# CMSC 11: Introduction to Computer Science

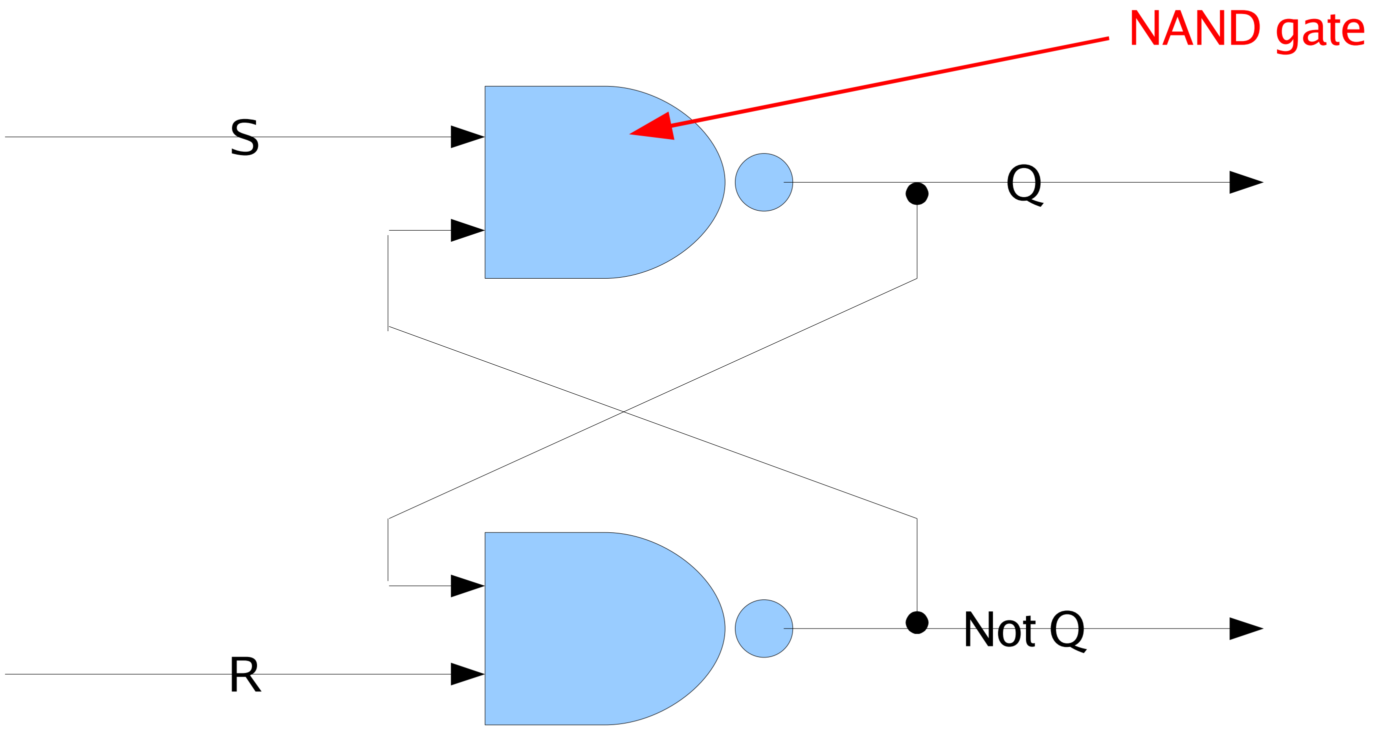
Jaderick P. Pabico <jppabico@uplb.edu.ph>
Institute of Computer Science, CAS, UPLB

# Review

- Logic diagram of a 1-bit adder and sequence of 1-bit adders to create a many-bit adder
- Encoding of decimal numbers to binary
  - direct encoding
  - binary coded decimal
- Encoding of symbols and alphabet to binary
  - ASCII, EBCDIC
- Encoders, decoders, ALU, flip-flop, NAND-gate

