COMBINATIONAL/SEQUENTIAL LOGIC PART: FLIP-FLOPS, BUSSES, ENABLERS, DECODERS, ACCESSING THE MEMORY, CIRCUIT DELAY & THE CLOCK PART I

Ayman Hajja, PhD

SYNCHRONOUS DIGITAL SYSTEMS

- Synchronous: all operations coordinated by a central clock
  - Heartbeat of the system
- Digital:
  - Everything is represented by discrete values (1 or 0)

TYPES OF CIRCUITS

- Synchronous Digital Systems consist of two basic types of circuits:
  - Combinational Logic (CL) circuits:
    - Output is a function of the inputs only, not the history of its execution
    - E.g., circuits to add A, B (ALUs)
  - Sequential Logic (SL):
    - Circuits that "remember" or store information — aka "State Elements"
    - E.g. memories and registers (Registers)

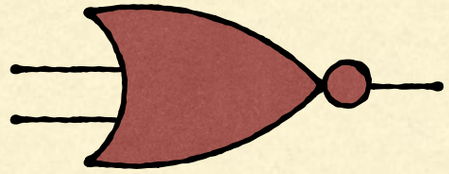
FUNCTIONAL COMPLETENESS

- A set of boolean functions is functionally complete, if all other boolean functions can be constructed from this set.
- The set of functions {AND, OR, NOT} are functionally complete
- Can we reconstruct 'OR' from 'AND' & 'NOT'? In other words, can we find an expression that is logically equivalent to 'A OR B' using only 'AND's & 'NOT's?

Yes we can. 'A OR B' is logically equivalent to ~(~A . ~B)

MORE COMBINATIONAL LOGIC GATES; NOR

Boolean operator that gives the value one if and only if all operands have a value of zero and otherwise has a value of zero.

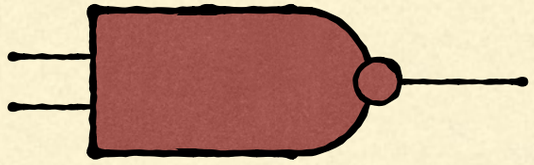


NOR Truth Table

A	B	A NOR B
1	1	0
1	0	0
0	1	0
0	0	1

MORE COMBINATIONAL LOGIC GATES; NAND

Boolean operator that gives the value zero if and only if all the operands have a value of one, and otherwise has a value of one.

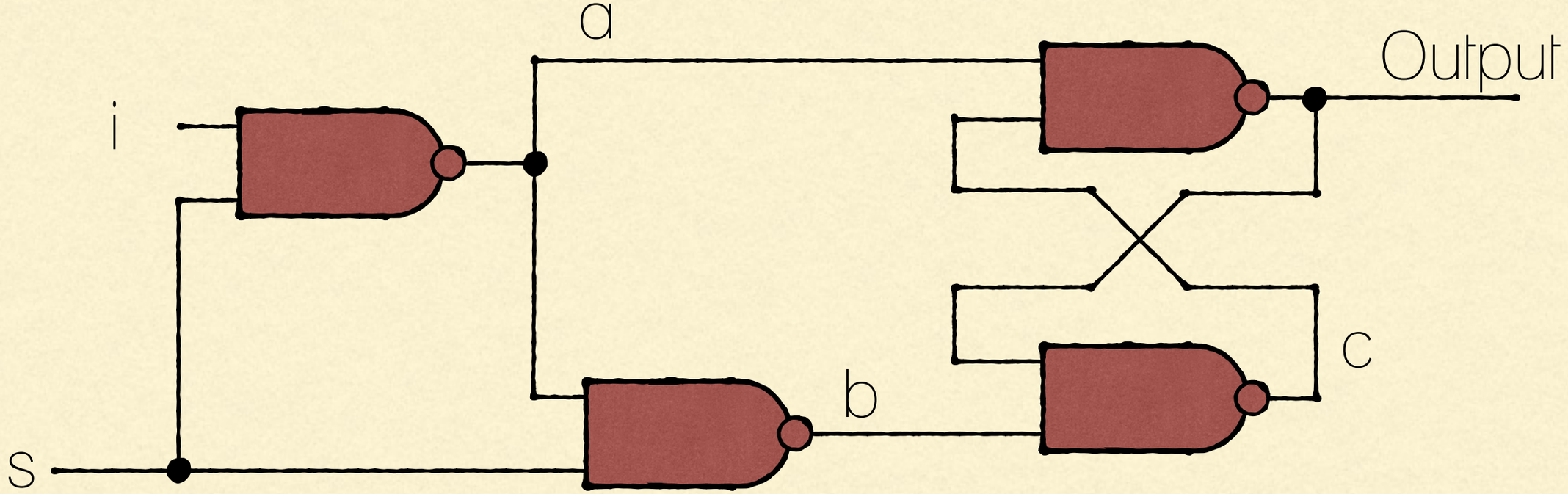


NAND Truth Table

A	B	A NAND B
1	1	0
1	0	1
0	1	1
0	0	1

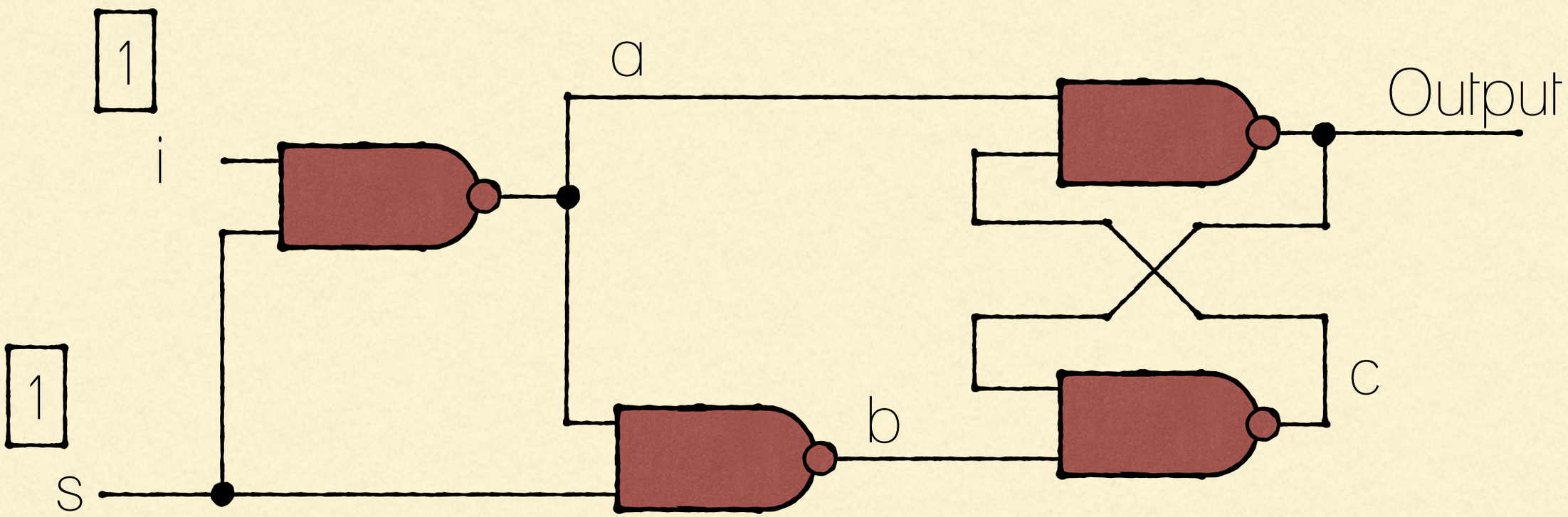
CONSTRUCTING A BIT

- The following diagram shows one bit of computer memory. It happens to be one of the neatest tricks you can do with few gates.



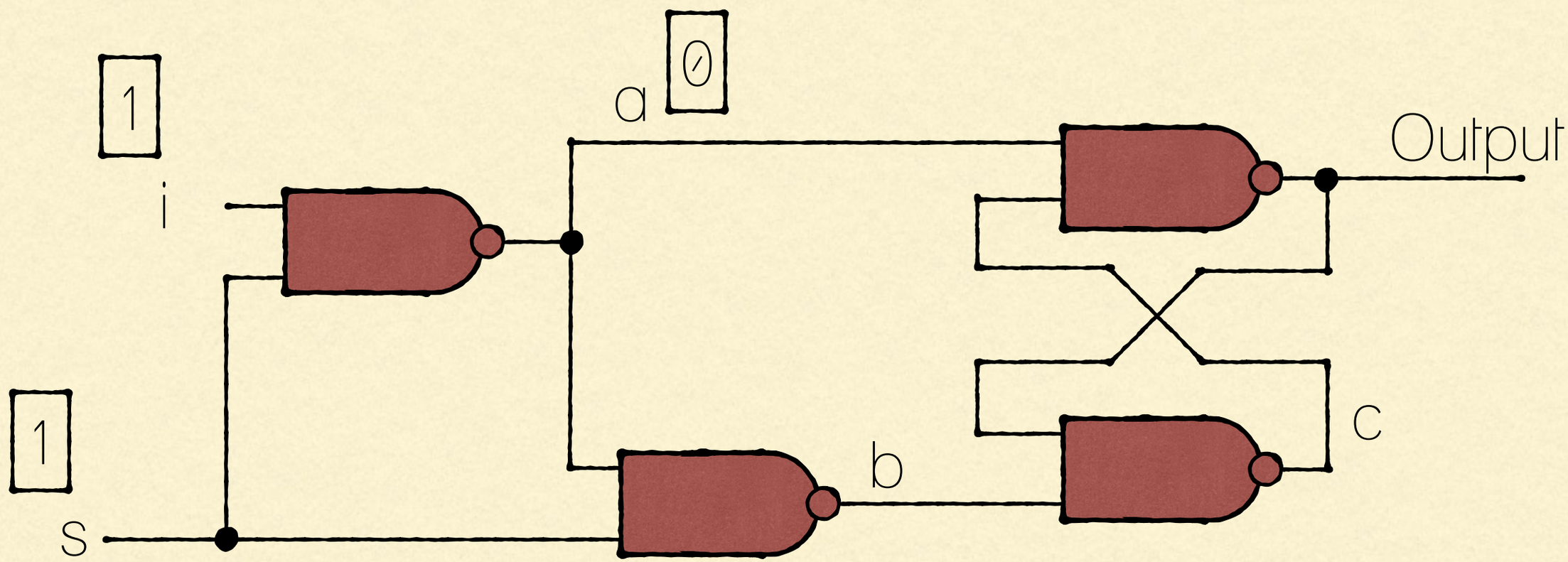
CONSTRUCTING A BIT

- Let's set i to 1, and s to 1. What's the output?



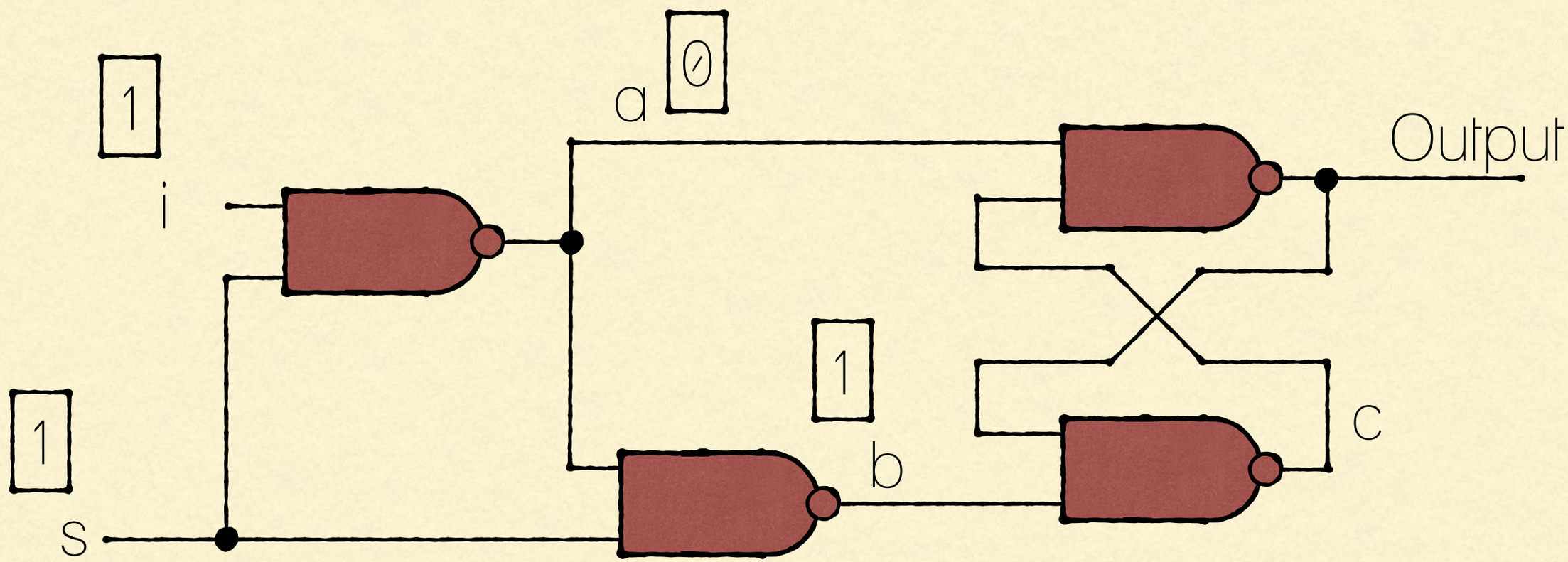
CONSTRUCTING A BIT

- Let's set i to 1, and s to 1. What's the output?



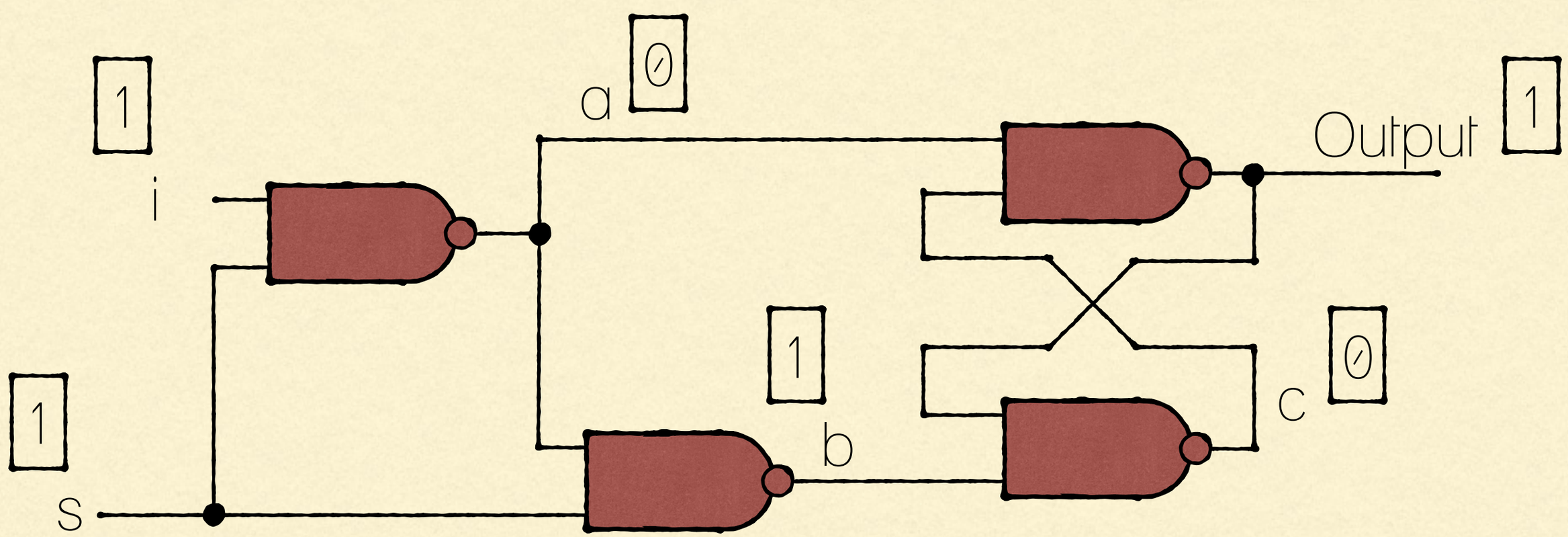
CONSTRUCTING A BIT

- Let's set i to 1, and s to 1. What's the output?



