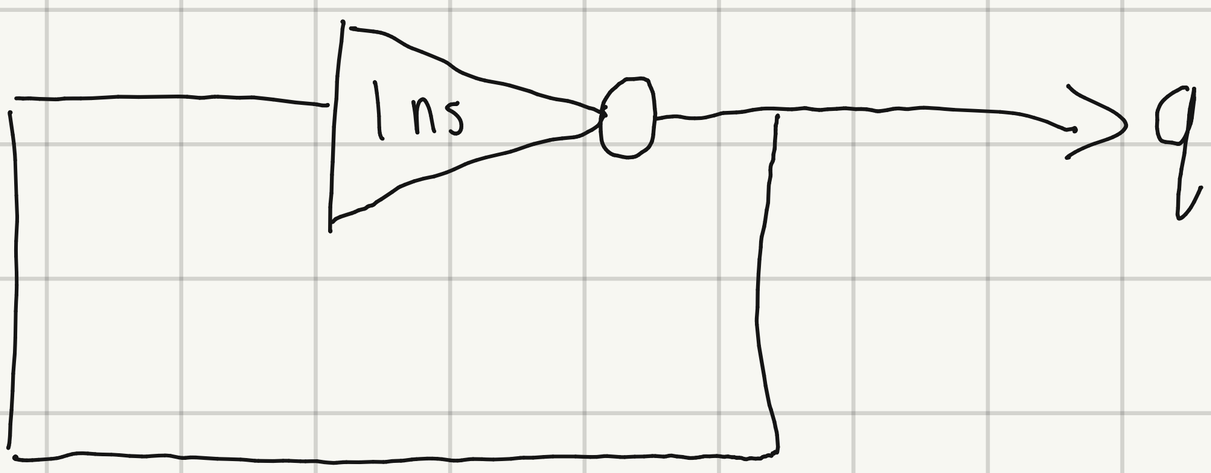


$t_{pLH}^{clk \to qout} = t_{pHL}^{clk \to qout} = 17\ ns$

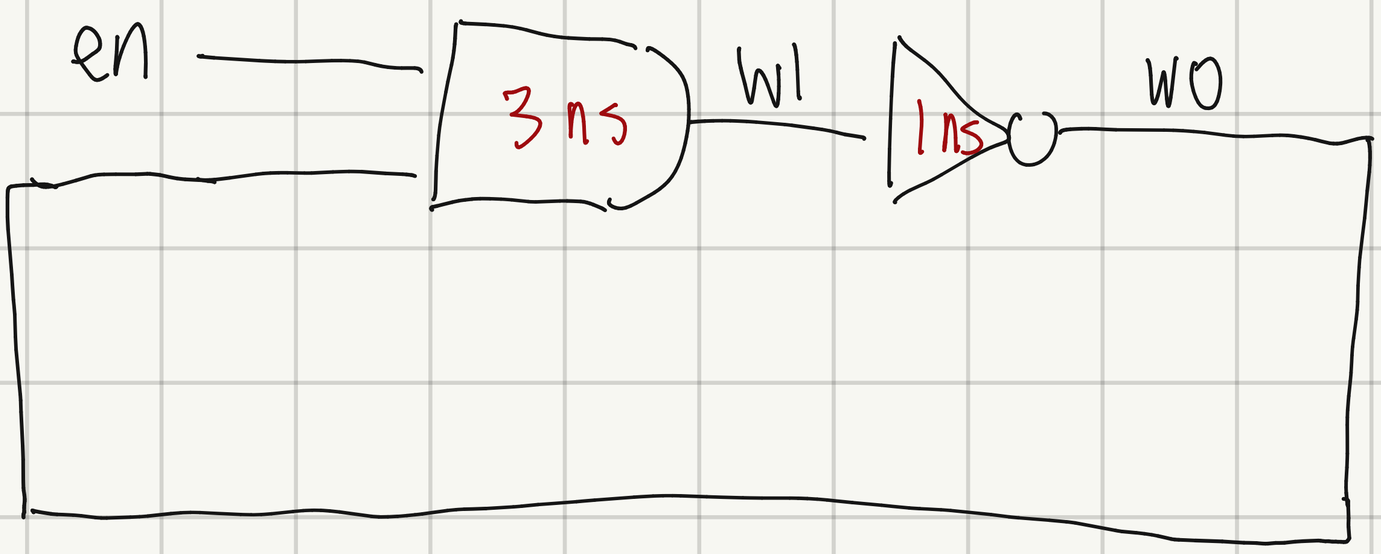
oscillator part 1

1-stage ring oscillator (works in theory)

```
input q;
output q;