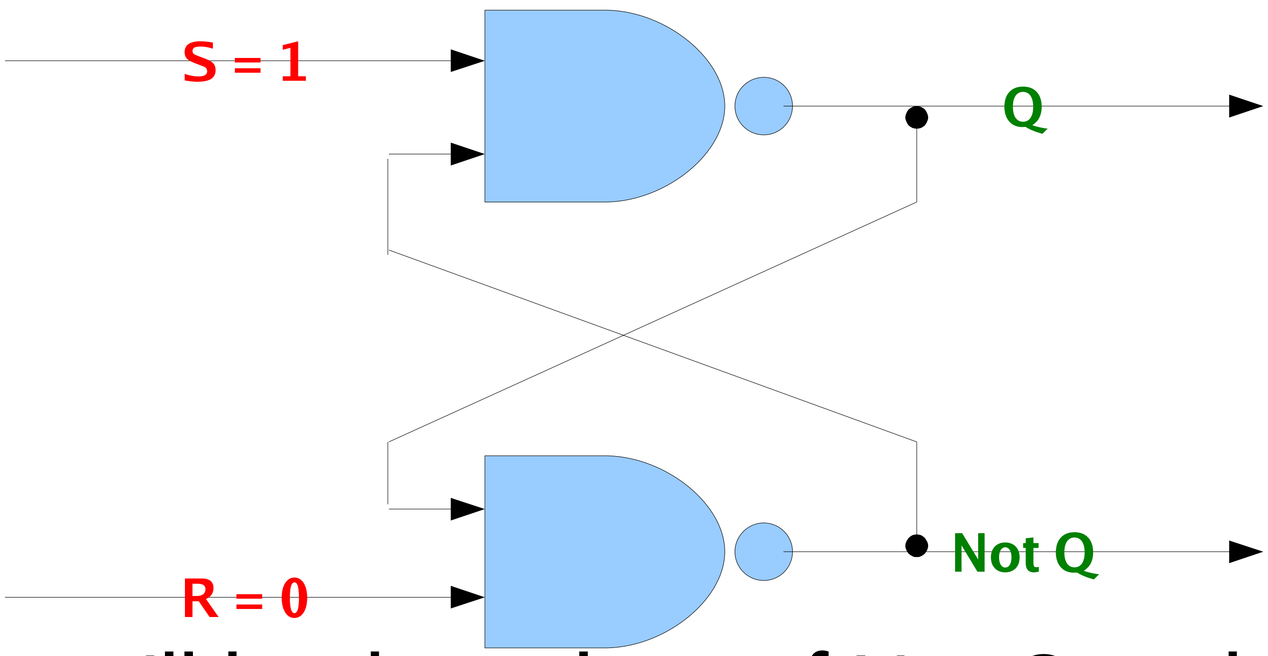
# Flip-Flop

- Refresh our memory how a flip-flop looks like
- And flip-flop was made to remember an output



# Flip-Flop

- Now let's put flip-flop in action
- Suppose S=1 and R=0

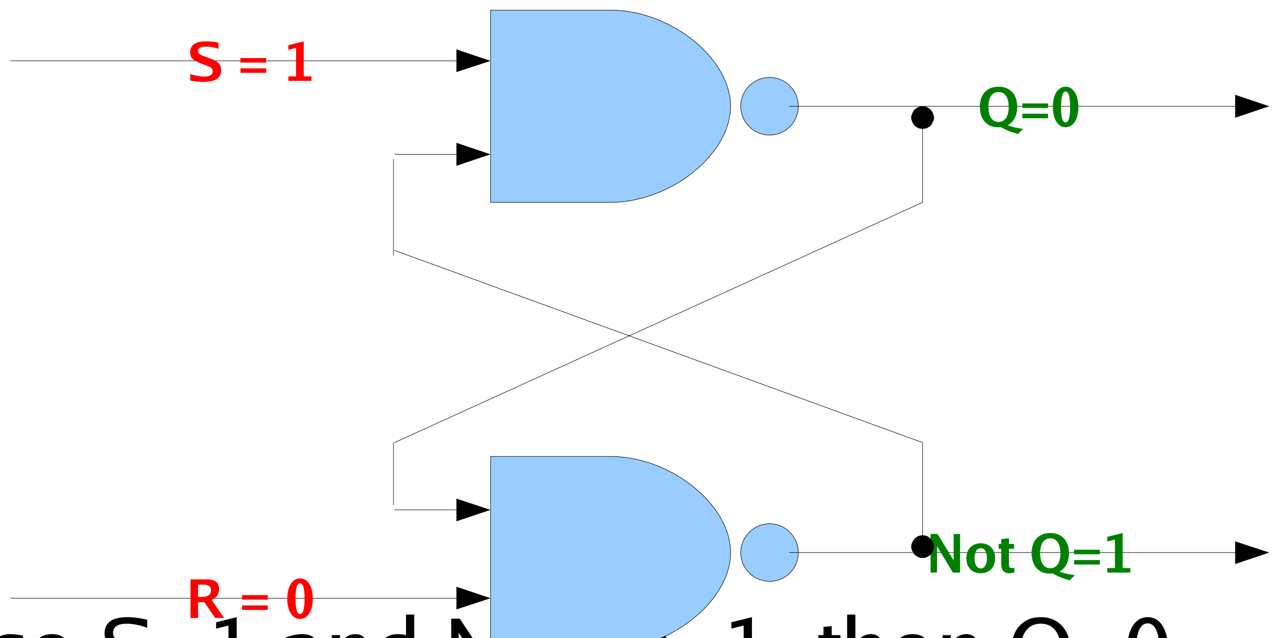



- What will be the values of Not-Q and Q?