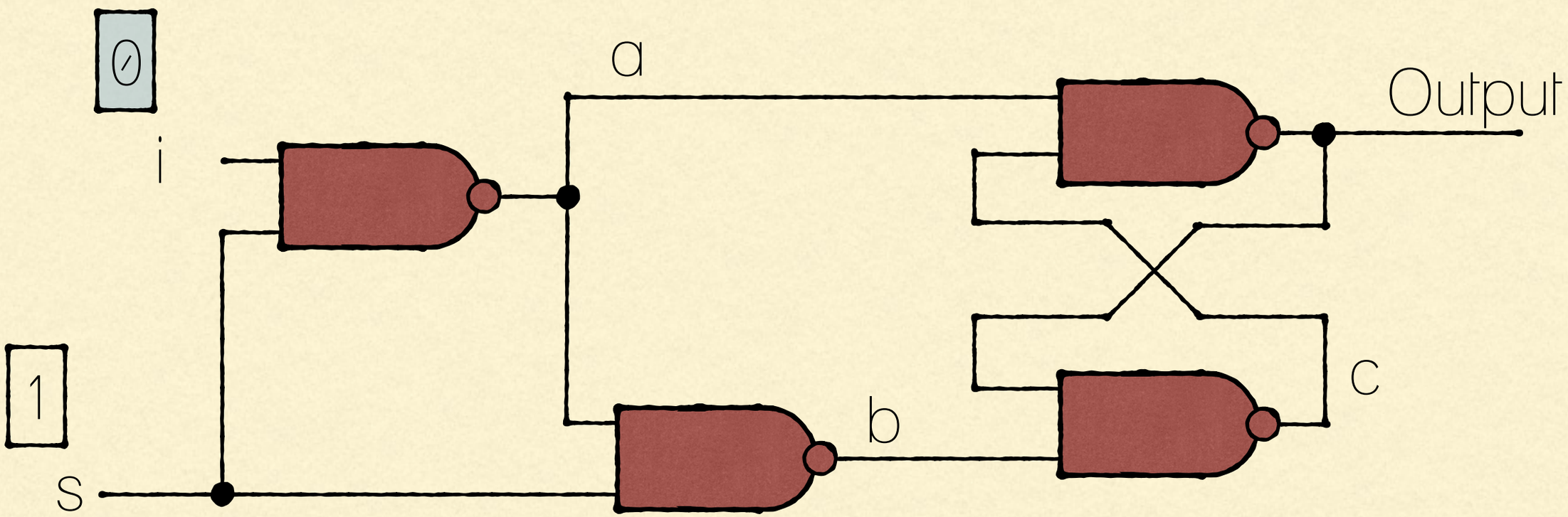
CONSTRUCTING A BIT

- Let's set i to 1, and s to 1. What's the output? The answer is 1 — Now let's change i to 0 and see what happens



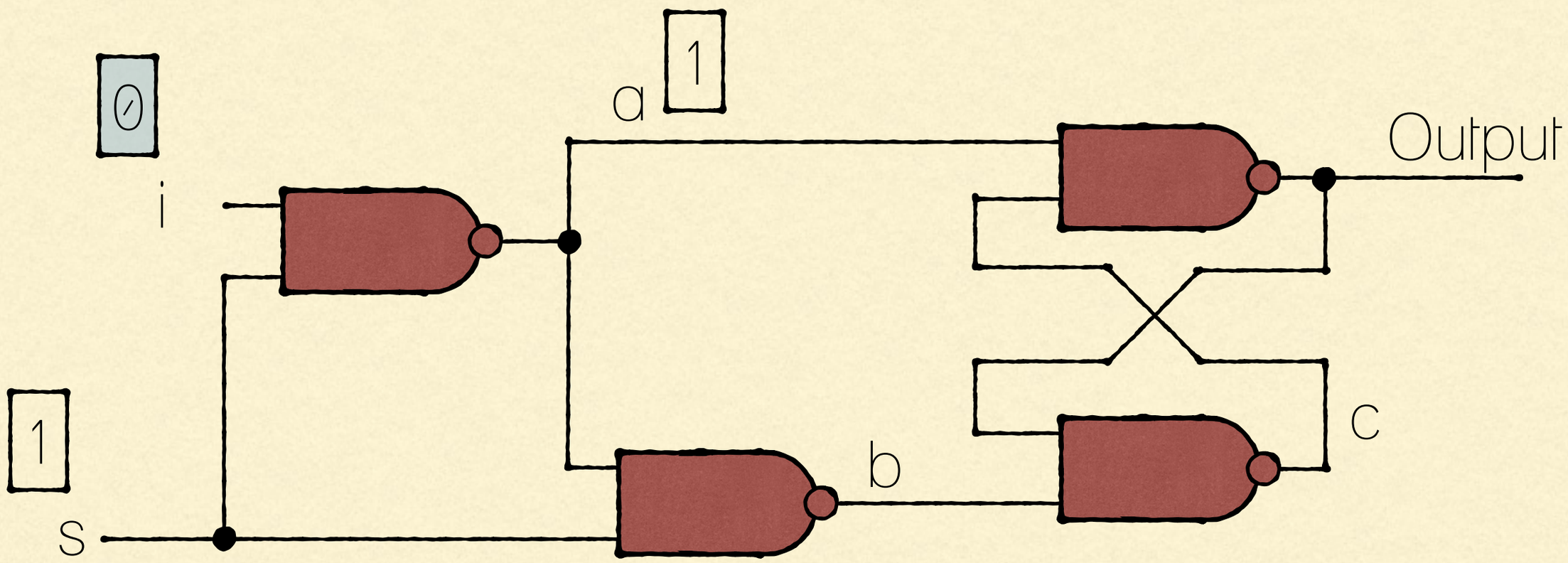
CONSTRUCTING A BIT

- Now let's change i to 0 and see what happens



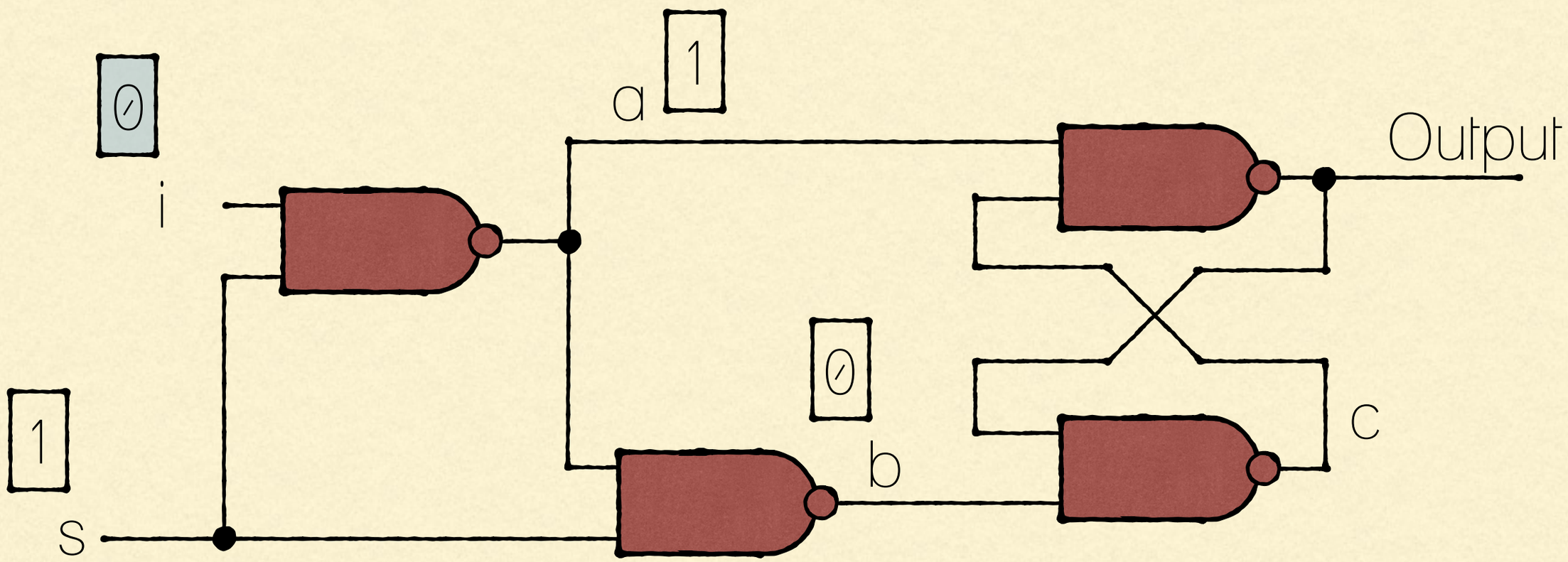
CONSTRUCTING A BIT

- Now let's change i to 0 and see what happens



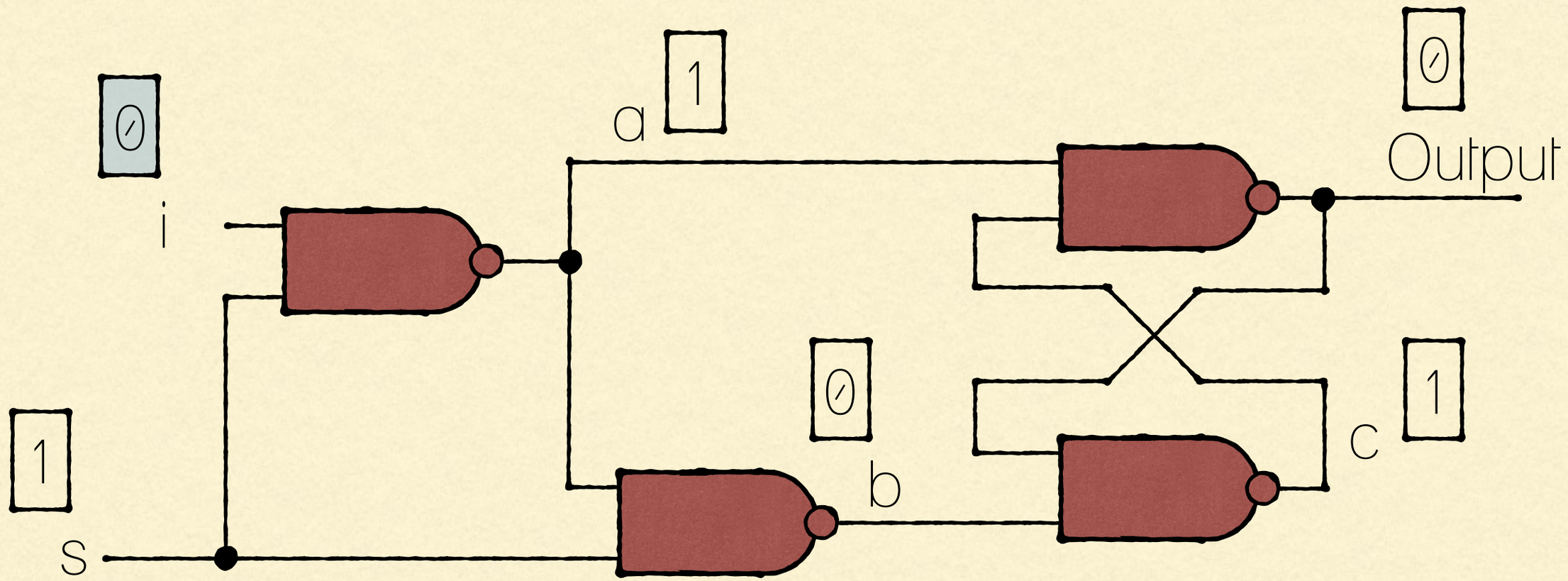
CONSTRUCTING A BIT

- Now let's change i to 0 and see what happens



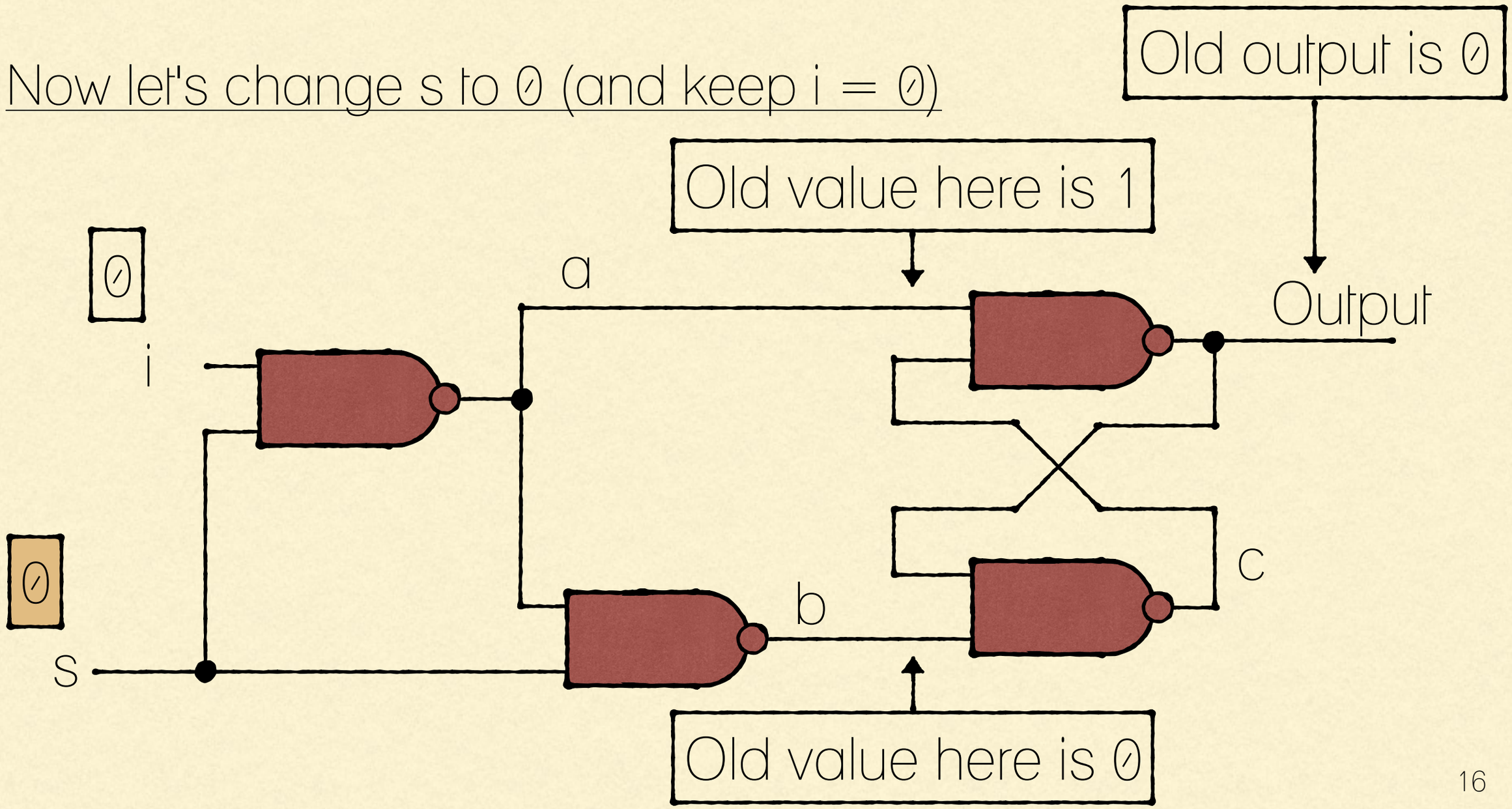
CONSTRUCTING A BIT

- Now let's change i to 0 and see what happens — Output becomes 0



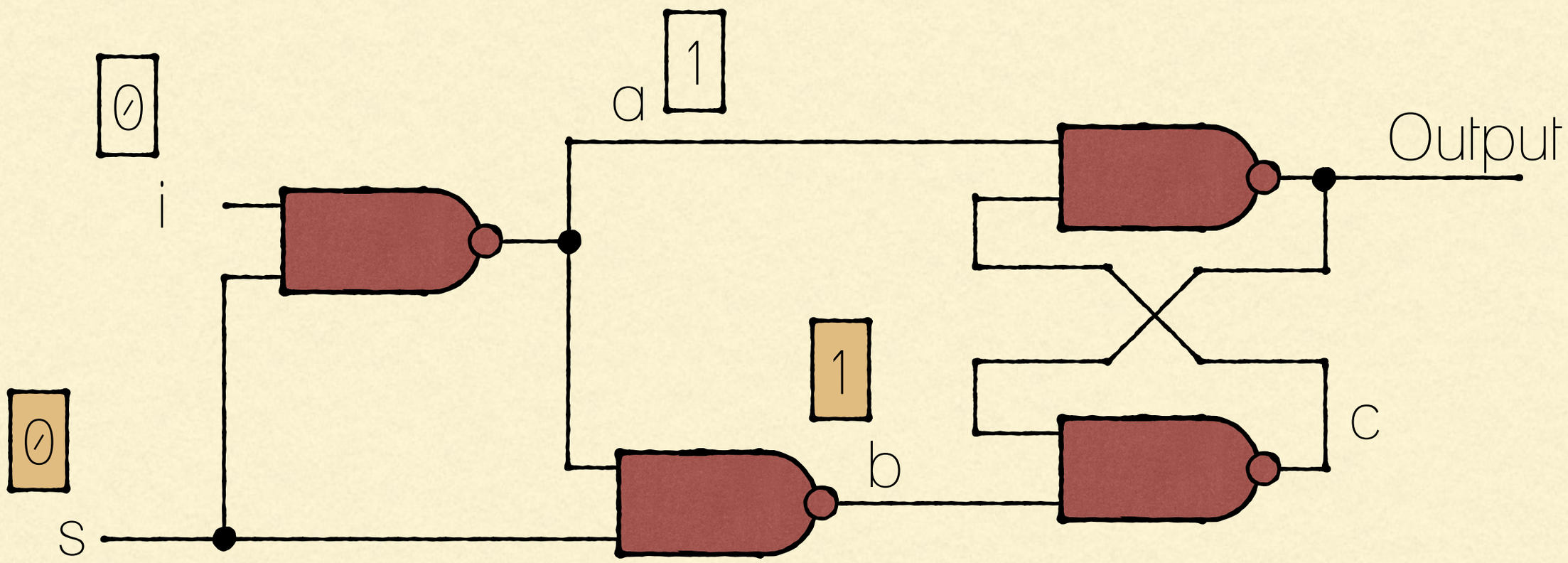
CONSTRUCTING A BIT

- Now let's change s to 0 (and keep i = 0)



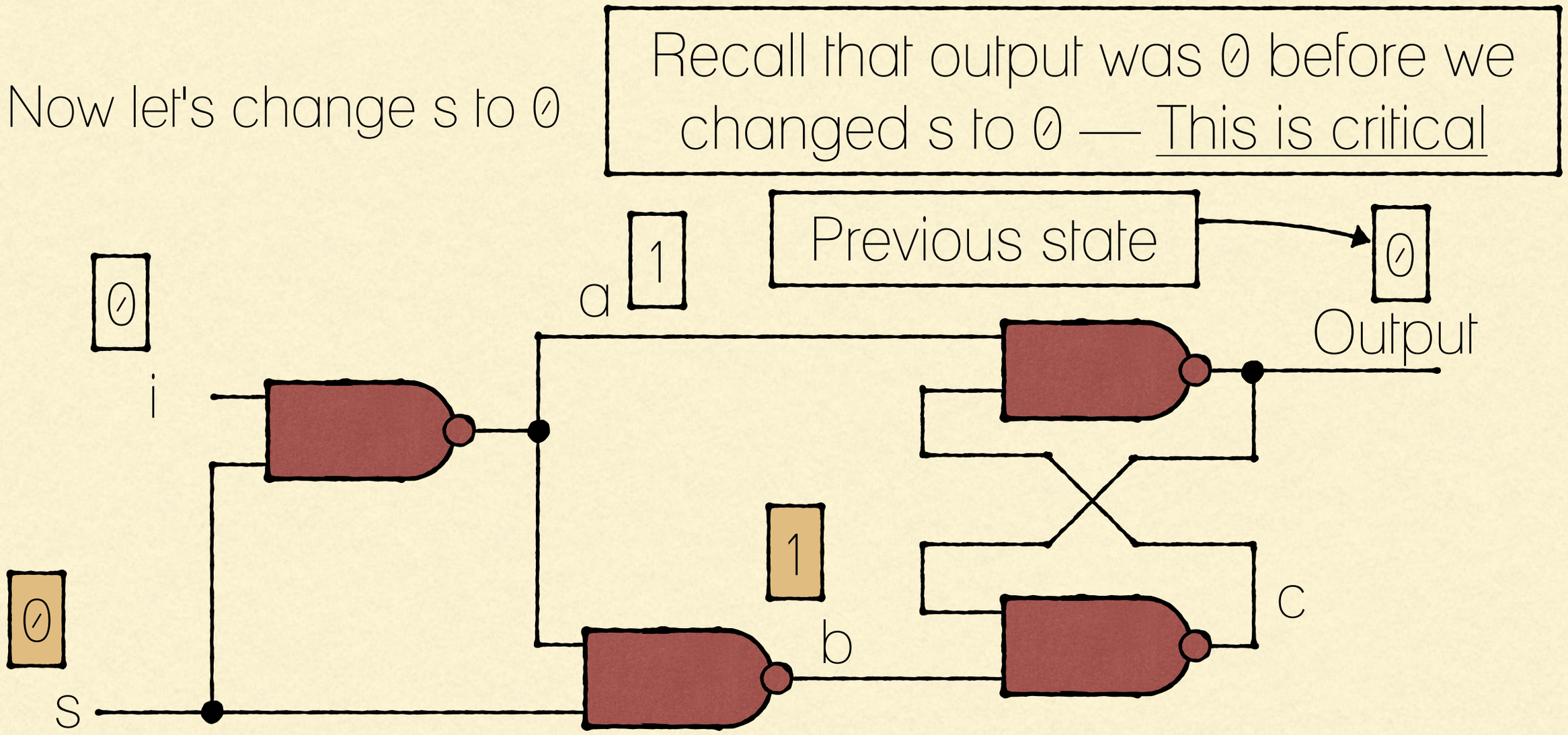
CONSTRUCTING A BIT

- Now let's change s to 0 (and keep i = 0)