```



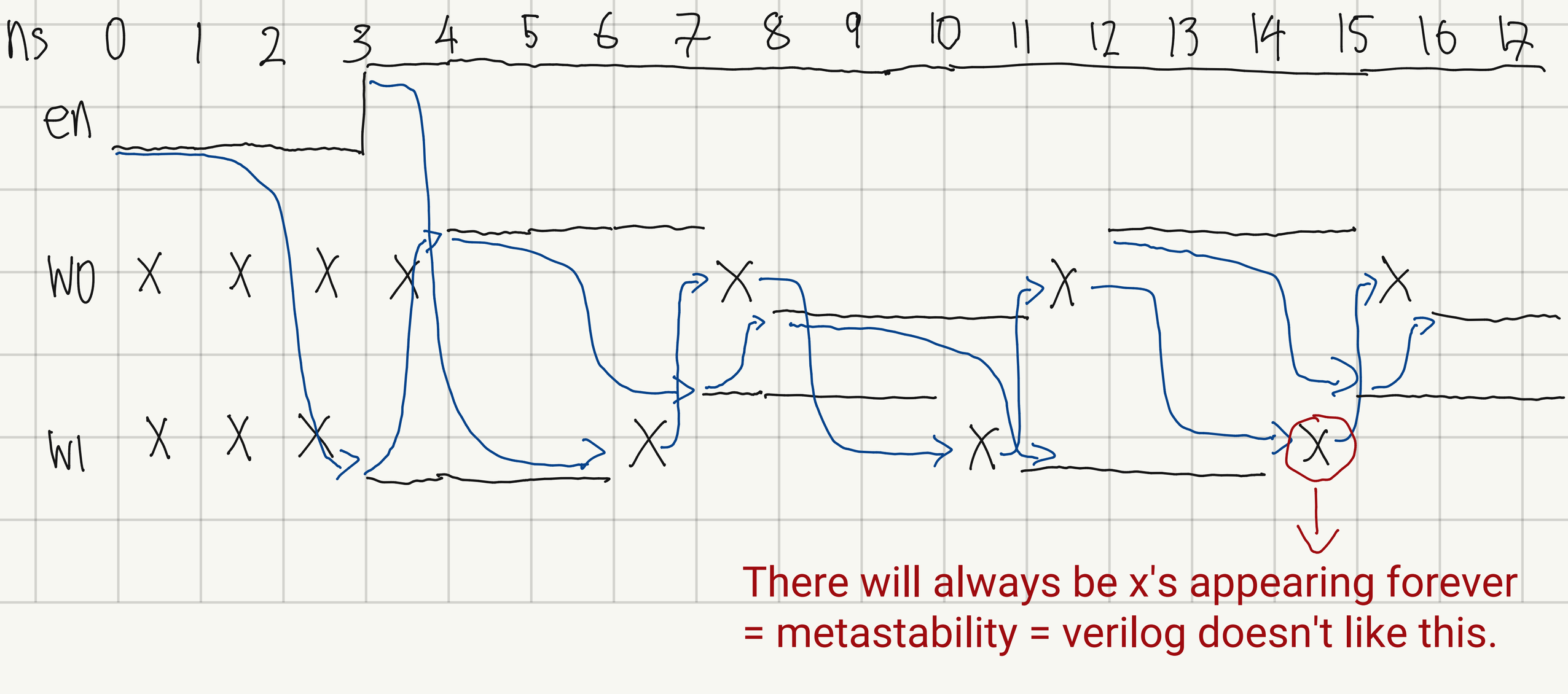
Ring oscillator w/ enable



en	wo	wl
0	0	0
0	1	0
1	0	0
1	1	1

setting en=0 makes wl=0 after 3ns, wo=1 after 4ns. At this point if en=1 then wl=1 and the oscillator works.