# Flip-Flop

| | | |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

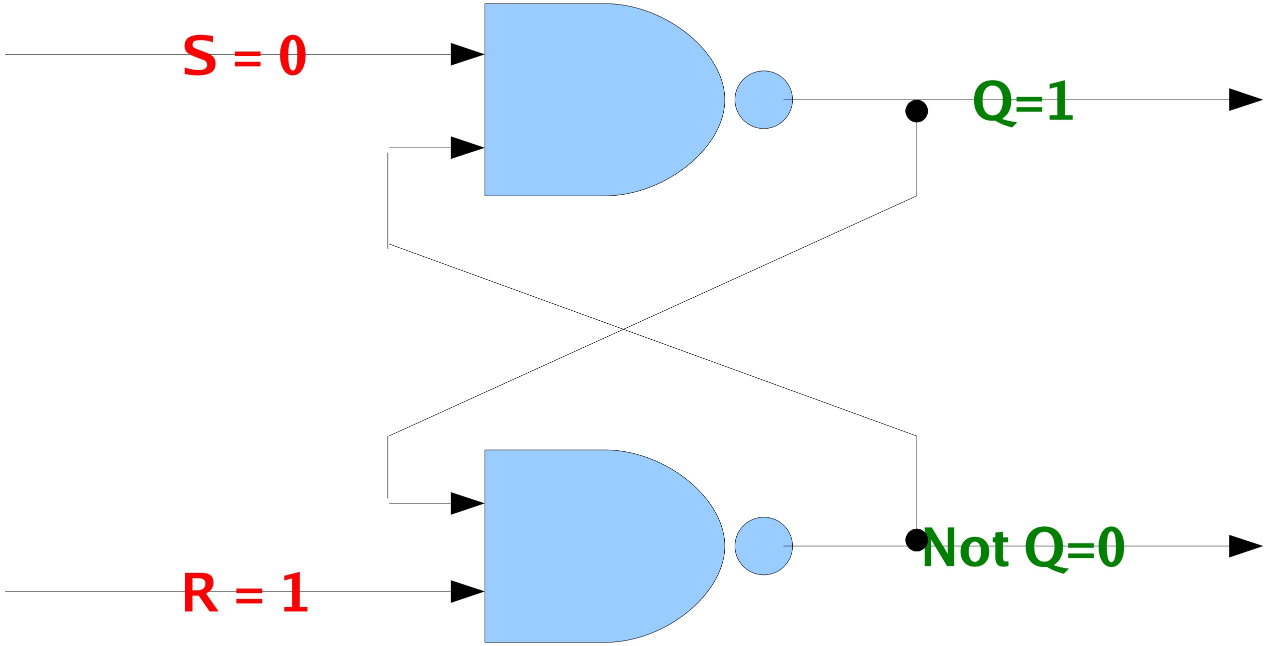
- Using the NAND I/O table, Not-Q = 1




- Since S=1 and Not-Q=1, then Q=0

# Flip-Flop

- If, we instead have S=0 and R=1