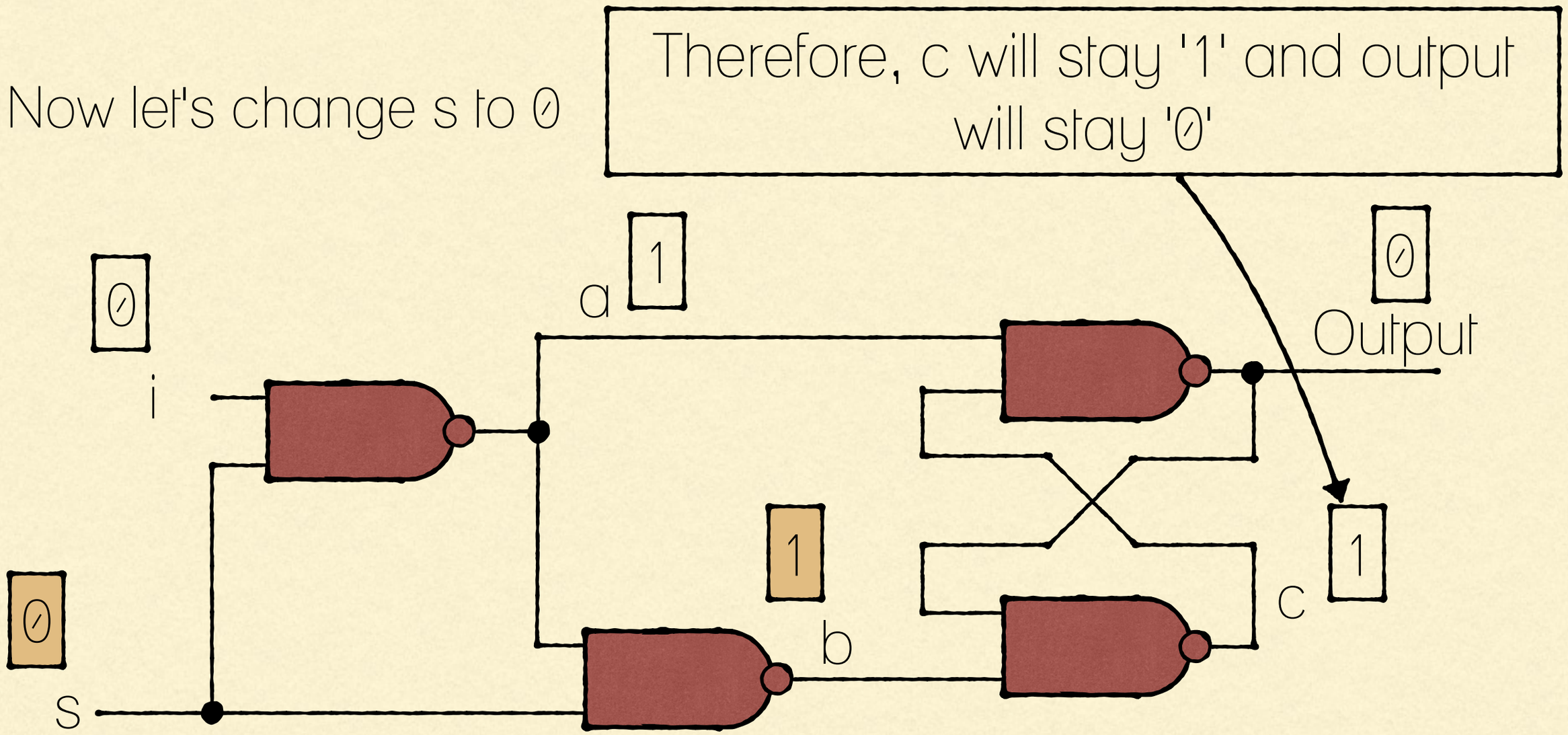
CONSTRUCTING A BIT

- Now let's change s to 0



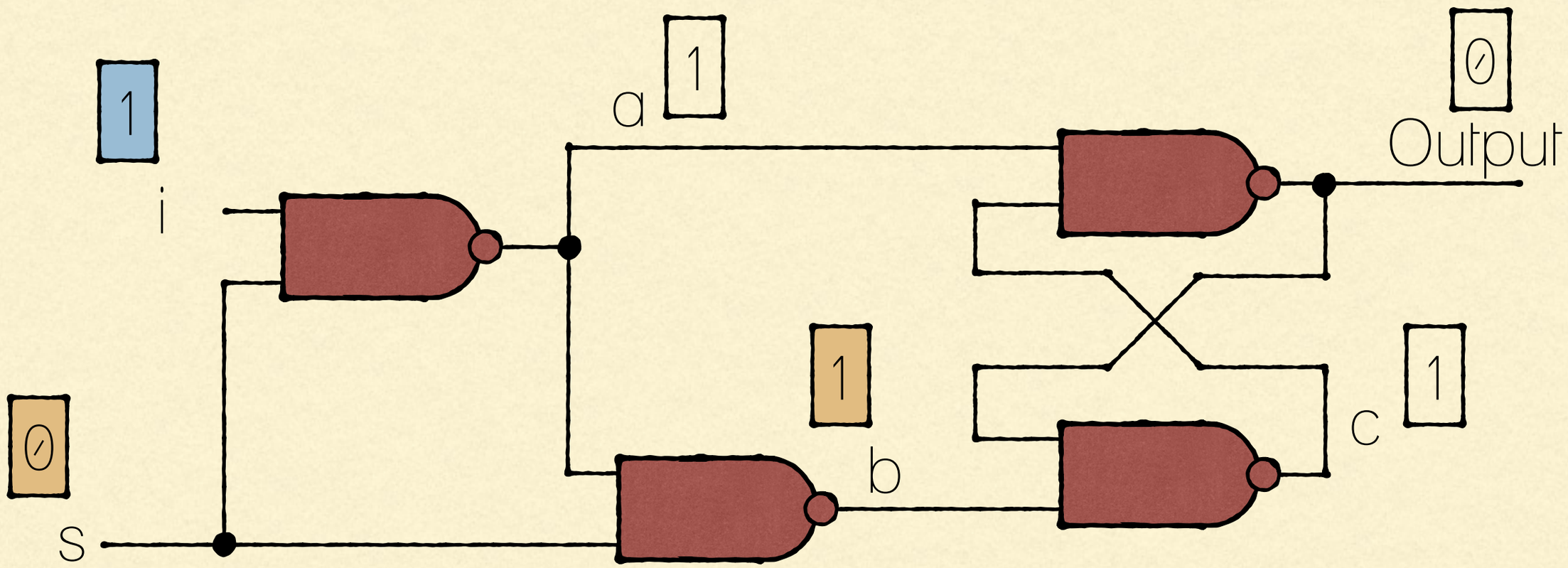
CONSTRUCTING A BIT

- Now let's change s to 0



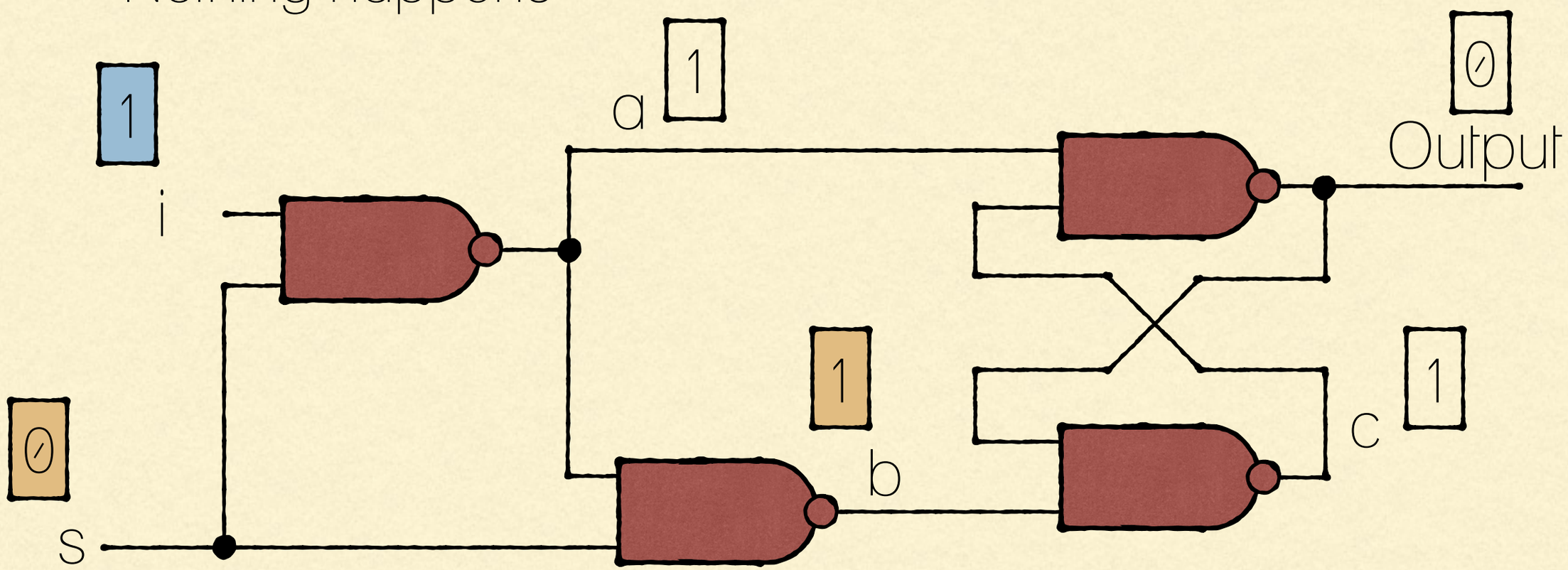
CONSTRUCTING A BIT

- Now let's keep s equal to 0 and change i to 1 — What happens?



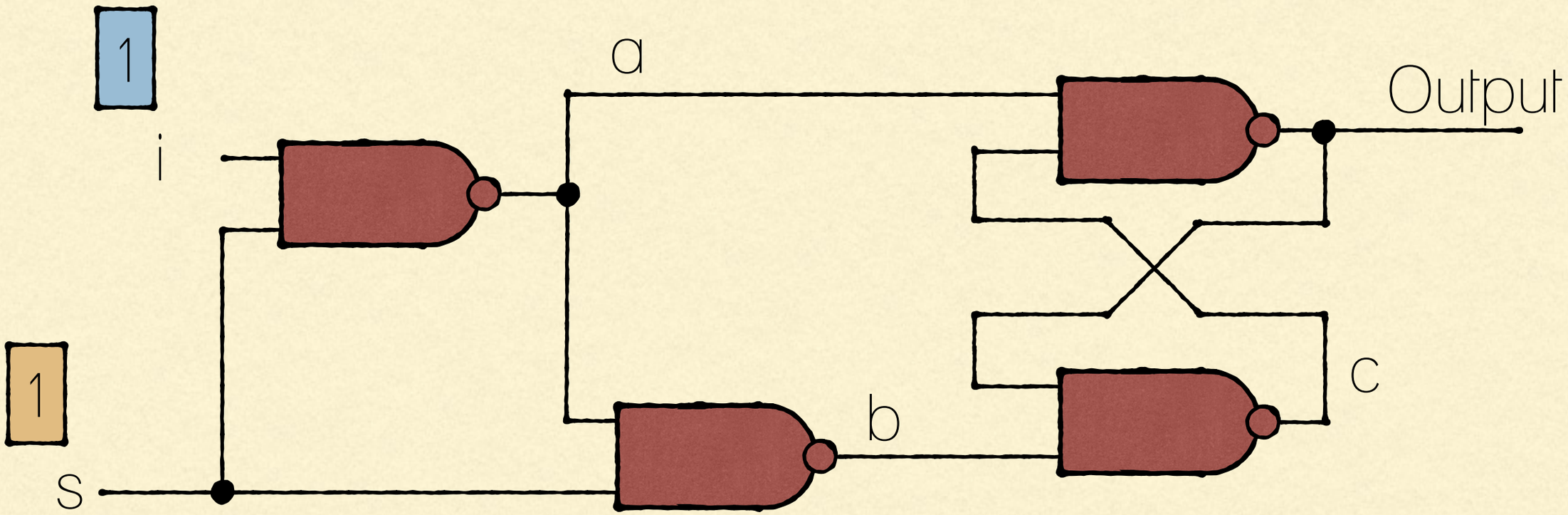
CONSTRUCTING A BIT

- Now let's keep s equal to 0 and change i to 1 — What happens?
  - Nothing happens



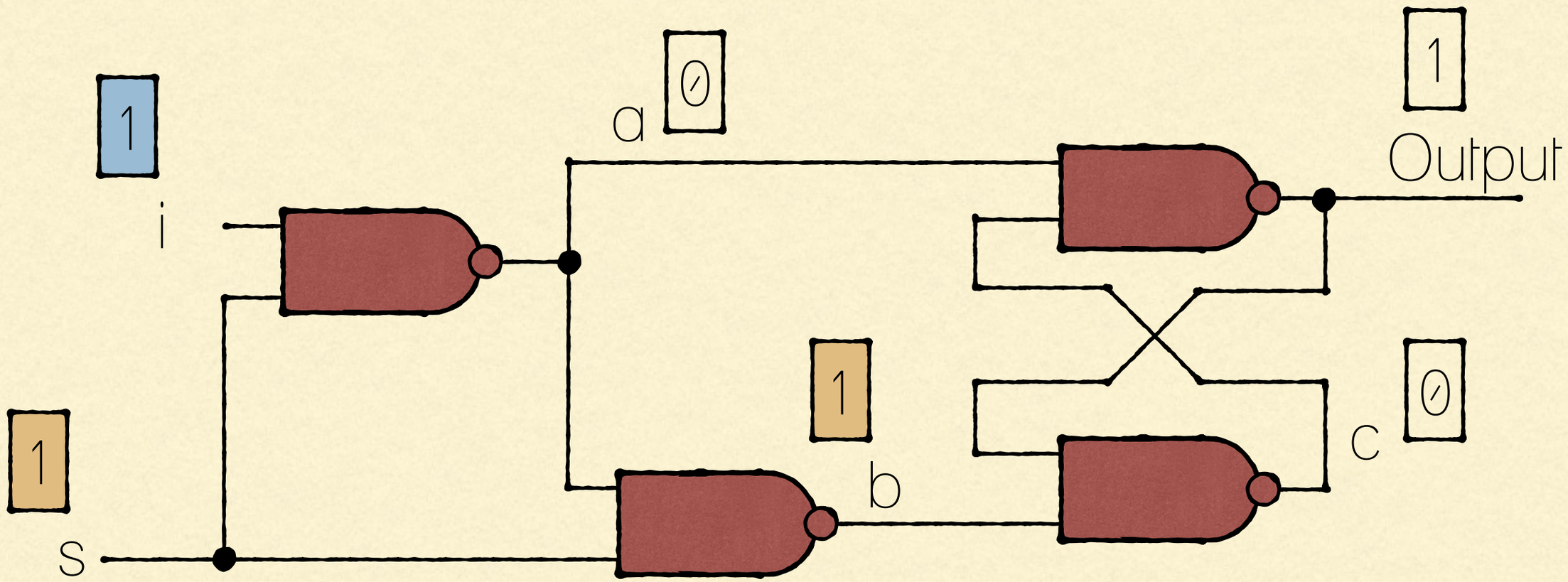
CONSTRUCTING A BIT

- Now let's change s back to 1, while keeping i equals to 1. What happens?



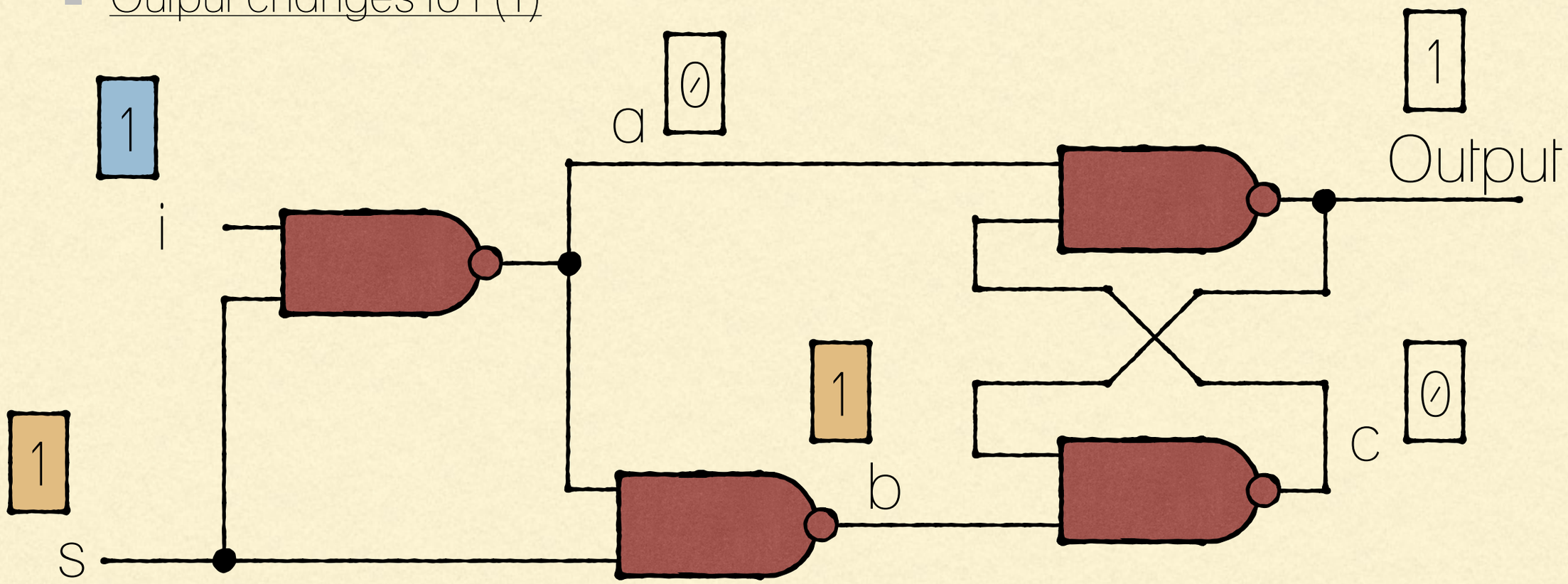
CONSTRUCTING A BIT

- Now let's change s back to 1, while keeping i equals to 1. What happens?