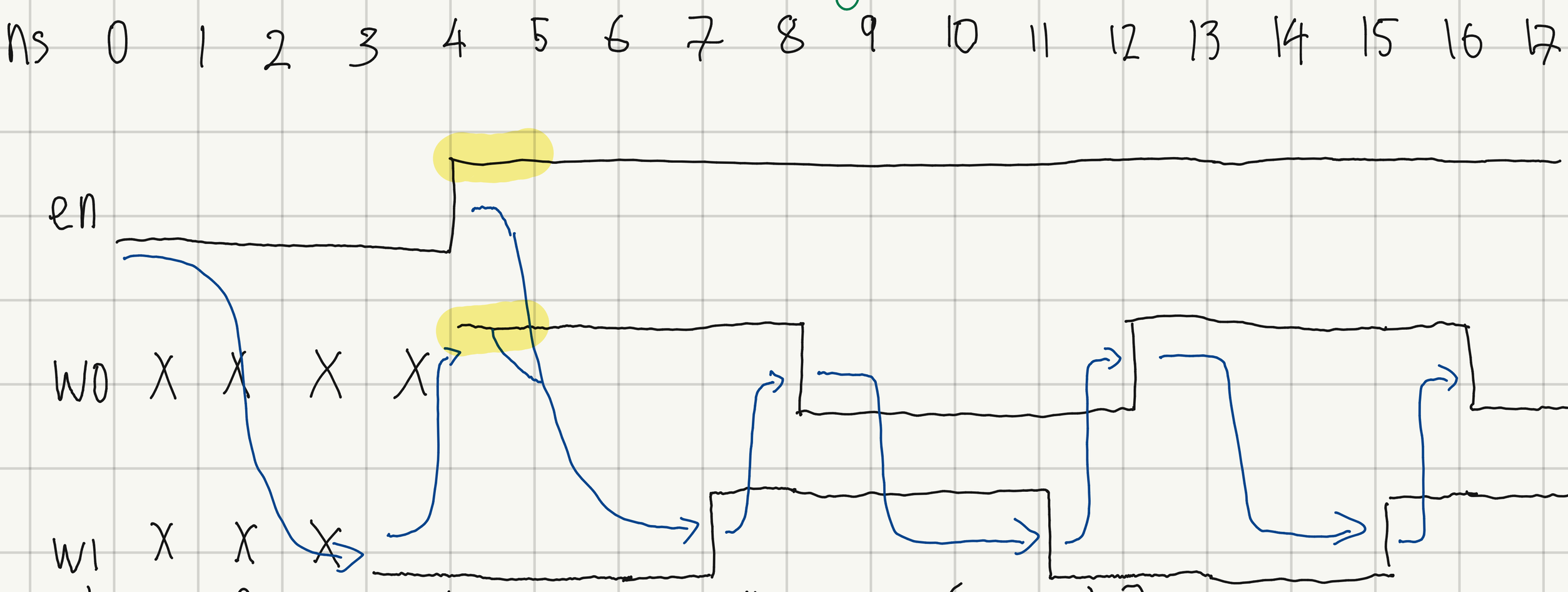
What if en=1 after 3ns? 不要



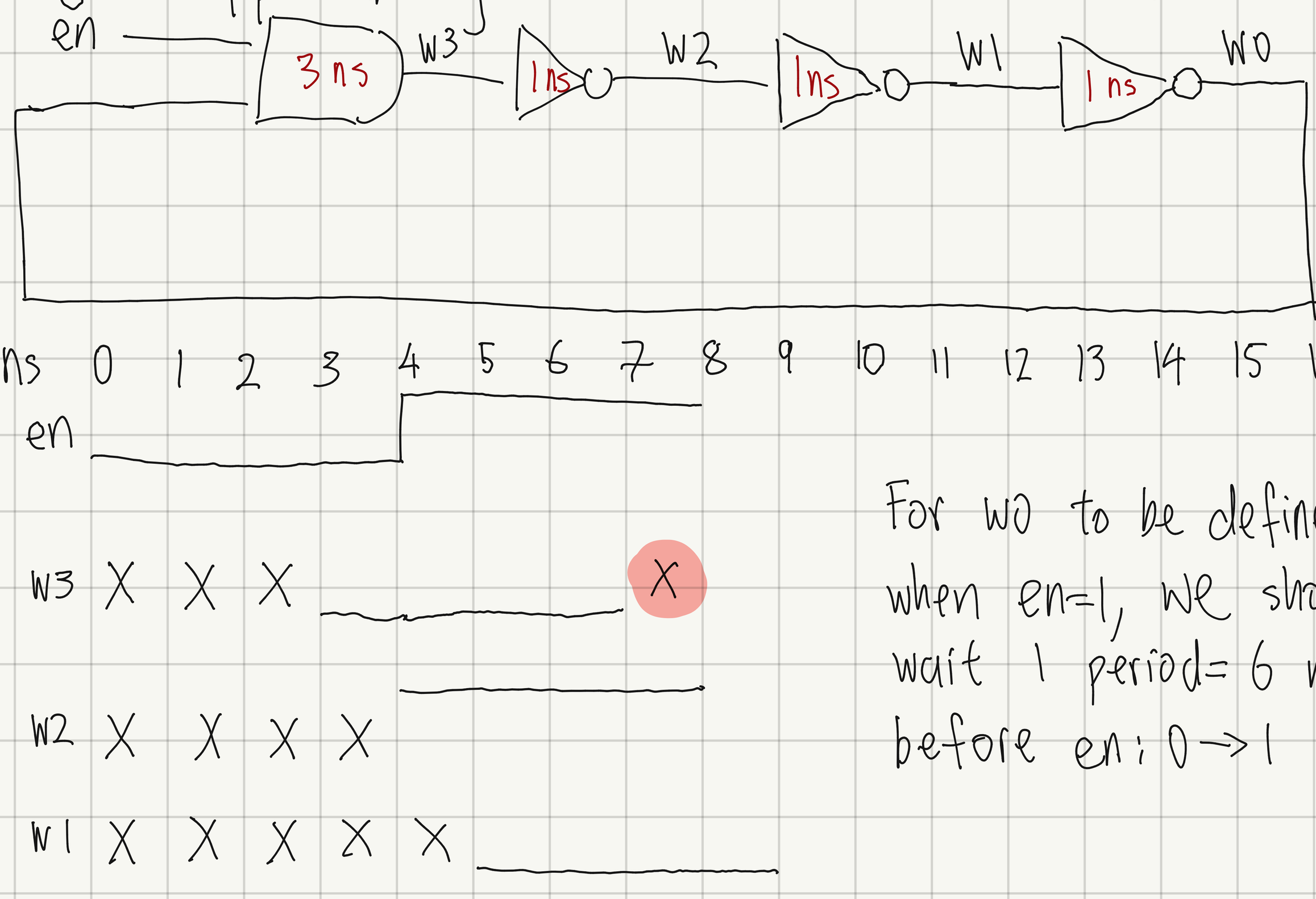
There will always be x's appearing forever
= metastability = verilog doesn't like this.

oscillator part 2

What if en=1 after 4ns? We get a 4ns oscillator!



What if we add more oscillators (n=3)? Will signals ripple through in a deterministic manner?