- Same diagram turned upside down

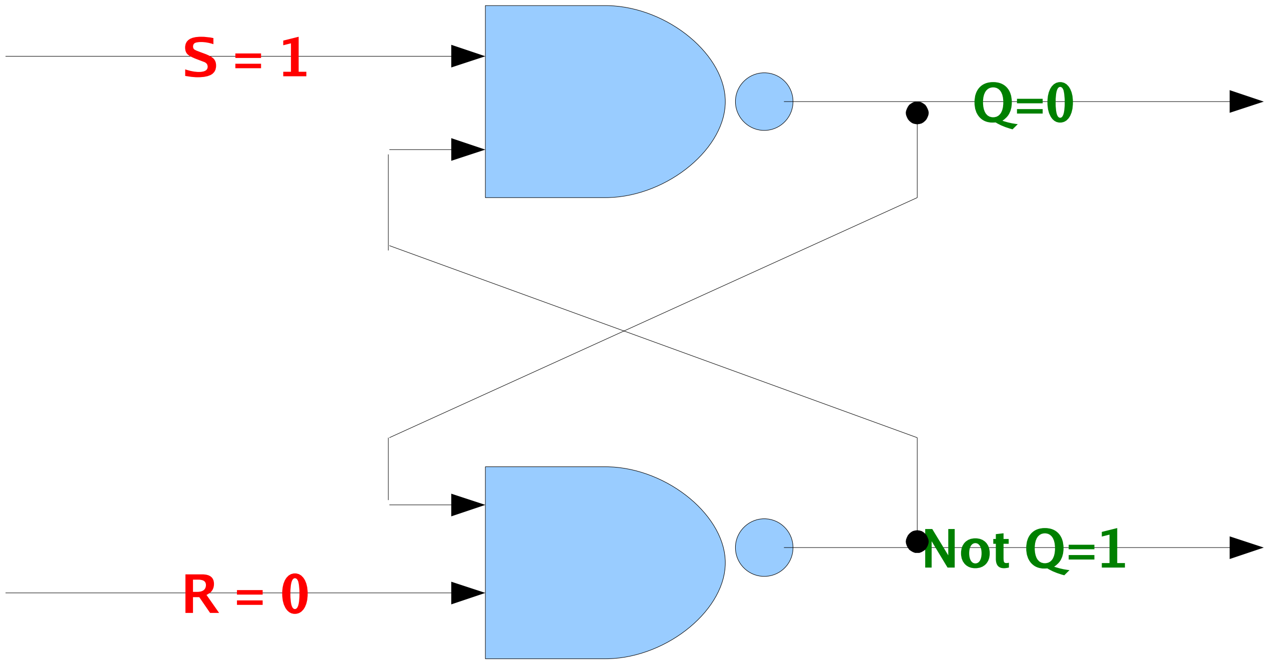



OK, great! But where's the memory there?

Yeah! Where?

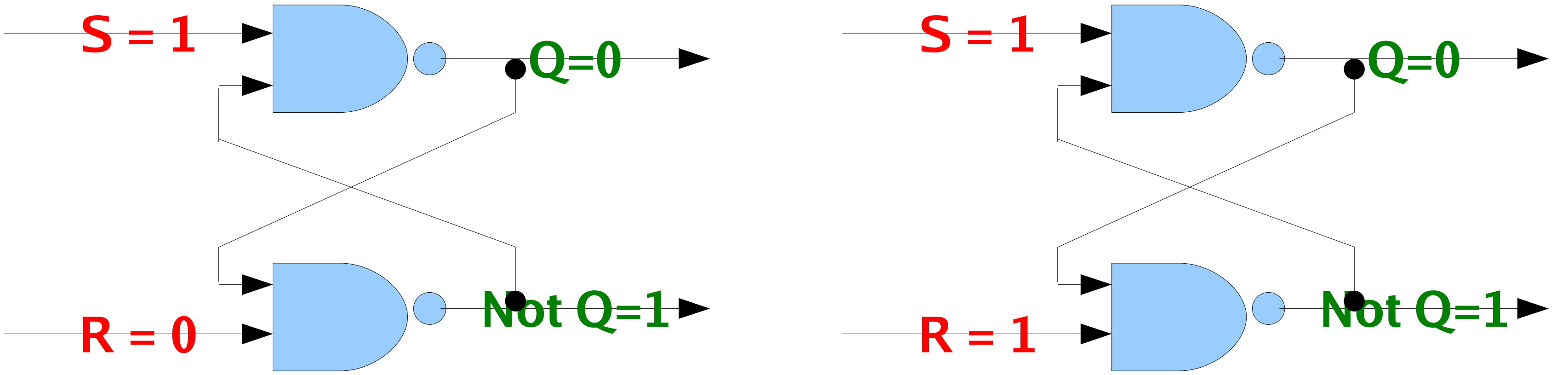
# Flip-Flop

- Now, what happens when the input CHANGES?
- Suppose we begin with S=1 and R=0:



# Flip-Flop

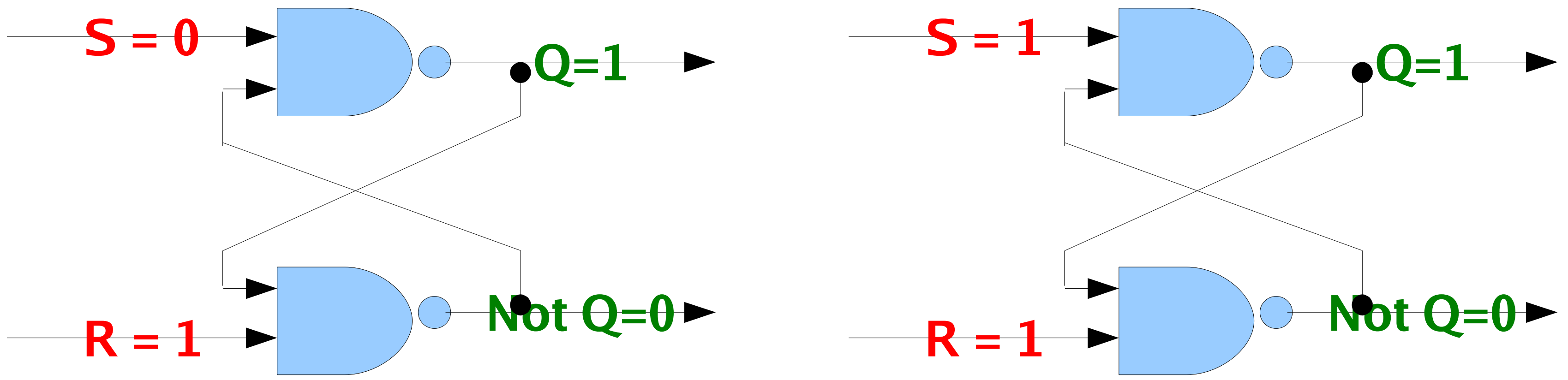
- Now, let's change to S=1 and R=1




- What does it do to the flip-flop's output?
- Answer: NOTHING!
- The lower NAND-gate's input becomes (0, 1), so it's output is still 1, so Q remains 0.

# Flip-Flop

- The same line of reasoning may be used if input changes from (S=0, R=1) to (S=1, R=1)

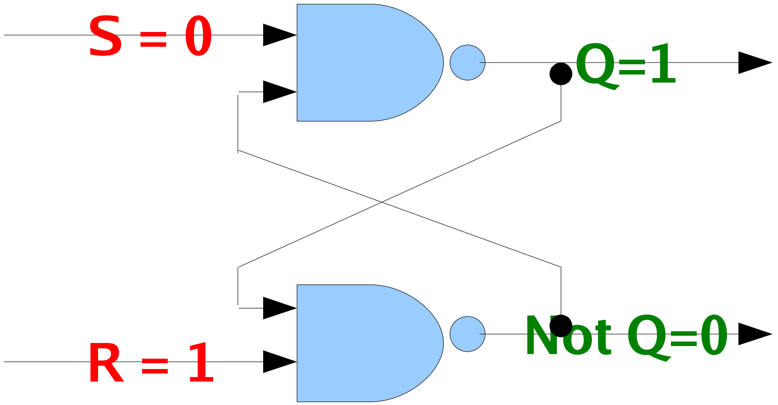



- A little weird, isn't it? The same input can produce two different outputs, depending on the previous input!

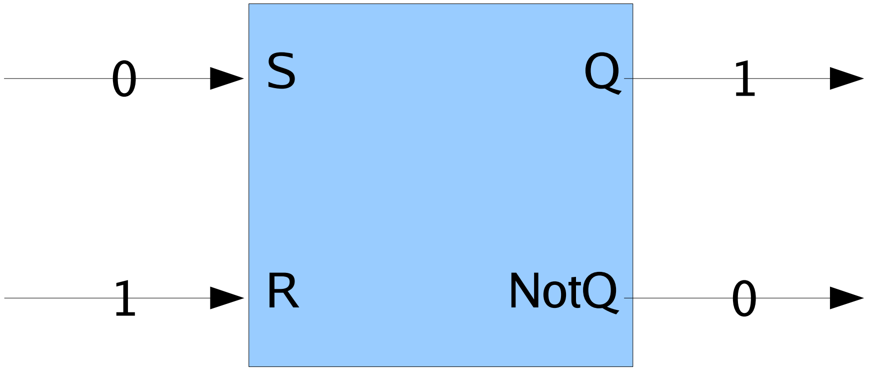
The flip-flop remembers!

# Flip-Flop

- Let's just simplify the drawing of flip-flop from this logic diagram:



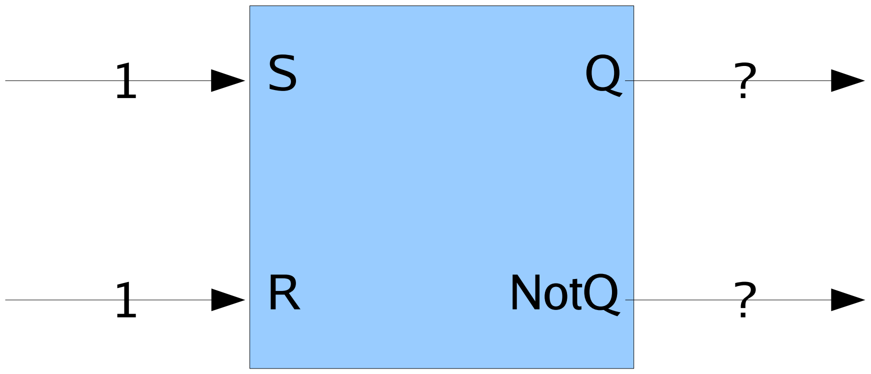

- ... to this block diagram:




Yeah, this helps us commit less drawing mistakes specially during exams!