CONSTRUCTING A BIT

- Now let's change s back to 1, while keeping i equals to 1.
  - Output changes to i (1)

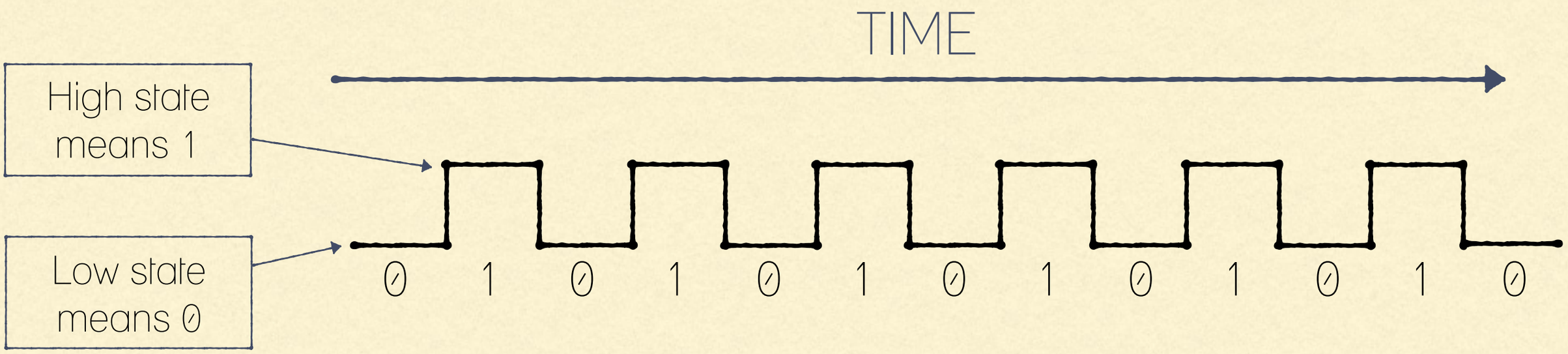


CONSTRUCTING A BIT

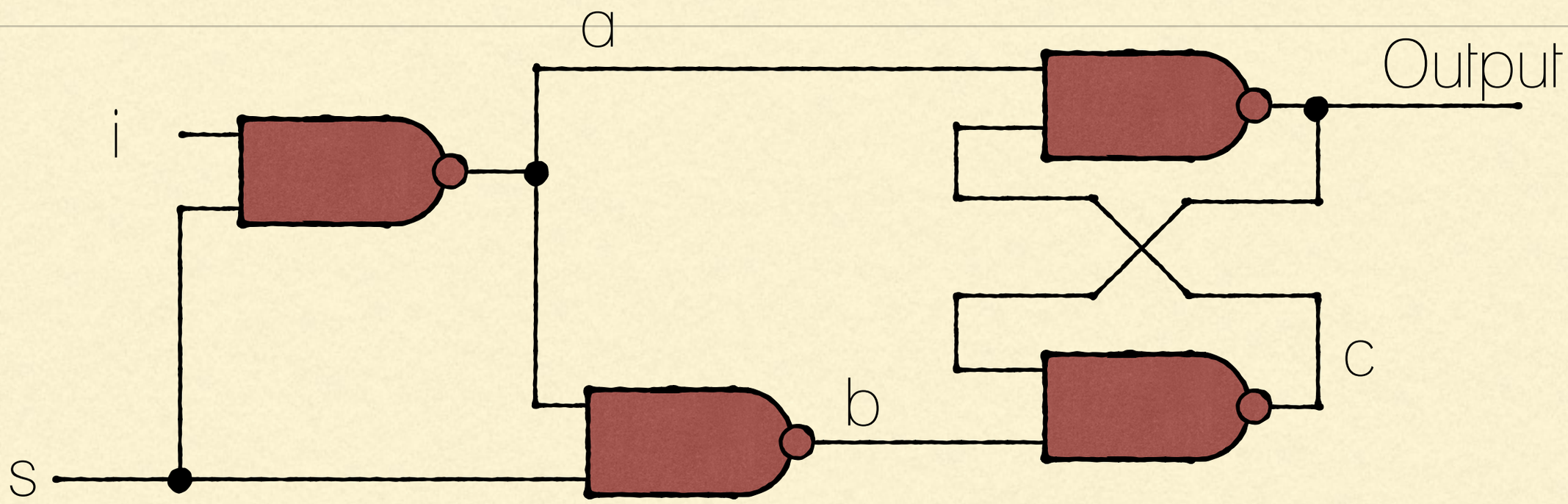
If we need to change the output, we pass the value we'd like to change it through 'i', and change the value of 's' to 1. Then we can lock the bit/value in place by changing 's' to 0

UNDERSTANDING LOGIC WAVEFORMS/SIGNALS

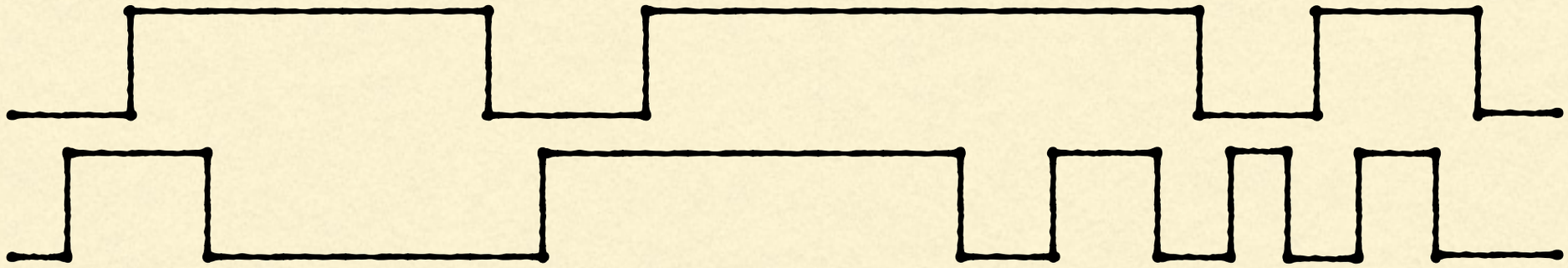
A logic signal is a digital signal with only two possible values;
'1'; represented by a high state as shown below
'0', represented by a low state
The signal below is an example of a logic waveform/signal



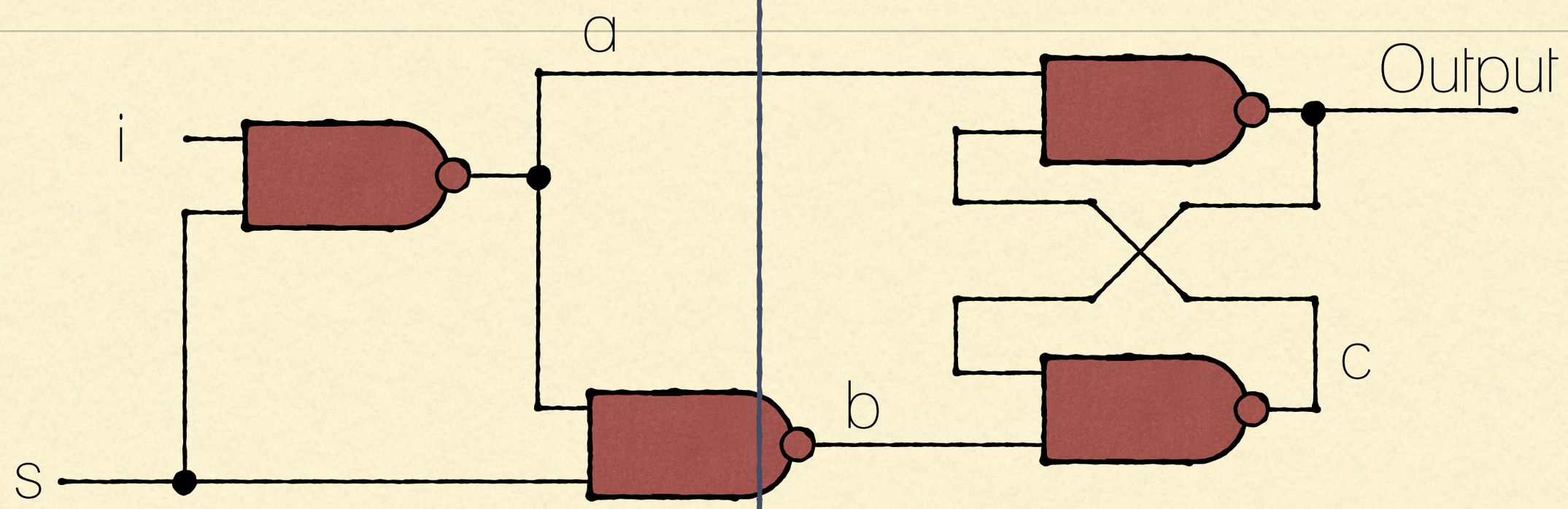
Recall that for the flip-flop below, the value of 'output' will be equal to 'input' only when 's' is equal to 1 (high state). When the value of 's' is equal to 0 (low state), then the value of output will be locked (and won't change)



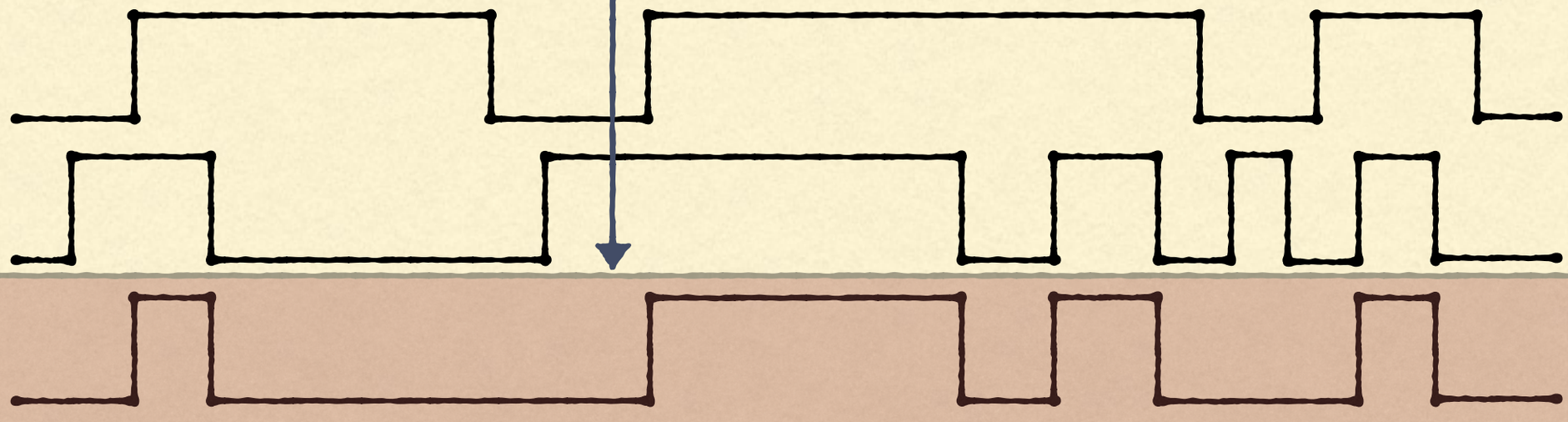
- Given the waveforms for 's' and 'i', draw the resulting waveform for the output 'q':
- s
- i (input)
- q (output)



SOLUTION; DETAILED EXPLANATION
PRESENTED ON NEXT SLIDES



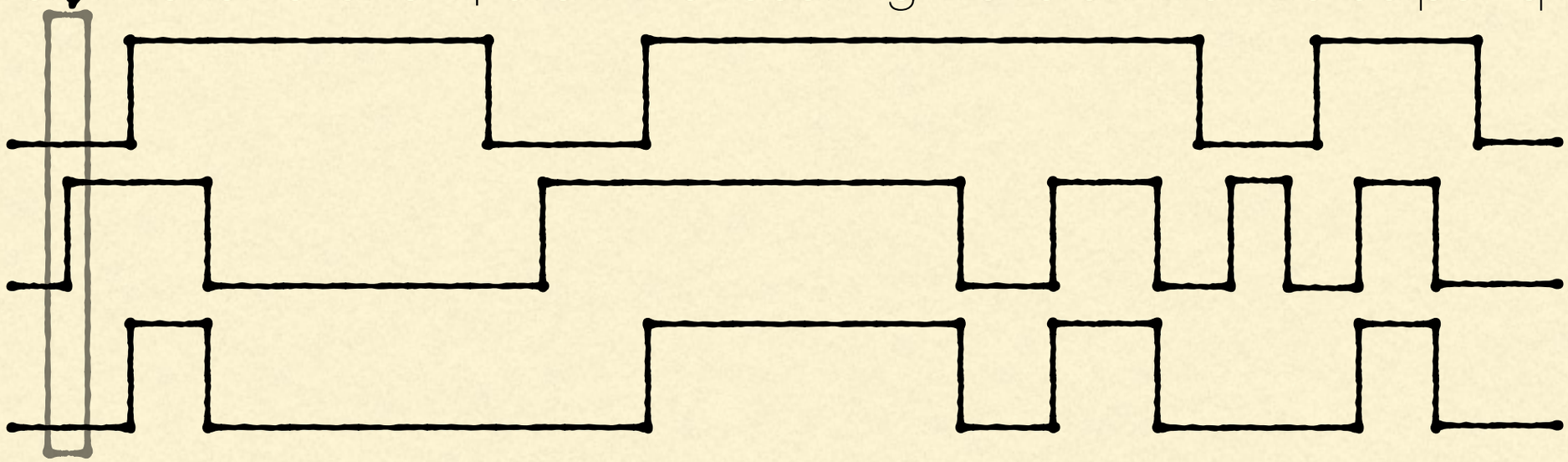
- Given the waveforms for 's' and 'i', draw the resulting waveform for the output 'q':
- s
- i (input)
- q (output)



Recall that for the flip-flop below, the value of 'output' will be equal to 'input' only when 's' is equal to 1 (high state). When the value of 's' is equal to 0 (low state), then the value of output will be locked (and won't change)

At this point of time, output is not equal to input because 's' is equal to 0, meaning that the state of the output 'q' is locked and won't change when input 'i' changes

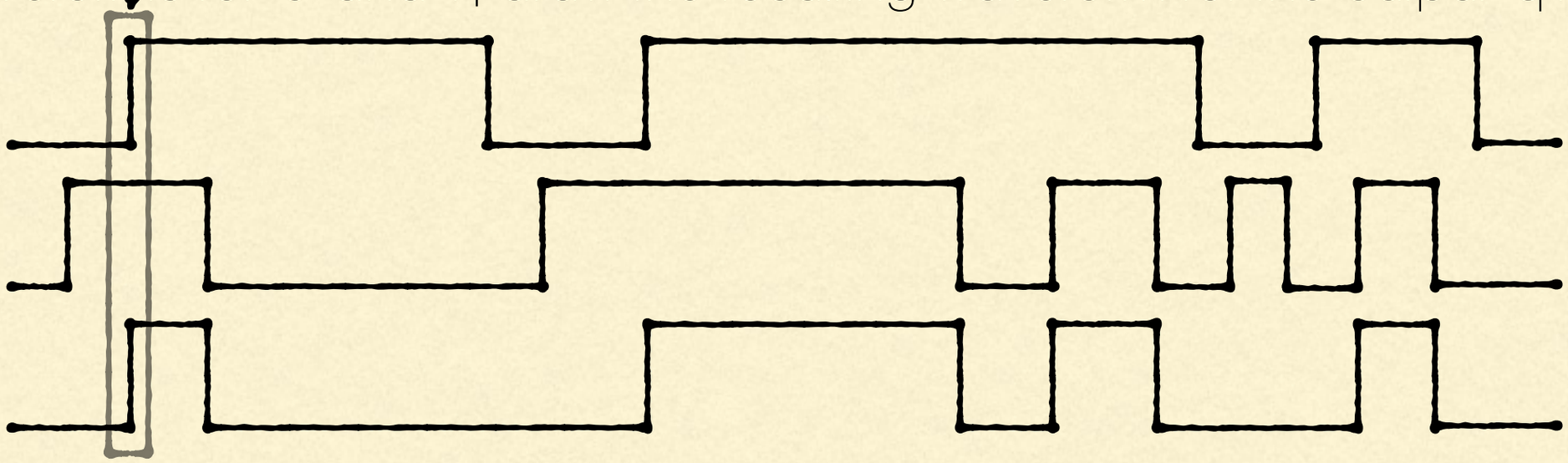
- Given the waveforms for 's' and 'i', draw the resulting waveform for the output 'q':
- s
- i (input)
- q (output)



Recall that for the flip-flop below, the value of 'output' will be equal to 'input' only when 's' is equal to 1 (high state). When the value of 's' is equal to 0 (low state), then the value of output will be locked (and won't change)

As soon as 's' changes to 1 (high state), the flip-flop will unlock and the output will be equal to input. Output becomes 1.