For w0 to be defined when en=1, we should wait 1 period = 6 ns before en: 0 → 1

oscillator part 3

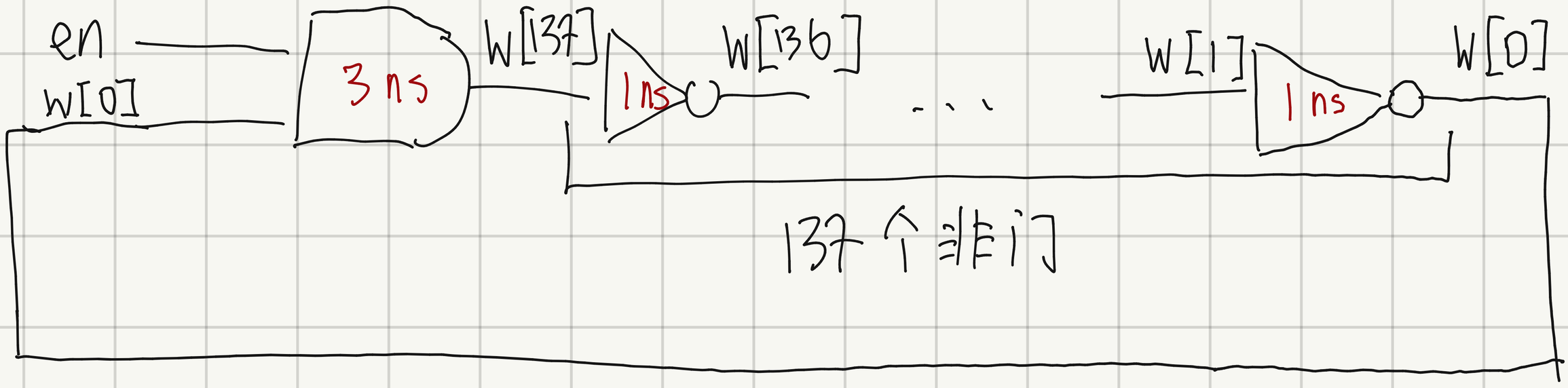
oscillator w/ period ≥ 2 * 139 ns, 实际周期 = 2 * 140 ns
= 280 ns