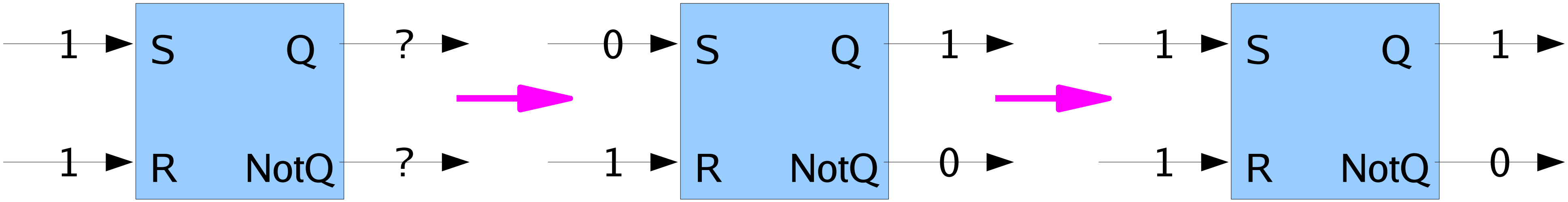
# Flip-Flop

- The way a flip-flop is used is this:
  - It begins by sitting there with a constant input of (S=1, R=1) and an output of nobody knows.



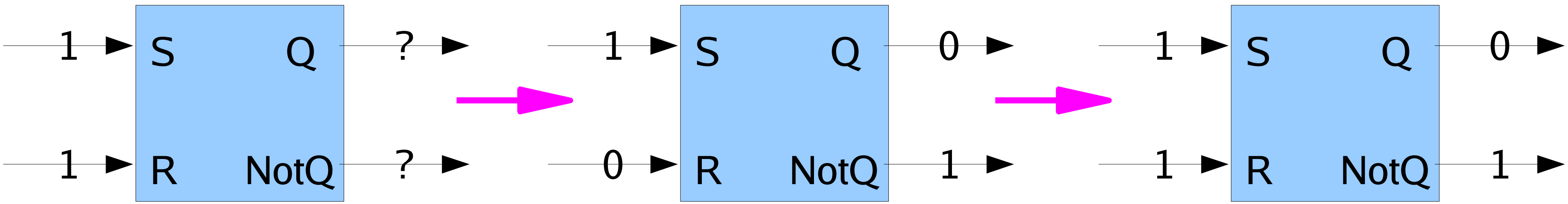
# Flip-Flop

- You SET the flip-flop (example, make Q=1) by flashing a 0 momentarily down the S-wire, and then returning it to 1:



# Flip-Flop

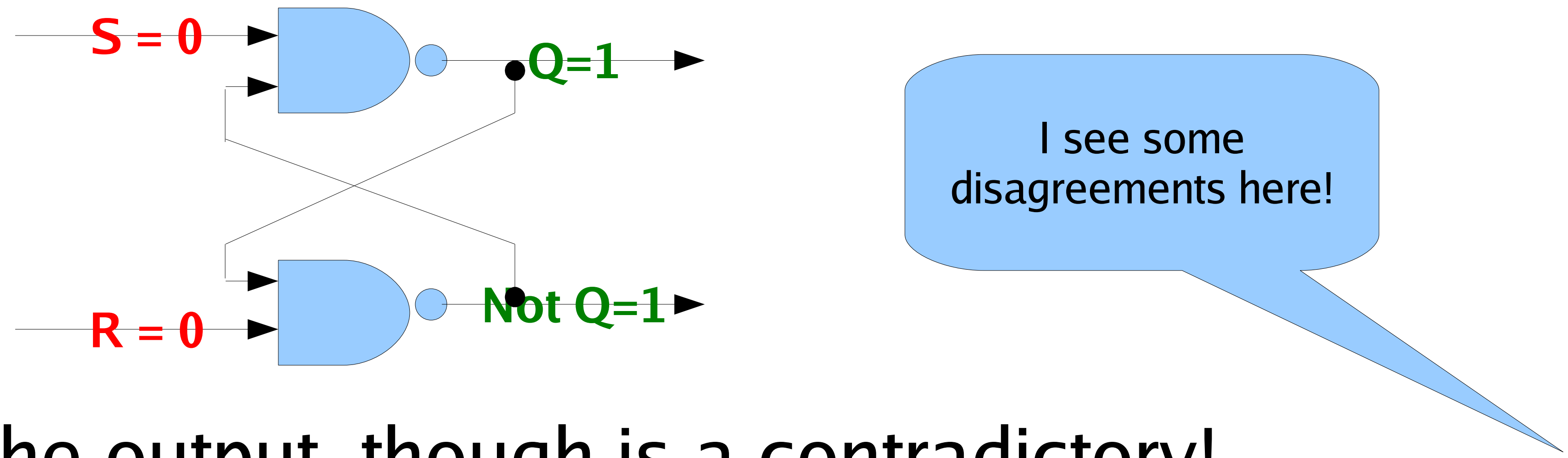
- Or you can RESET it (make Q=0) by flashing a 0 down the R-wire, then returning it to 1:




- In either case, as long as (S=1, R=1) keeps coming in, the flip-flop will maintain its output until it's changed with another incoming 0.

# Flip-Flop

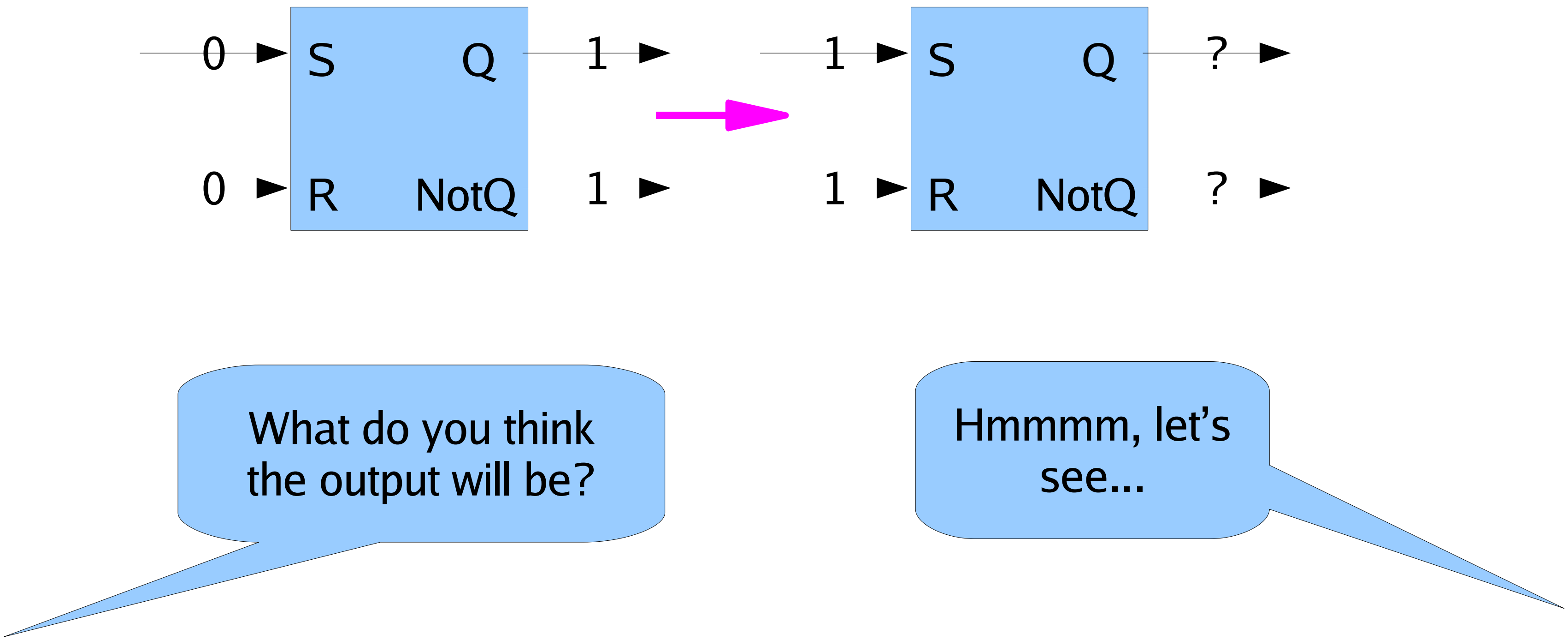
- The only input combination we haven’t checked is (R=0, S=0).
- It is easy to verify that it produces output of (Q=1, Not-Q=1)




- The output, though is a contradictory!