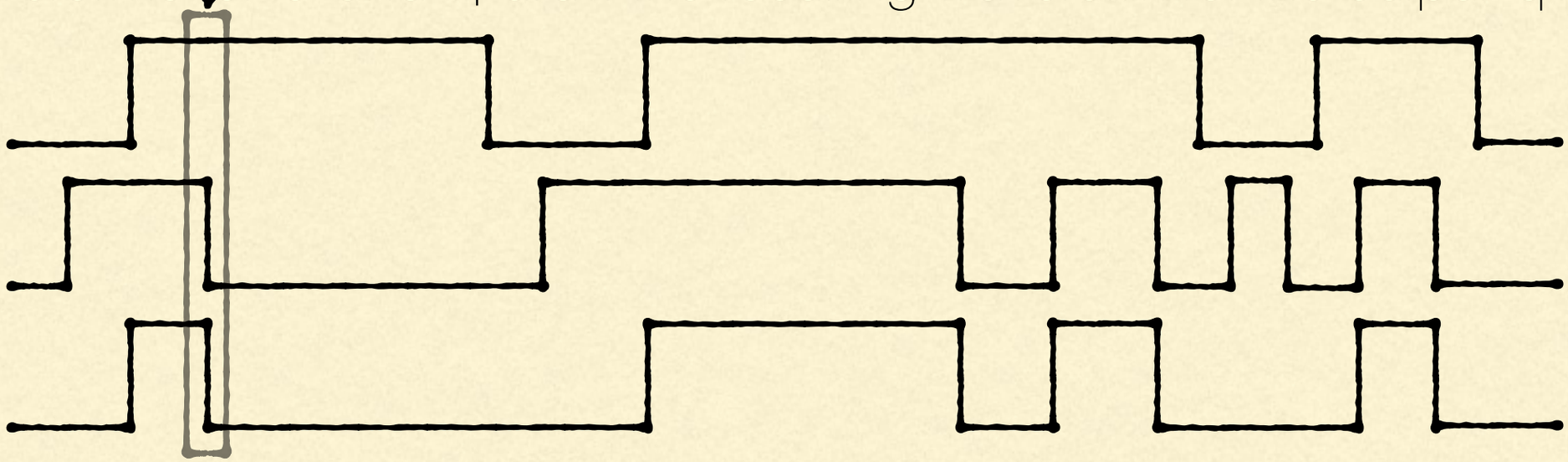
- Given the waveforms for 's' and 'i', draw the resulting waveform for the output 'q':
- s
- i (input)
- q (output)



Recall that for the flip-flop below, the value of 'output' will be equal to 'input' only when 's' is equal to 1 (high state). When the value of 's' is equal to 0 (low state), then the value of output will be locked (and won't change)

Since 's' is equal to 1 (flip-flop still unlocked), when we change the input 'i' to 0, the output 'q' will also become 0

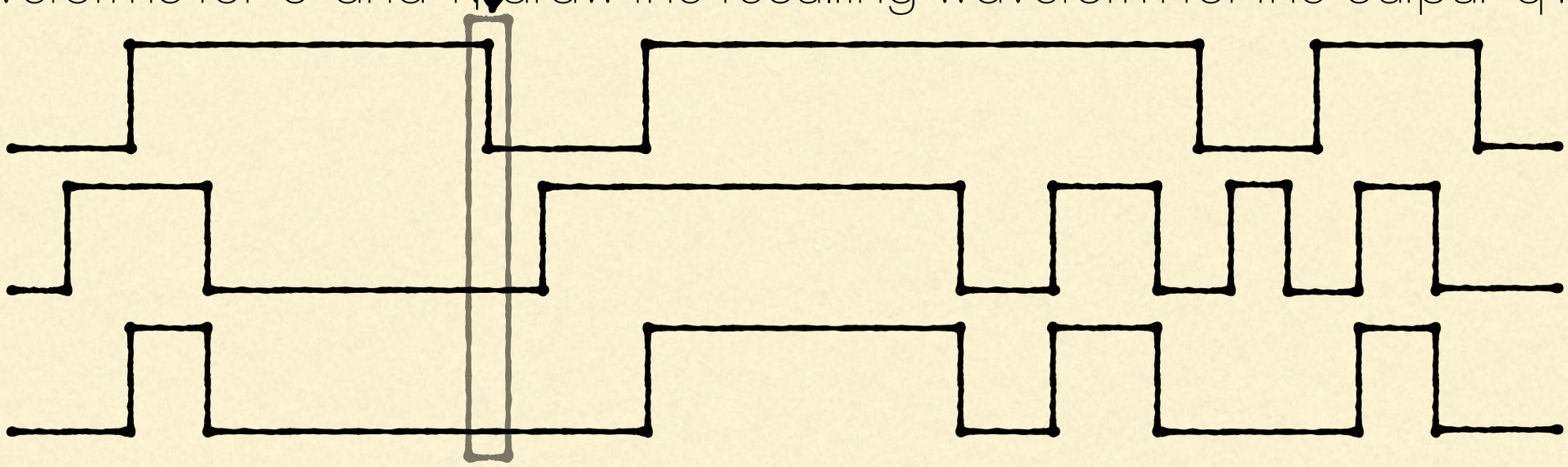
- Given the waveforms for 's' and 'i', draw the resulting waveform for the output 'q':
- s
- i (input)
- q (output)



Recall that for the flip-flop below, the value of 'output' will be equal to 'input' only when 's' is equal to 1 (high state). When the value of 's' is equal to 0 (low state), then the value of output will be locked (and won't change)

Here, 's' changes to 0, meaning that the flip-flop will lock the current value of output

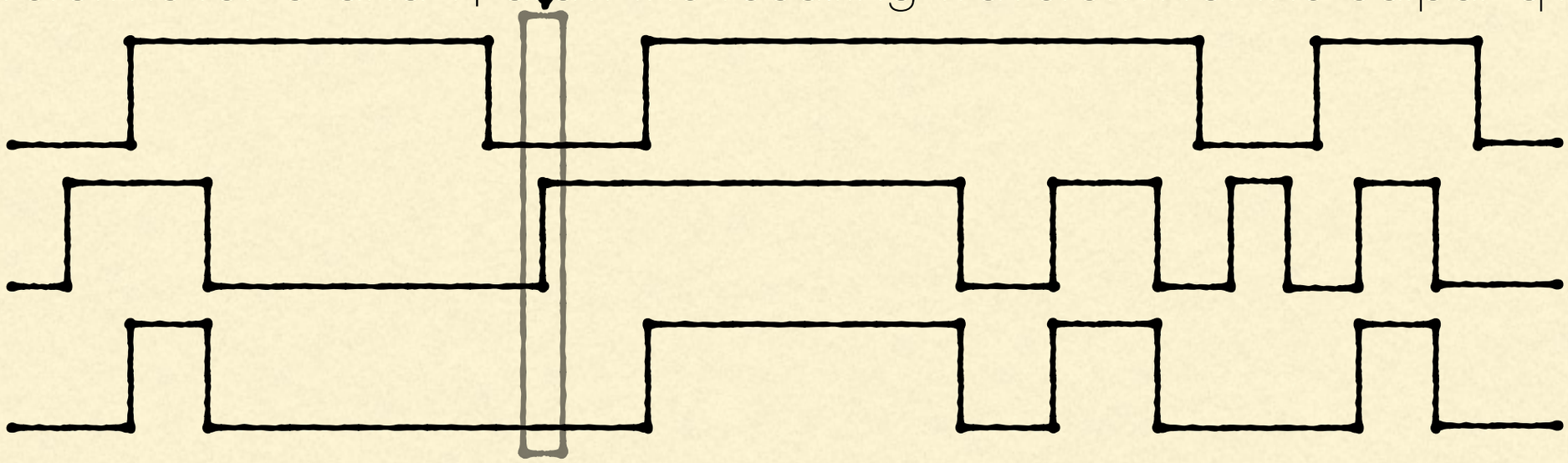
- Given the waveforms for 's' and 'i', draw the resulting waveform for the output 'q':
- s
- i (input)
- q (output)



Recall that for the flip-flop below, the value of 'output' will be equal to 'input' only when 's' is equal to 1 (high state). When the value of 's' is equal to 0 (low state), then the value of output will be locked (and won't change)

Here, although we changed the input 'i' to 1, since 's' is equal to 0 (flip-flop locked), the output won't change

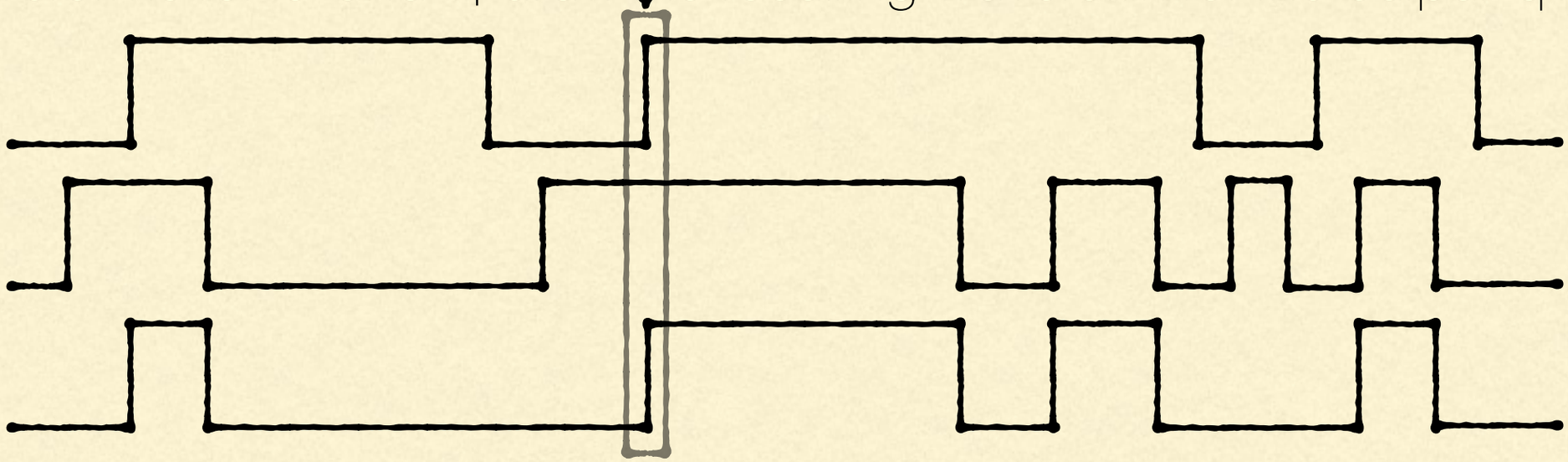
- Given the waveforms for 's' and 'i', draw the resulting waveform for the output 'q':
- s
- i (input)
- q (output)



Recall that for the flip-flop below, the value of 'output' will be equal to 'input' only when 's' is equal to 1 (high state). When the value of 's' is equal to 0 (low state), then the value of output will be locked (and won't change)

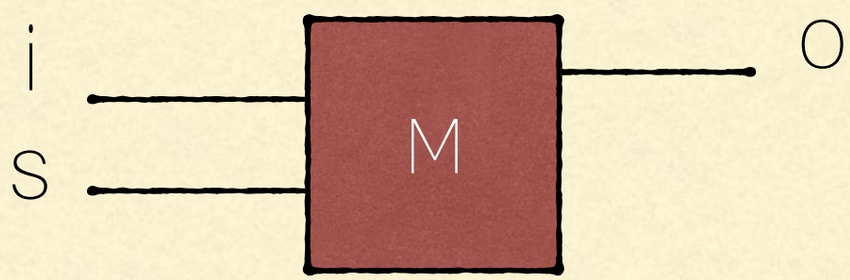
As soon as 's' changes to 1, the flip-flop will unlock, and as a result the output 'q' will be equal to input 'i'. So on and so forth…

- Given the waveforms for 's' and 'i', draw the resulting waveform for the output 'q':
- s
- i (input)
- q (output)



CONSTRUCTING A BIT

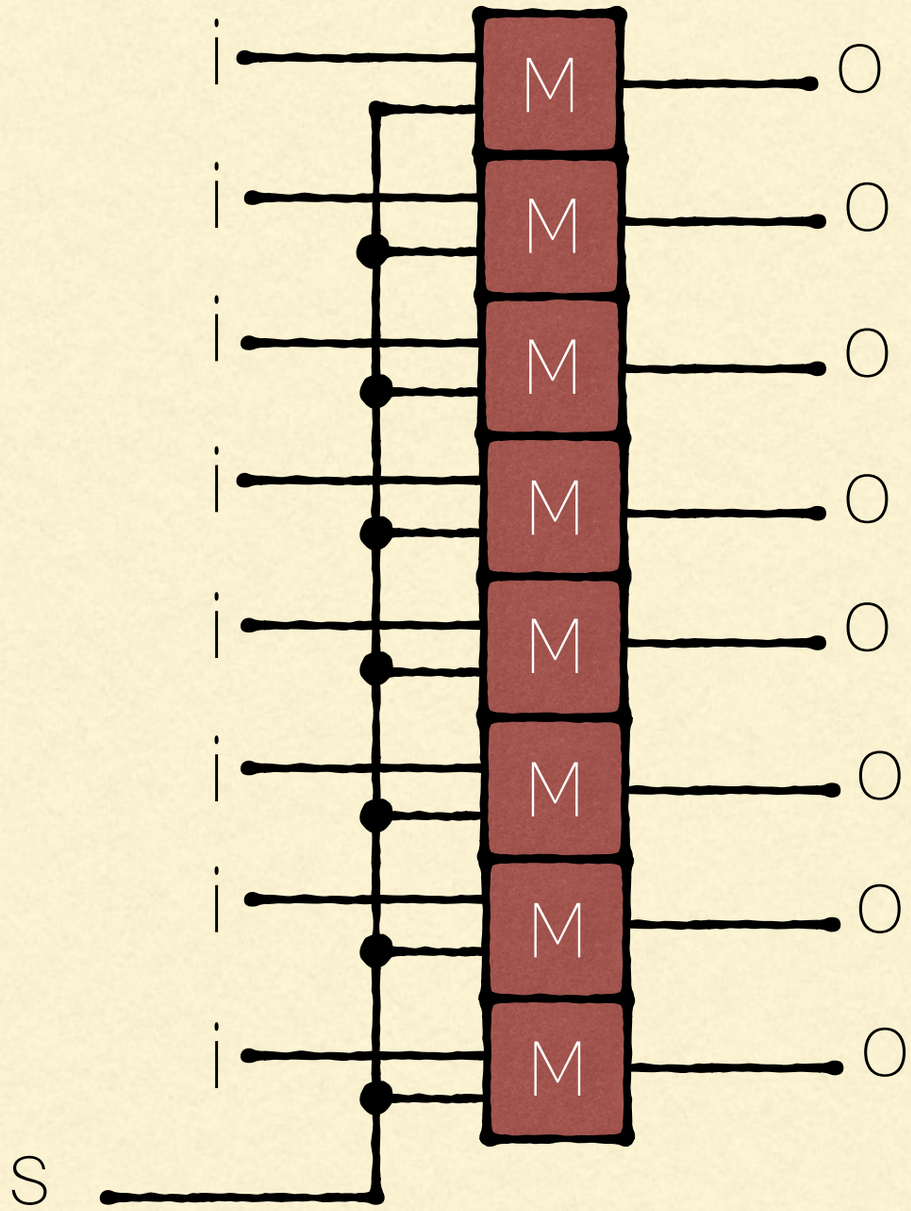
- Now that we know how one bit is constructed, we no longer need to look at that tricky internal wiring. From now on, we will just use this diagram to represent it;



- 'i' is the input bit that we'd like to save. 's' is the input that allows 'i' into the memory when 's' is on, and locks it in place when 's' is off

FROM A BIT TO A BYTE

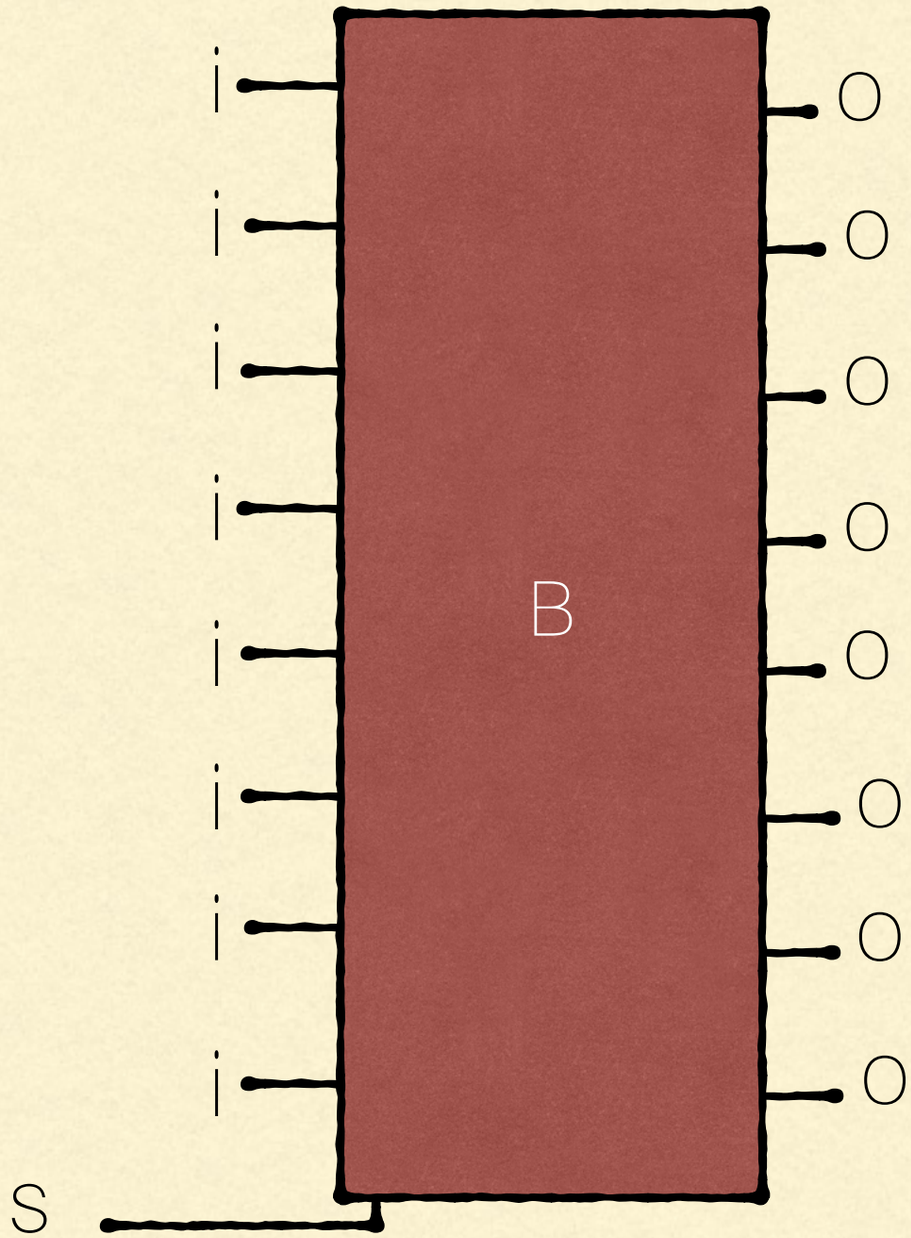
- Here is a diagram of how a byte is constructed from 8 bits:
- We have taken eight of our memory bits, each one still has its own data input 'i' and its out output 'o' but we have wired all 8 of the set inputs 's' together



FROM A BIT TO A BYTE

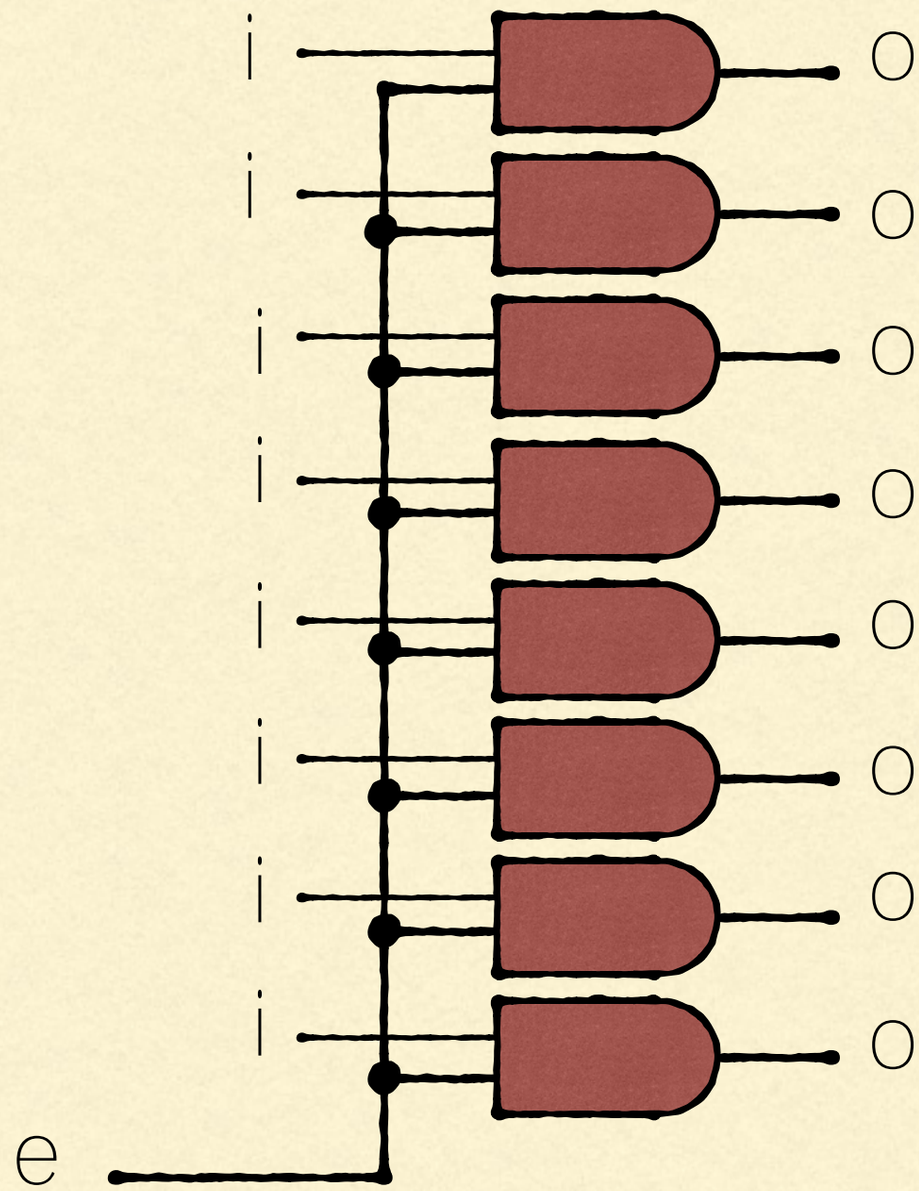
- Here is a diagram of how a byte is constructed from 8 bits:

- We have taken eight of our memory bits, each one still has its own data input 'i' and its out output 'o' but we have wired all 8 of the set inputs 's' together



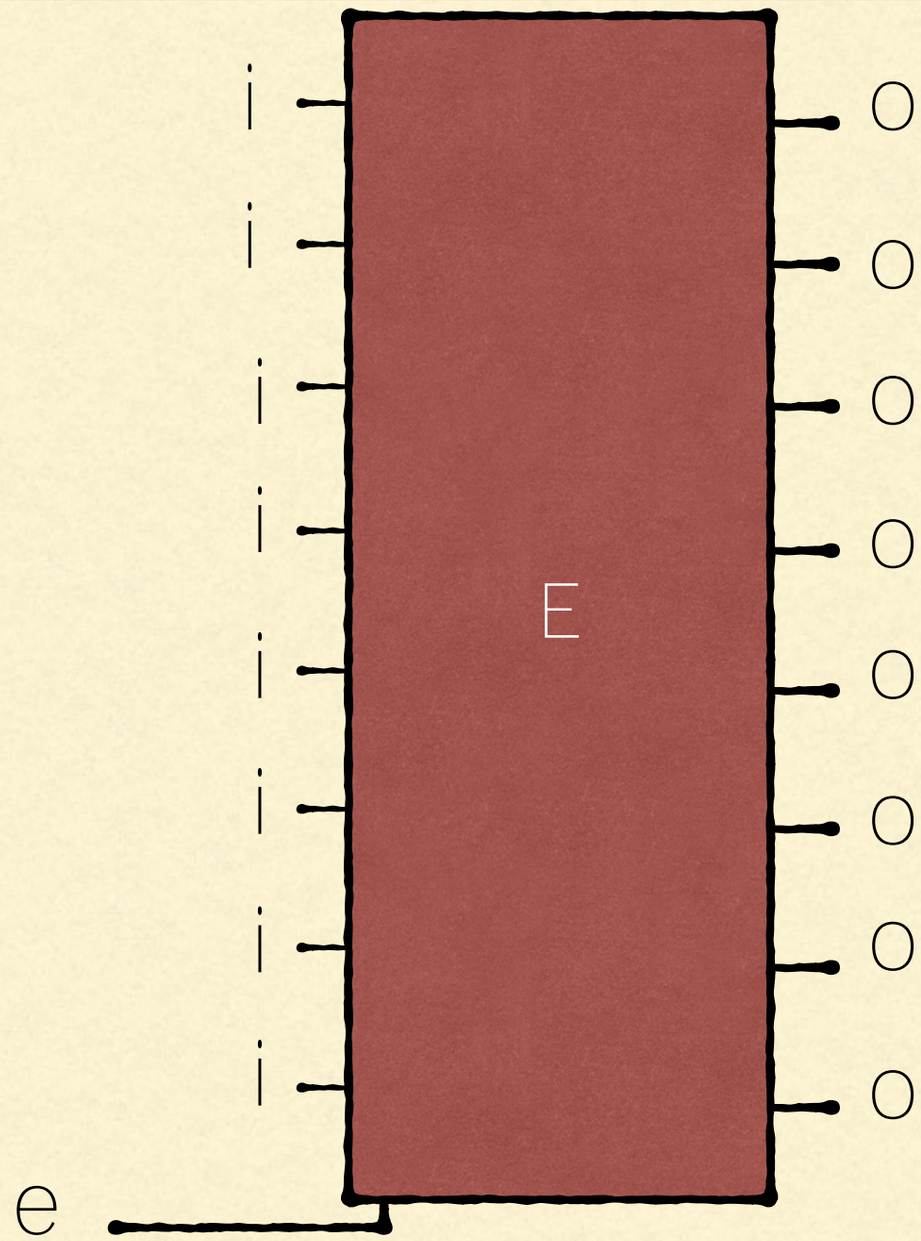
FROM A BIT TO A BYTE

- Here is a diagram of an enabler:
  - if the value of 'e' is 0, the input will not be transferred to the output
  - If the value of 'e' is 1, the output will be equal to the input



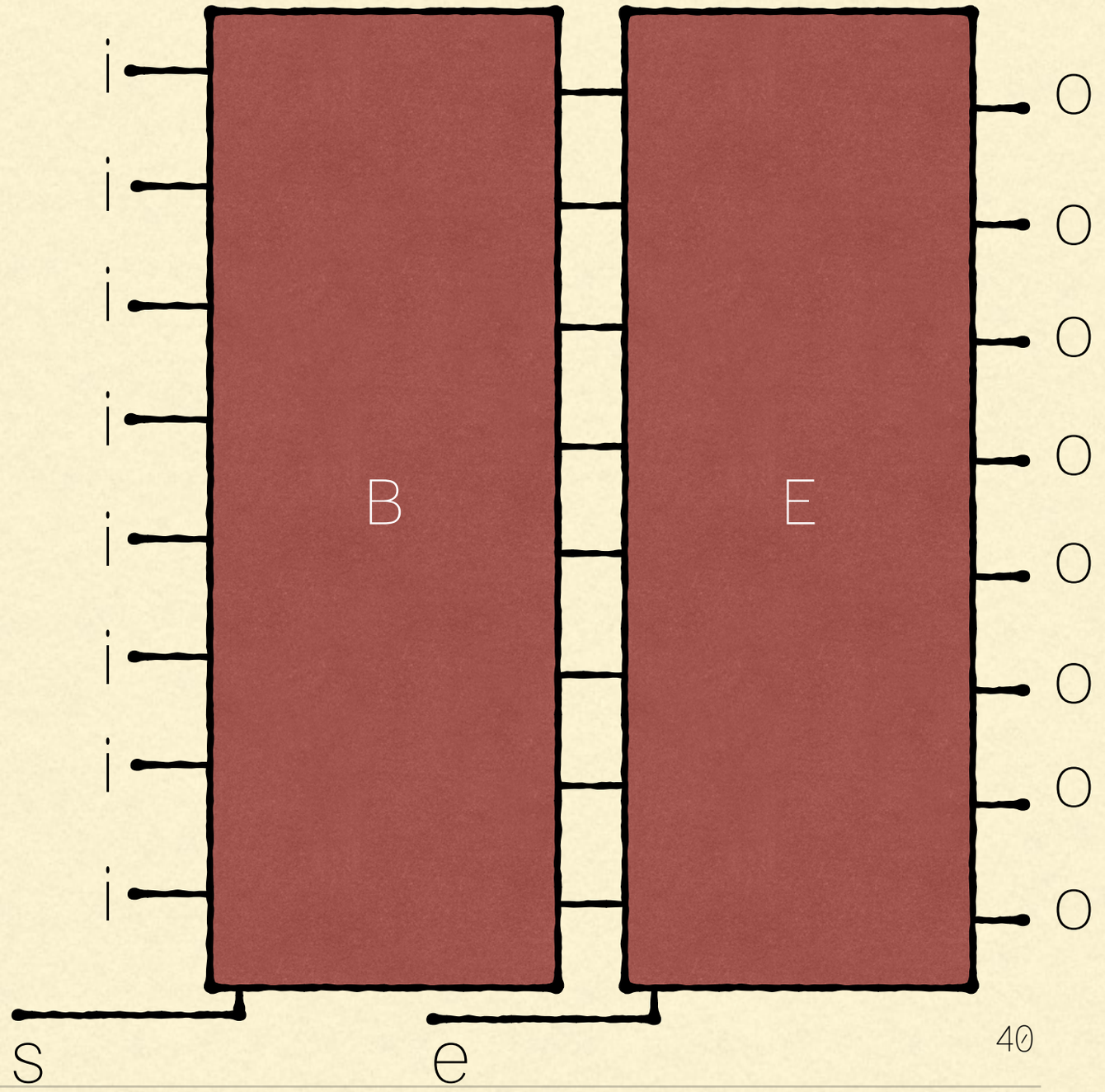
FROM A BIT TO A BYTE

- Here is a diagram of an enabler:
  - if the value of 'e' is 0, the input will not be transferred to the output
  - If the value of 'e' is 1, the output will be equal to the input



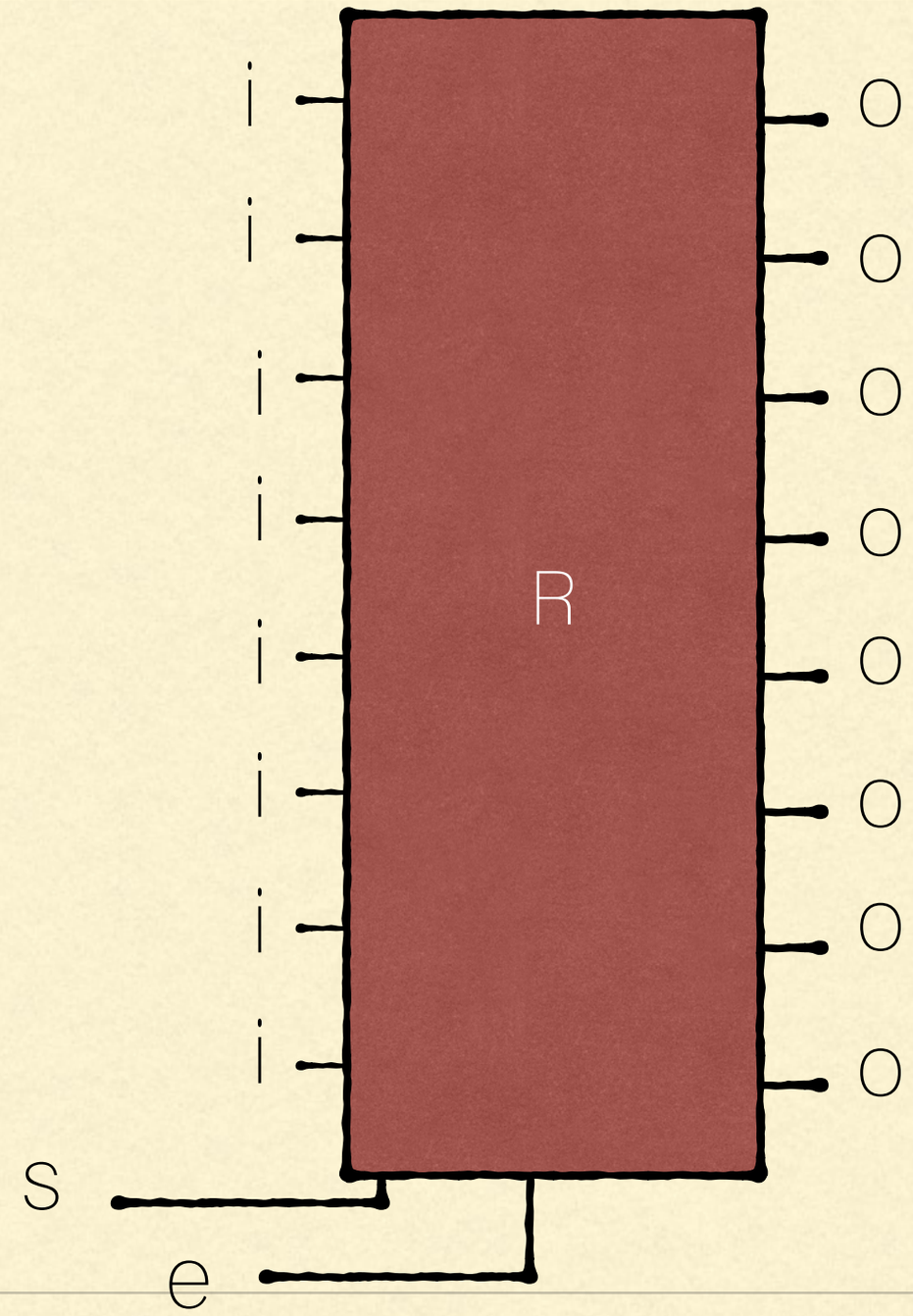
8-BIT REGISTERS

- Let's take a byte, and connect it to an enabler, as shown in graph to the right.
- Now we have a combination that can store eight bits. It captures them all at the same time, and it can either keep them to itself, or let them out to be used use somewhere else. This combination of a Byte and an Enabler is called a Register.



8-BIT REGISTERS

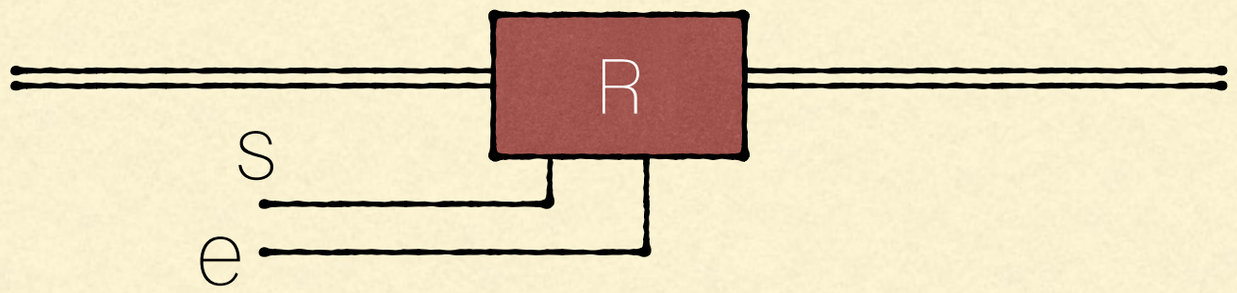
- Let's take a byte, and connect it to an enabler, as shown in graph to the right.
- Now we have a combination that can store eight bits. It captures them all at the same time, and it can either keep them to itself, or let them out to be used use somewhere else. This combination of a Byte and an Enabler is called a Register.



THE BUS

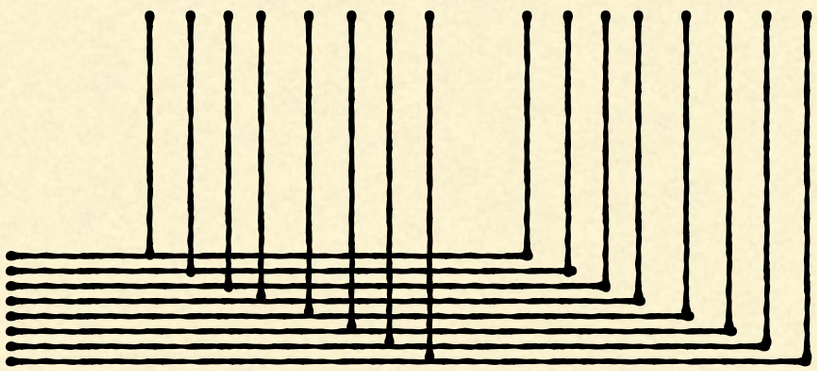
- There are many places in a computer where 32 (or 8 for simplicity) wires are needed to connect registers together. Our simplified registers have eight memory bits, each of which have an input and an output

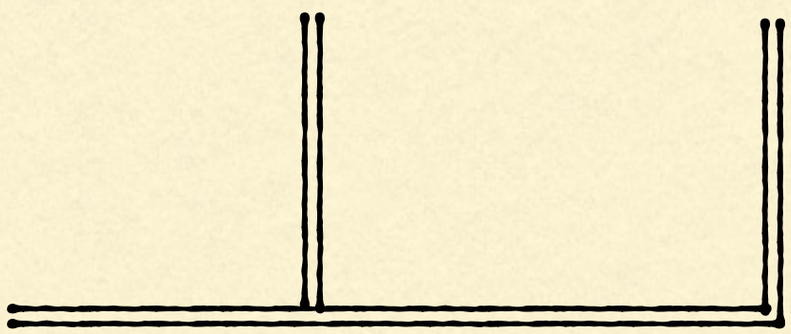
  - To simplify our diagram, we will replace our eight wires with a double line:



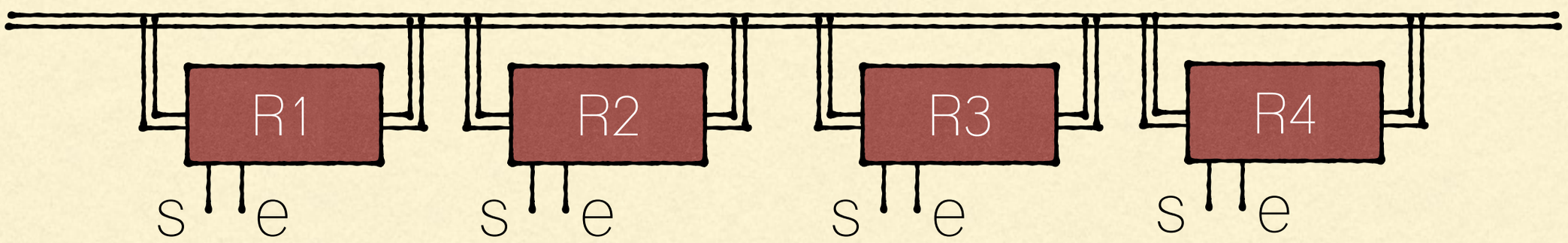
THE BUS

- Where there is a connection between two of these bundles of wires, one wire of each bundle is connected to one wire of the other bundle as shown in the diagram on the left, but we will simplify it and just draw it like the diagram on the right.