Let n = # not gates

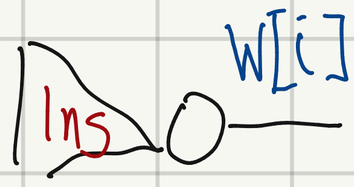
AND_DELAY + n * NOT_DELAY ≥ 139

3 + n * 1 ≥ 139

n ≥ 136, n%2=1, n=137

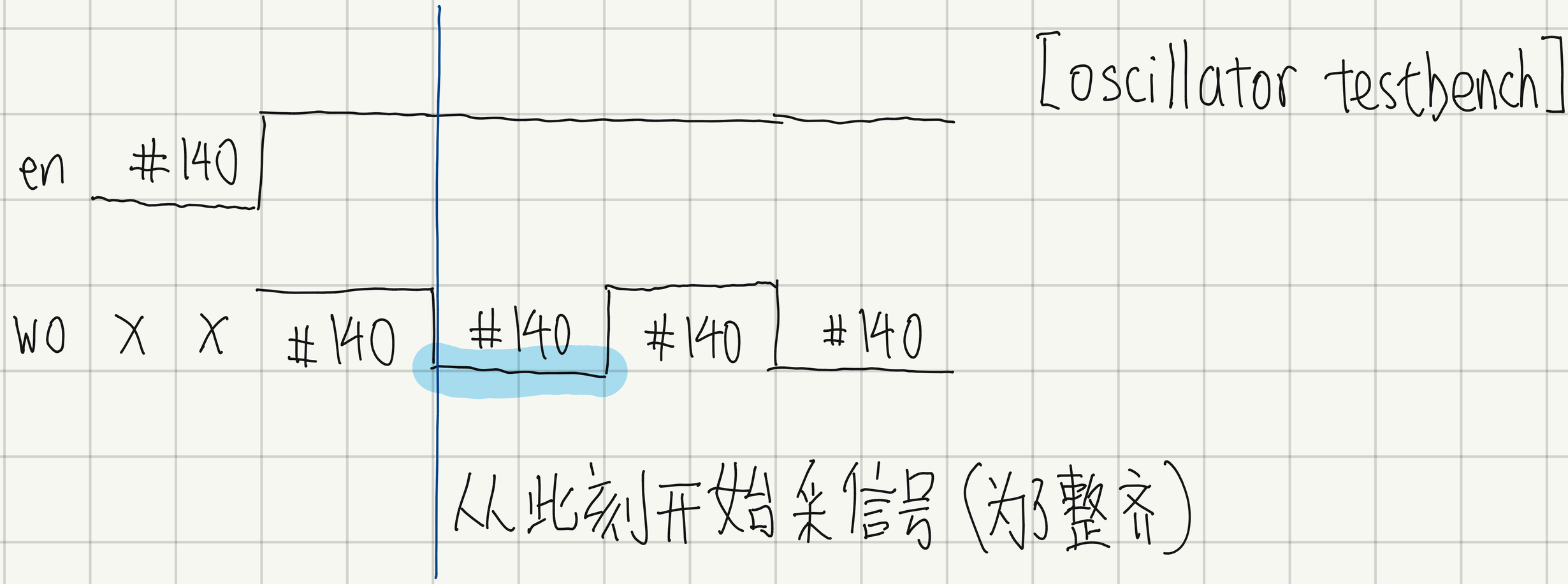


Verilog generate

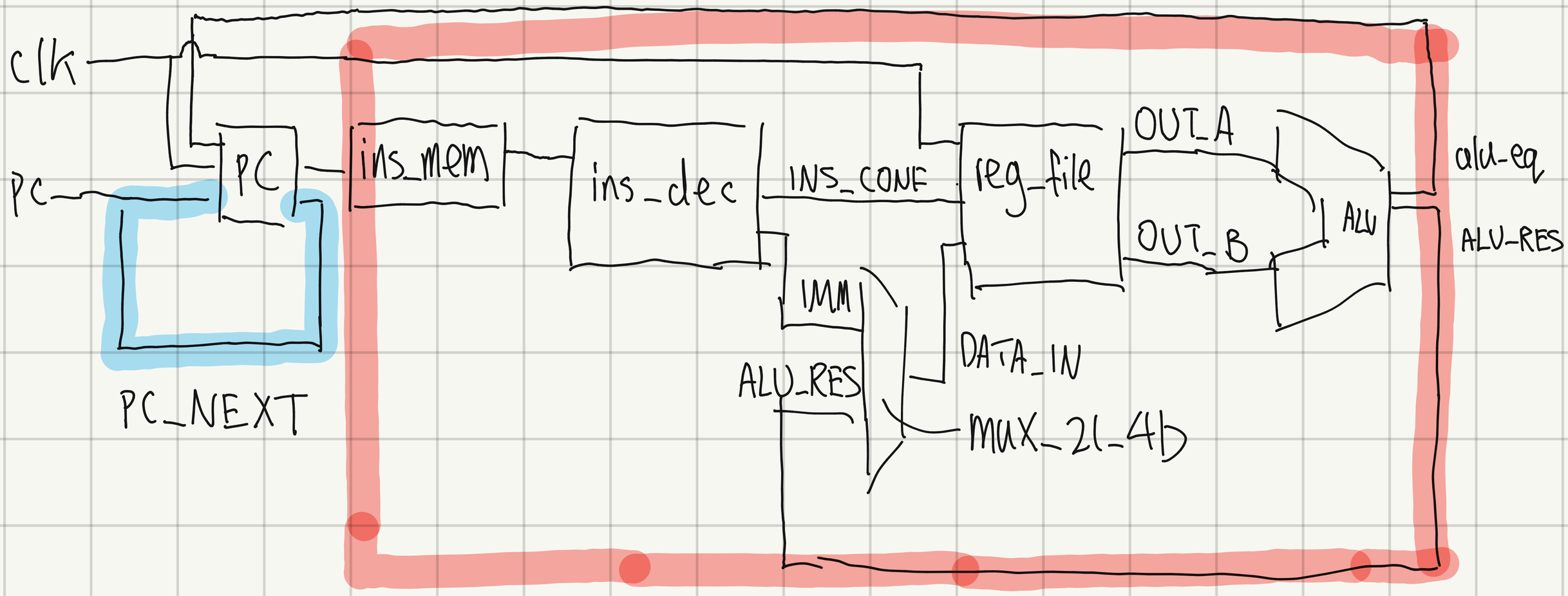


X 137 = w[136] ... w[0]

[oscillator testbench]



Datapath delay



PC init
PC=0

-> clk=0

为什么clk=0的时候算ALU? 因为ALU result来自ins中的信息, 在clk 0->1时会更新: reg=ALU result

ins_mem (28 ns) +
ins_dec (4ns) +
read_reg (14 ns) +
max (
  alu_eq (8 ns) +
  PC_setup (75 ns),
  ALU (51 ns) +
 mux_2_1_4b (7ns) +
 write_reg_setup (35ns) +
)
= 139 ns

83 ns
93 ns

clk=1
max (PC hold (17ns),
  write_reg_hold (17ns))
= 17 ns

∴ need clock period ≥ 2(139 ns) = 278 ns