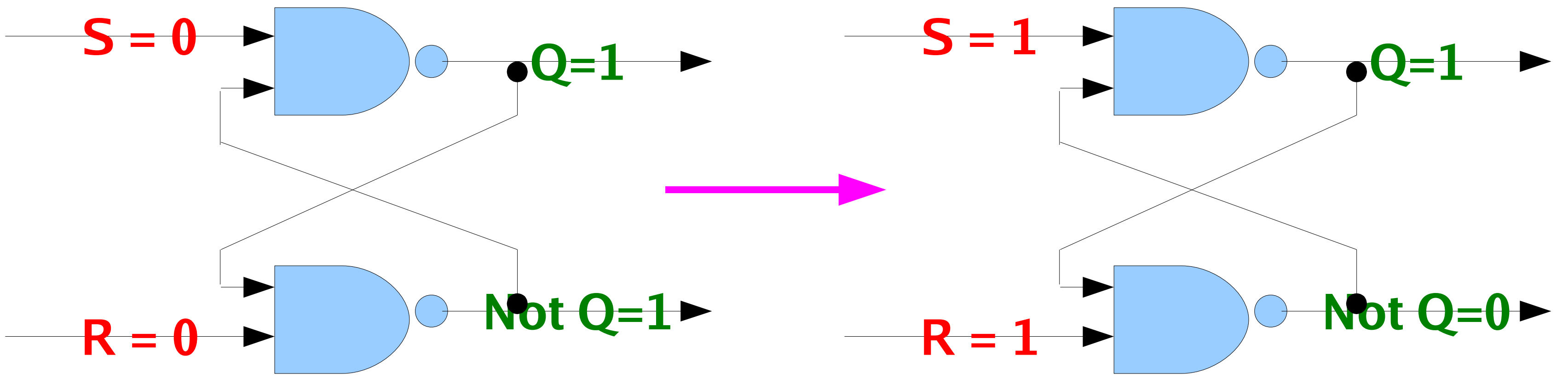
# Flip-Flop

- The output is contradictory because when Q=1, then Not-Q must be 0.
- And what will happen when the input returns to (S=1, R=1)?



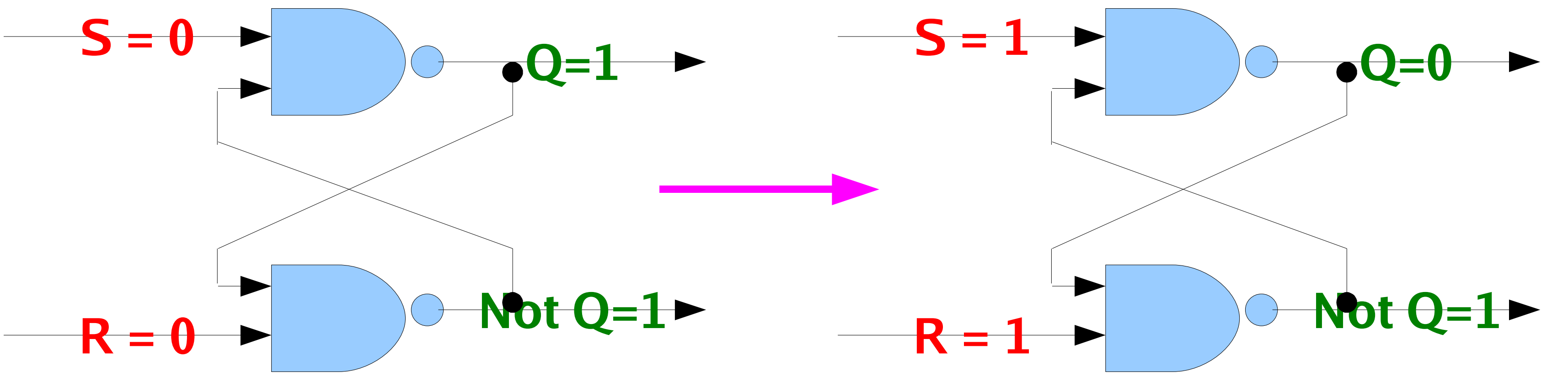
# Flip-Flop

- The answer is not so clear.
- Actually it depends on which output happens to flop first
  - If Not-Q is first to change, we get:



# Flip-Flop

– If Q flops first, we get:




- Since there is NO WAY of knowing which of these will actually happen, and we don't want our flip-flops in random and contradictory states, the input (S=0, R=0) is **DISALLOWED**!

# Flip-Flop

- We an now summarize the basic R-S flip-flop with the following I/O table:

| S | R | Q | Not-Q |
|---|---|---|---|
| 1 | 1 | No change | |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | Disallowed | |

- Flip-flop inputs are always arranged to make certain the disallowed state will not happen.

# Homework assignment

1. A NOR-gate is a shorthand of writing NOT OR. Draw the NOR-gate (hint: similar thinking is used as in the NAND-gate)
2. Derive the I/O table for the NOR-gate
3. A basic R-S flip-flop may also be made out of NOR-gates, draw a flip-flop with NOR-gates instead of NAND-gates
4. What is the output of your flip-flop with NOR-gates if (R=0, S=1)?