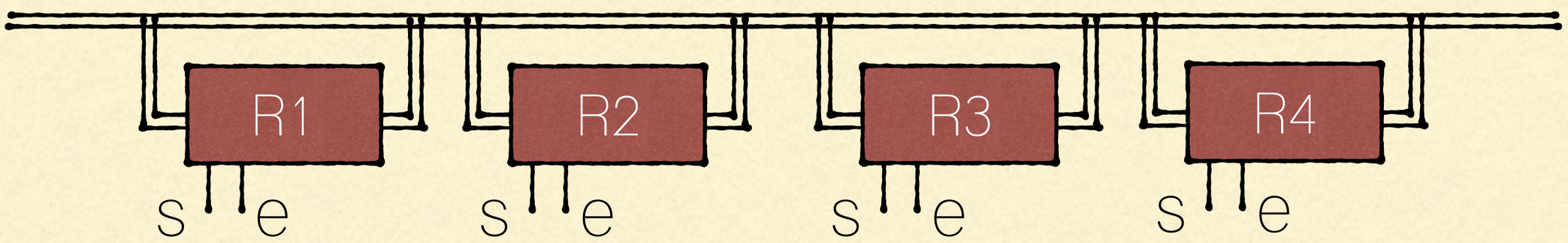


BUS EXMPLE TO COPY VALUES OF REGISTERS



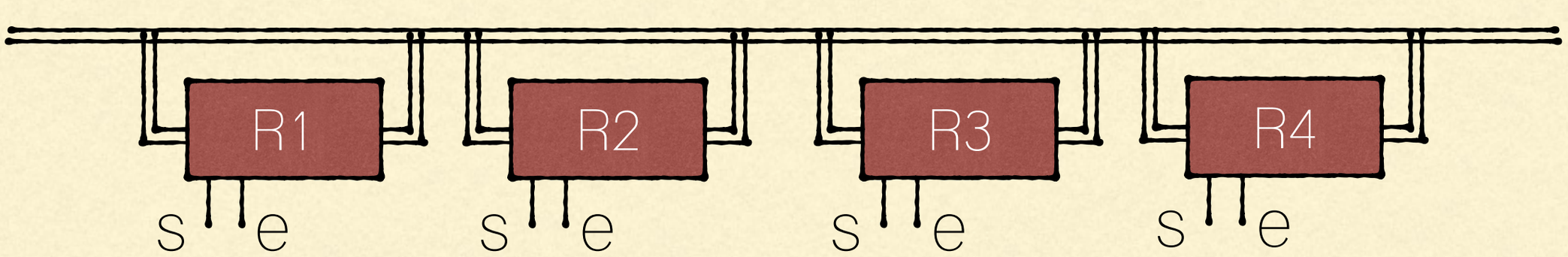
BUS EXMPLE TO COPY VALUES OF REGISTERS

- How can we copy the values from one register to another?

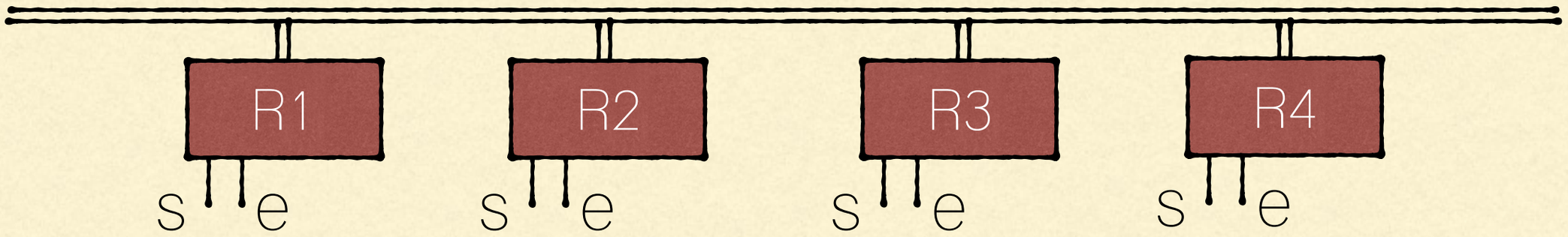


BUS EXMPLE TO COPY VALUES OF REGISTERS

- How can we copy the values from one register to another?
- e = 1 for the register we want to copy from, s = 1 for the register(s) we want to copy to



These two diagrams are identical



DECODER

- A binary decoder has n output bits, and the integer inputs bits serve as the "address" or bit number of the output bit that is to be activated.

- This type of decoder asserts exactly one of its n output bits, for every unique combination of input bit states. Each output bit becomes active only when a specific, corresponding integer value is applied to the inputs.

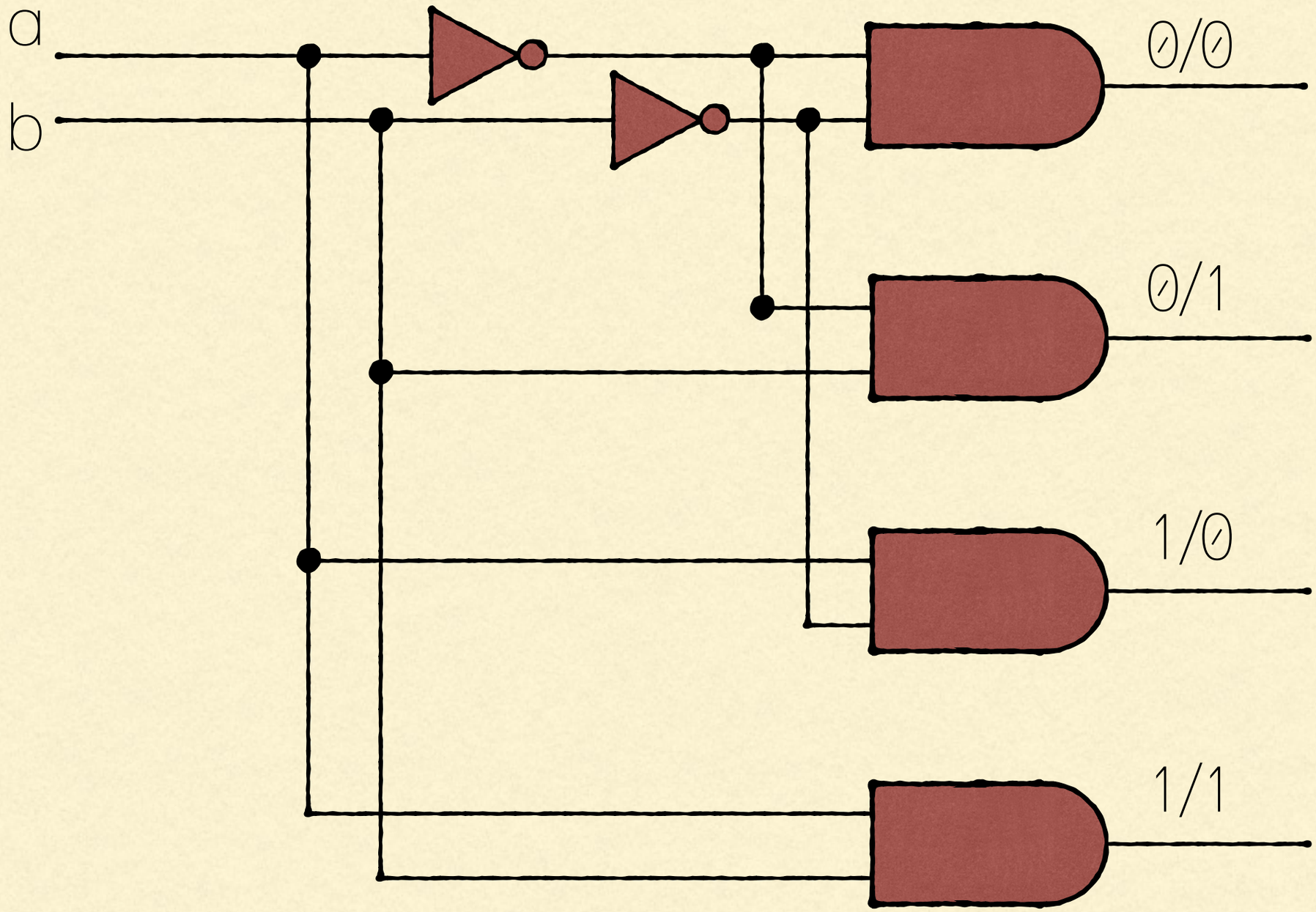
- For example, output bit number 0 is selected when integer value 0 is applied to the integer inputs.

DECODER

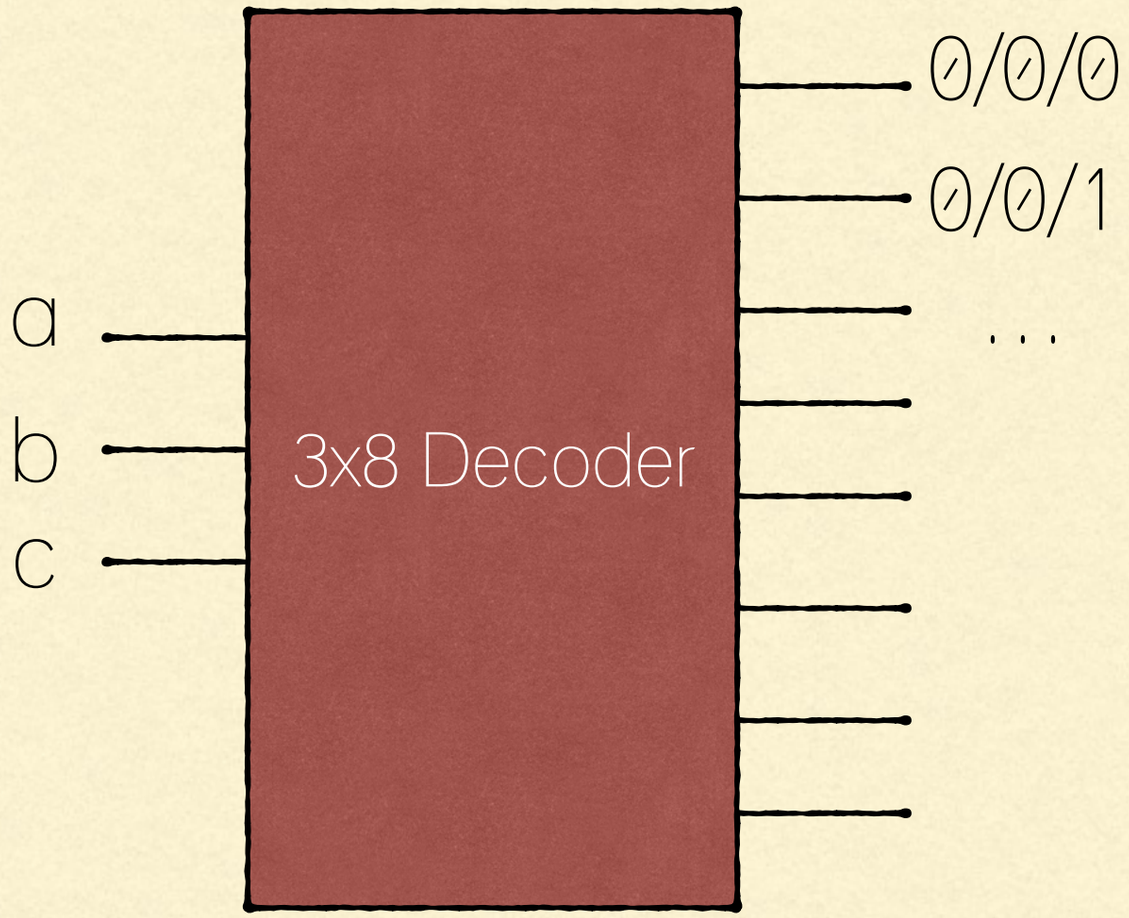
- Truth table for a 2-to-4 line decoder:

a	b	0/0	0/1	1/0	1/1
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

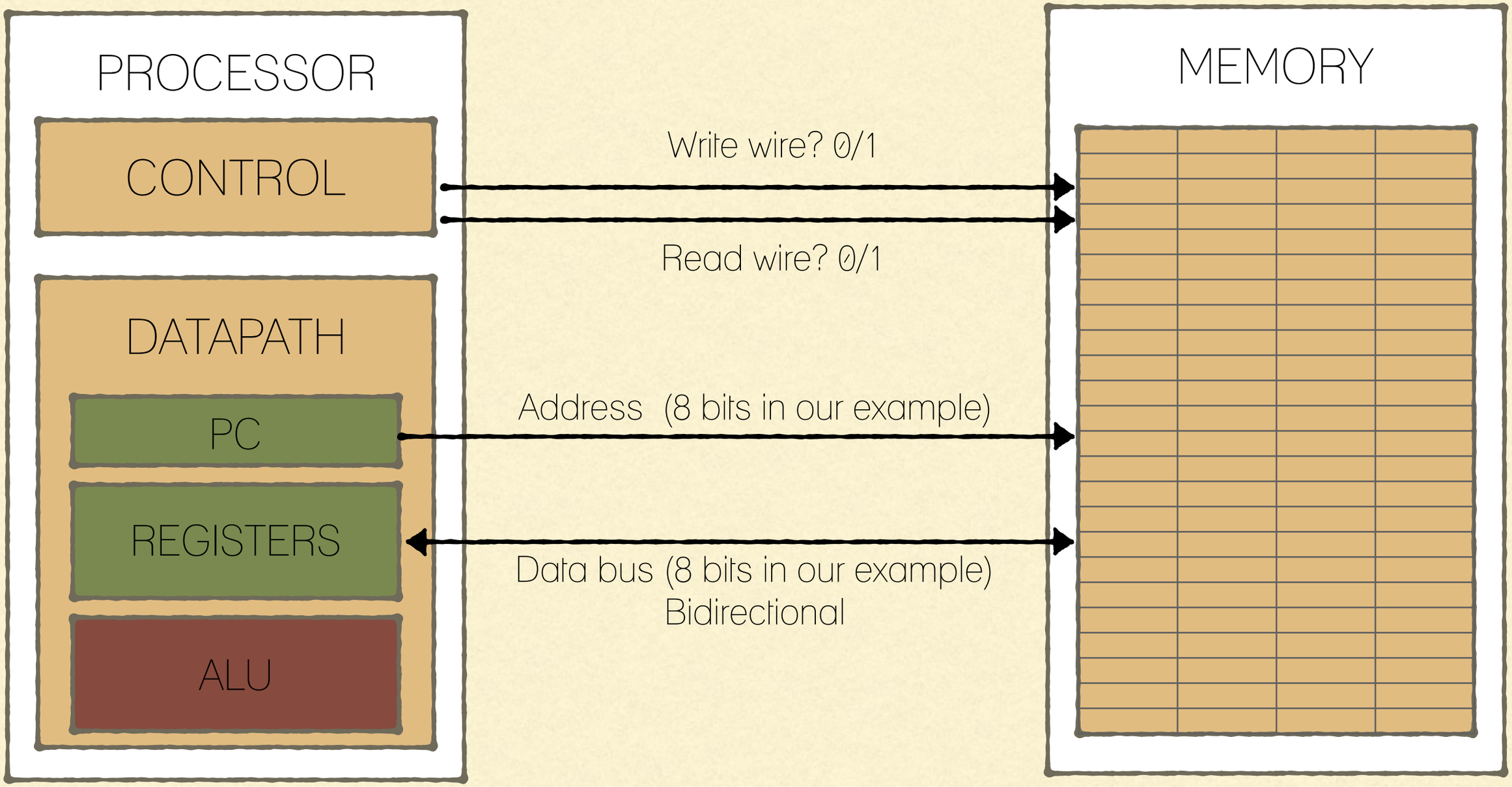
2x4 DECODER



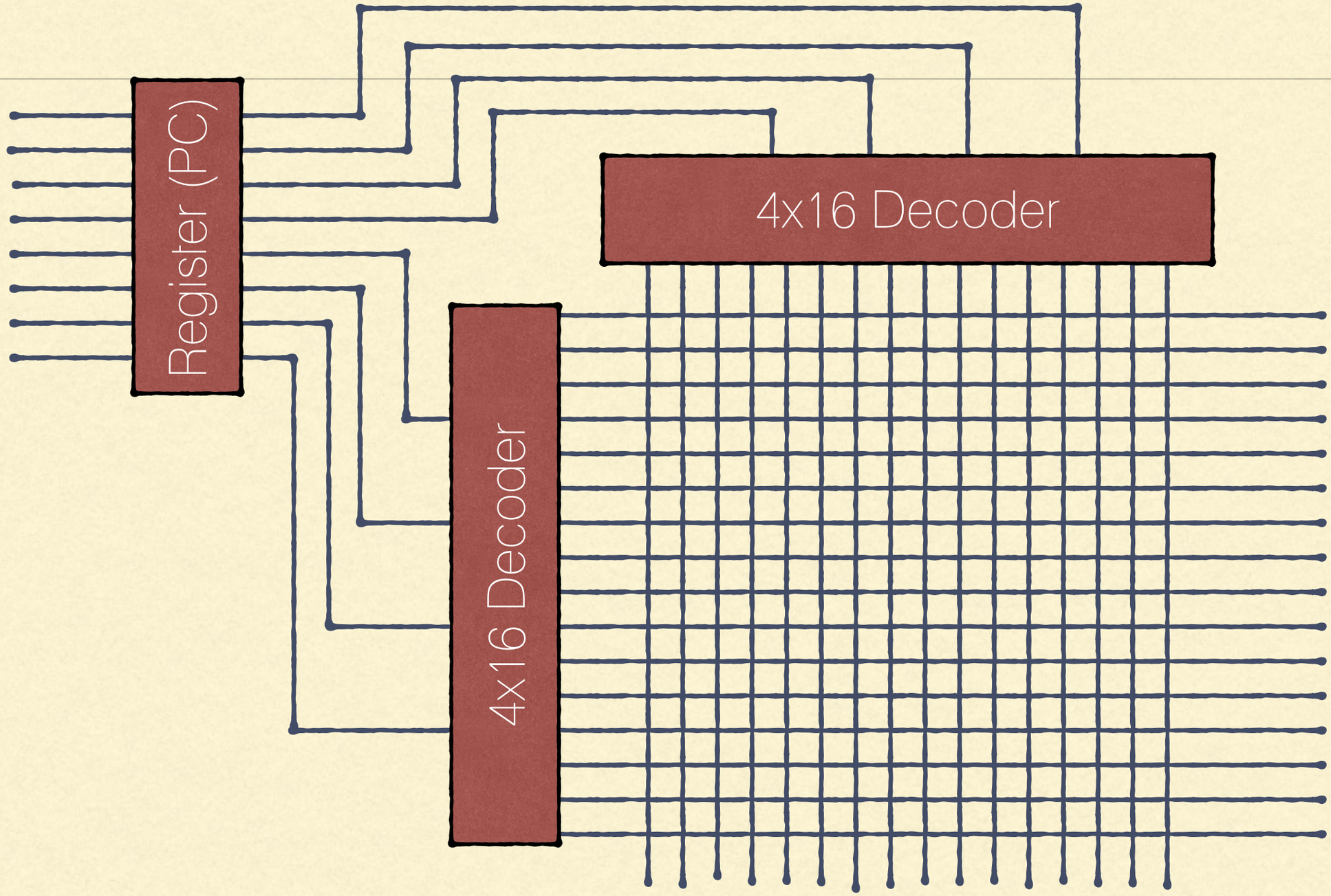
3x8 DECODERS



MEMORY ADDRESSES ARE IN BYTES

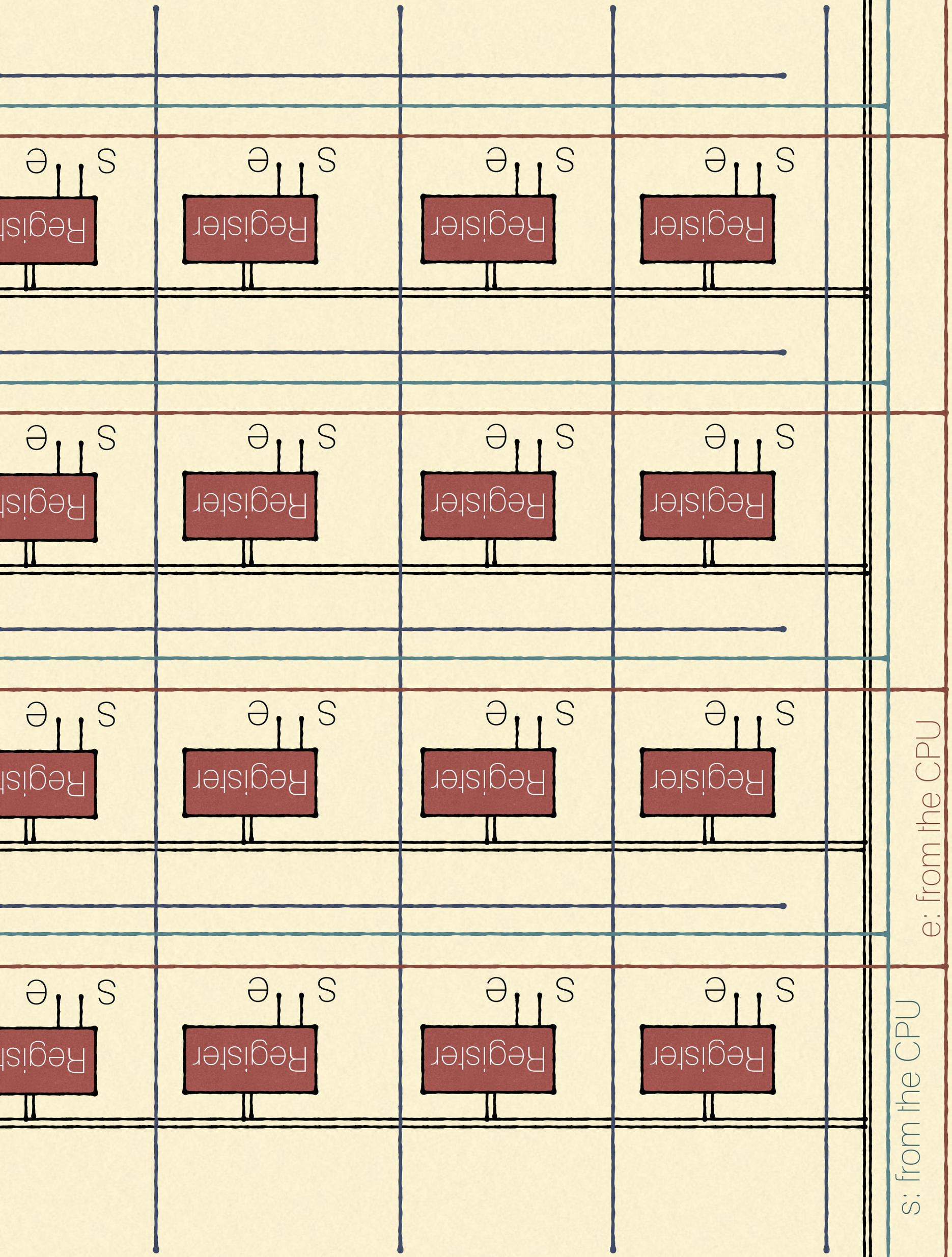


ACCESSING MEMORY



s: from the CPU

e: from the CPU


Register
S
E
S: from the CPU
e: from the CPU

'e' from CPU

Register

e

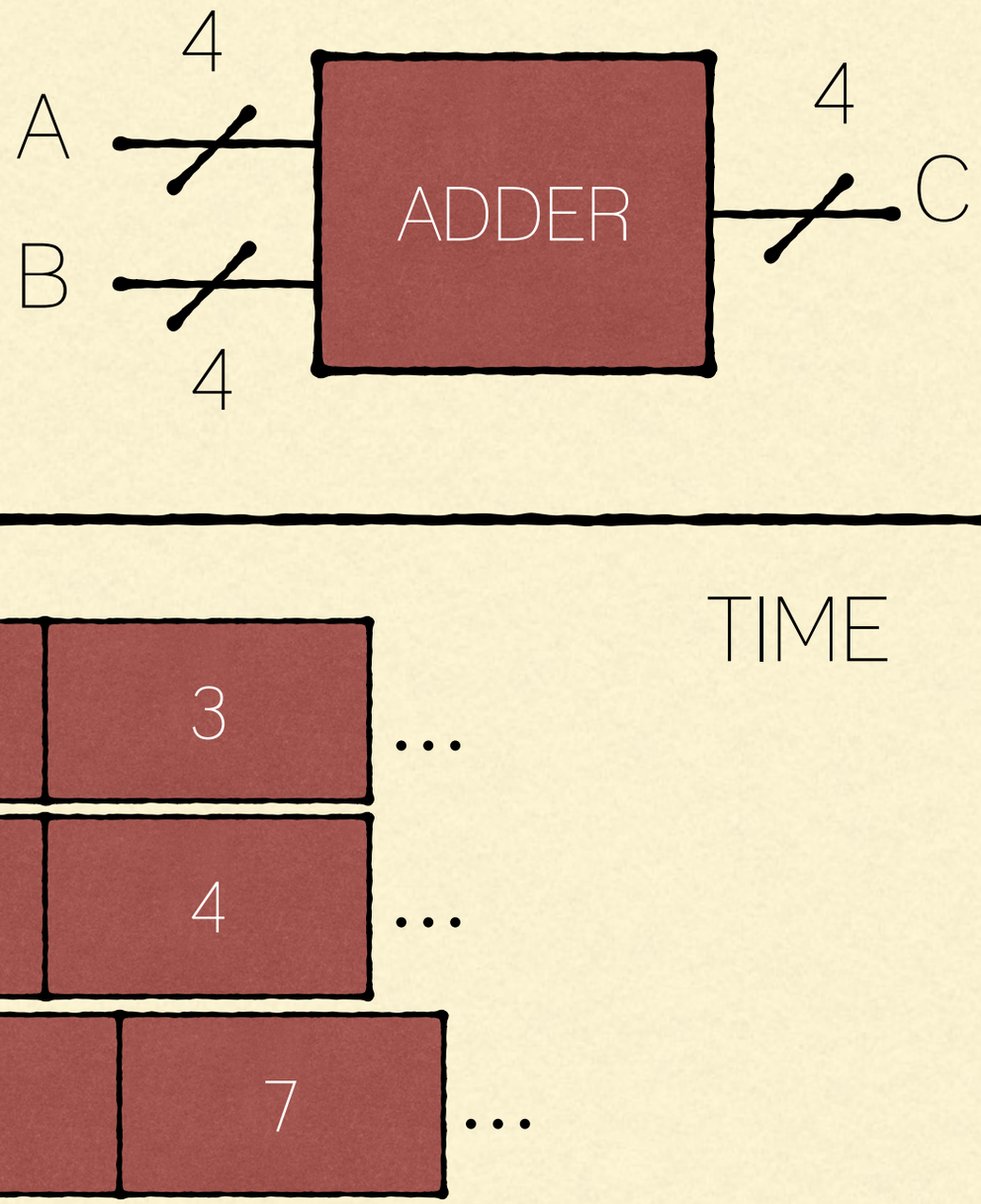
s

Grid

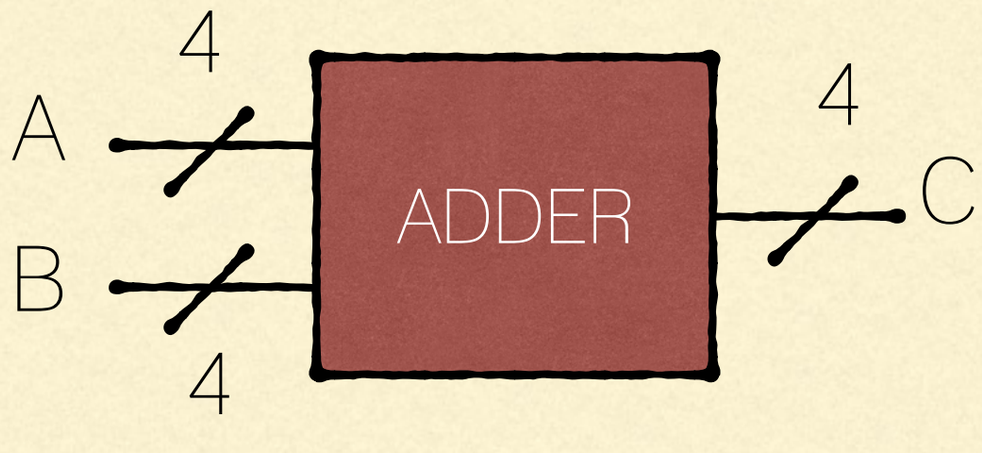
Grid

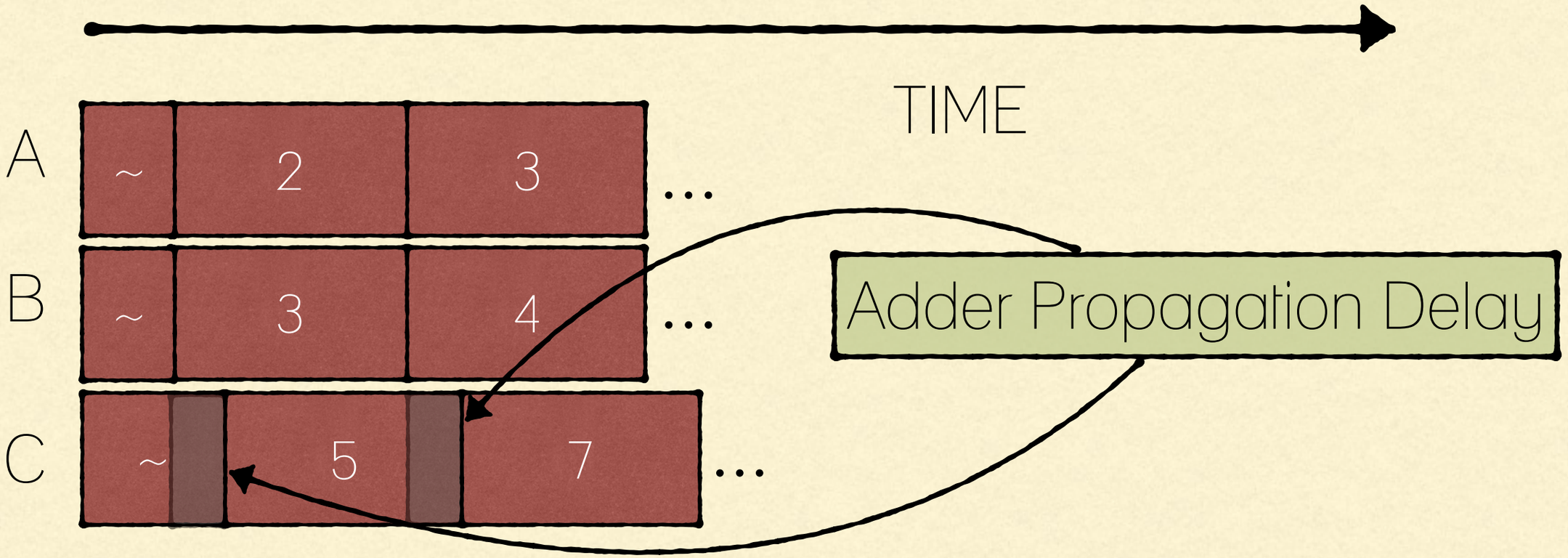
's' from CPU

SIGNALS AND WAVEFORMS; CIRCUIT DELAY

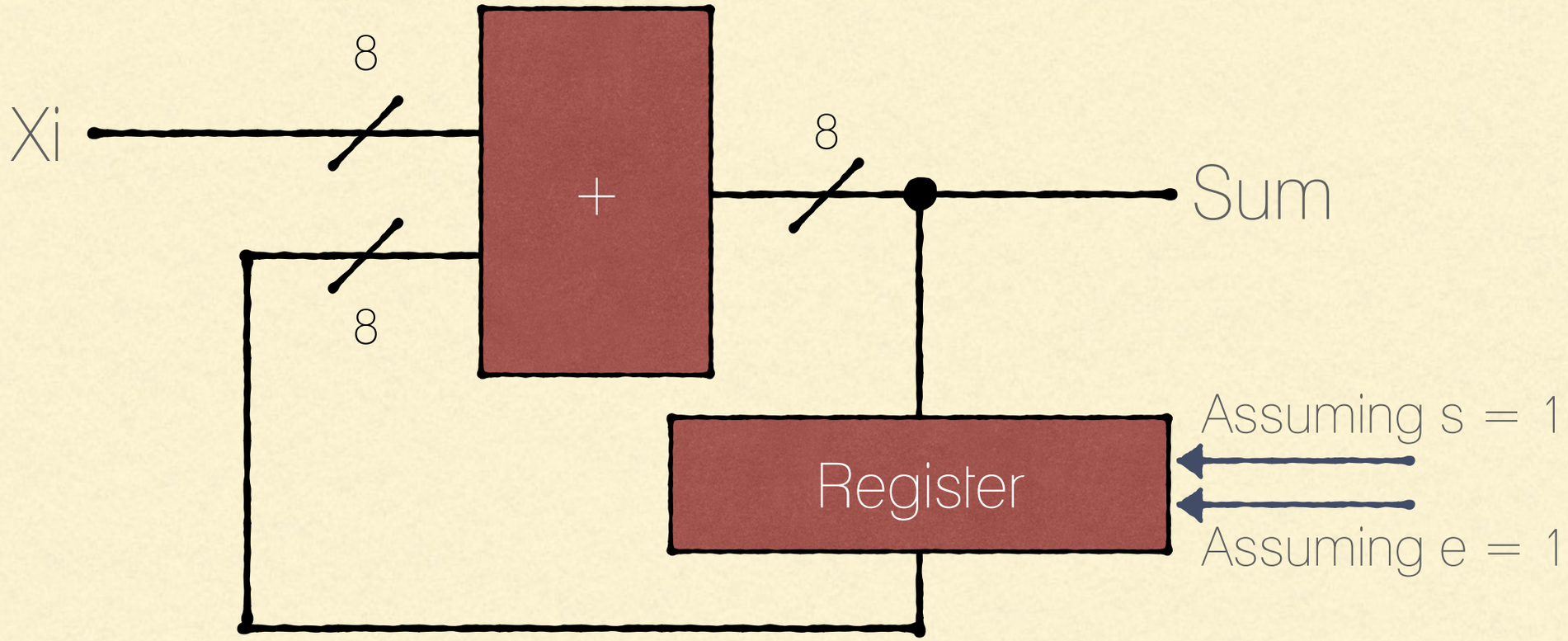


SIGNALS AND WAVEFORMS; CIRCUIT DELAY



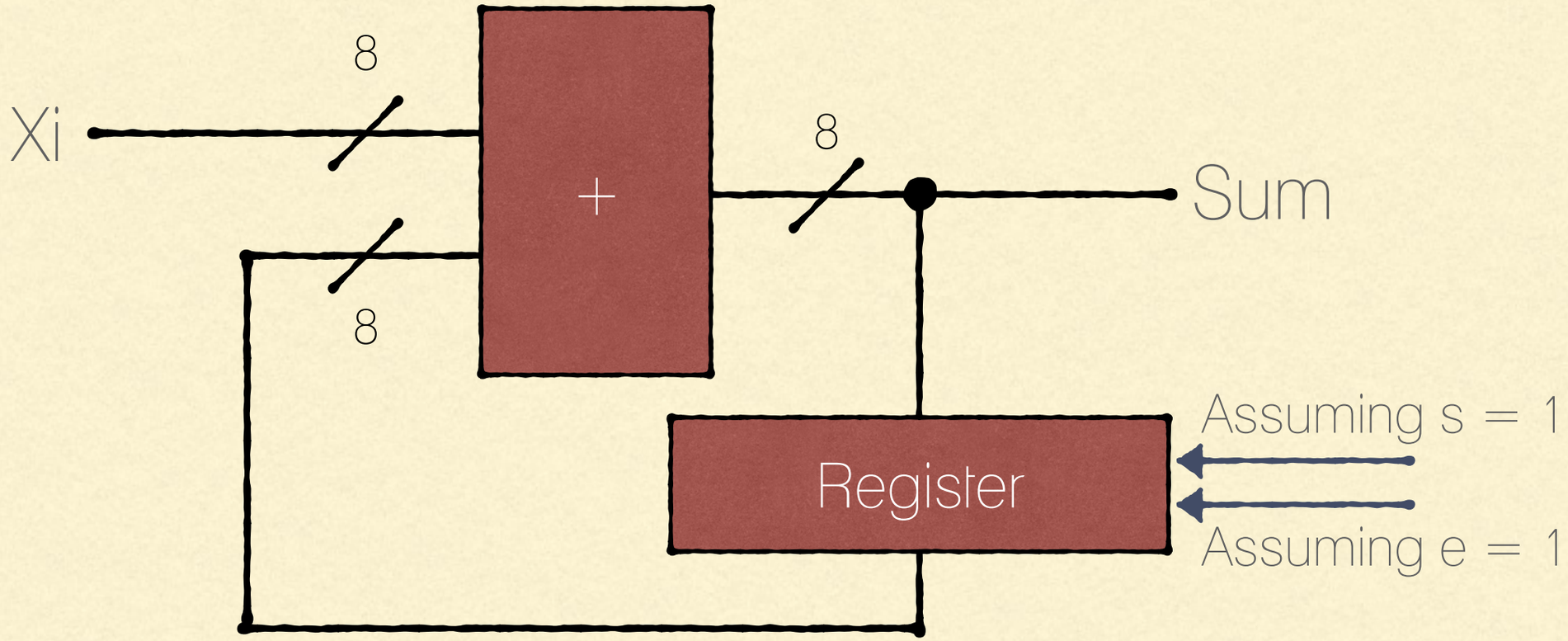


ACCUMULATOR EXAMPLE



- Register is used to hold and transfer the data to adder

ACCUMULATOR EXAMPLE



- Register is used to hold and transfer the data to adder
- However, we need to control the next iteration of the loop — too slow/fast?

INTRODUCING EDGE-TRIGGERED D-TYPE FLIP FLOP

- Edge-triggered d-type flip-flop

On the rising edge of the clock, the input d is sampled and transferred to the output. At all other times, the input d is ignored

- Example waveform:
- CLK
- d (input)
- q (output)