# Computer Systems
# Lecture 24

# Overview

- Half adder
- Full adder
- Sequential logic circuits
- Flip-flops
- Other types of flip-flops

# Doing arithmetic via logic

- Binary addition. For the addition of single-bit binary numbers: (Half-)adder.

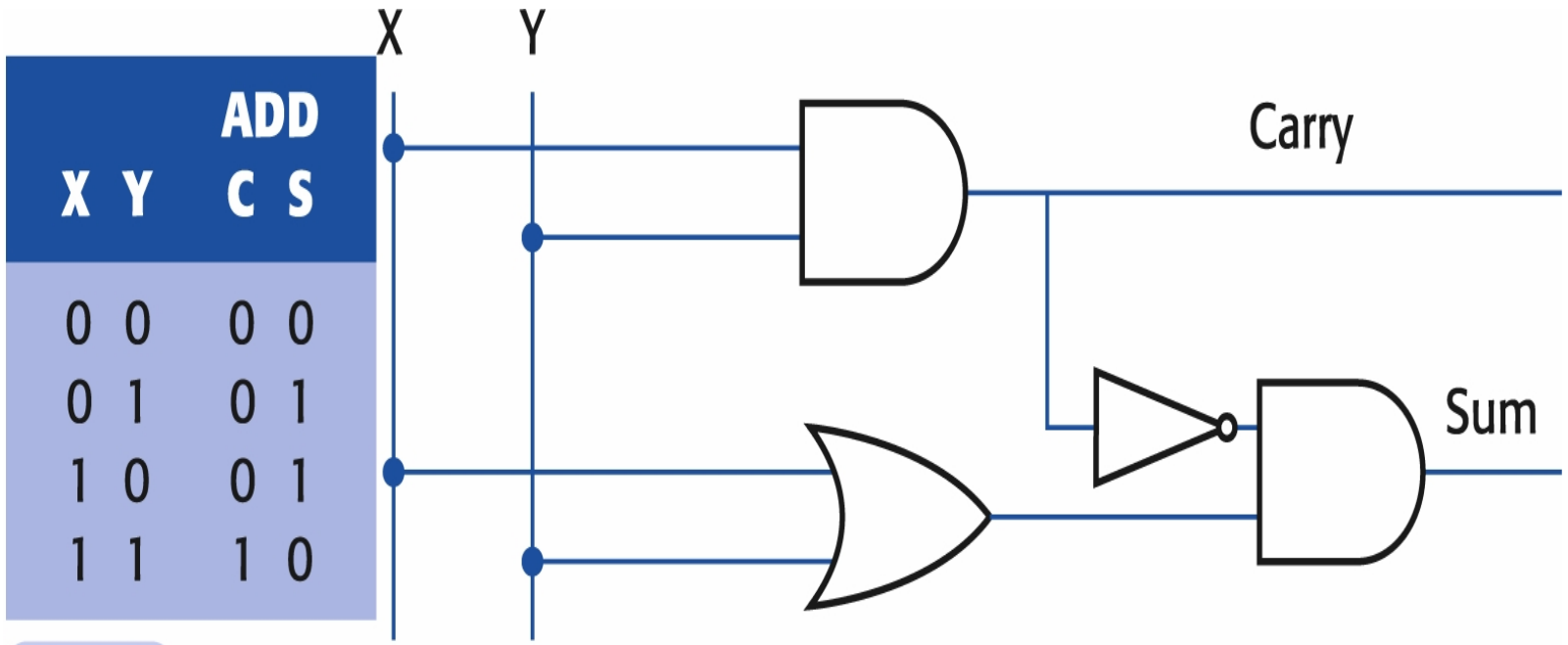



**Fig. 5.2** Adder v.1.

# Other possible designs for half adder

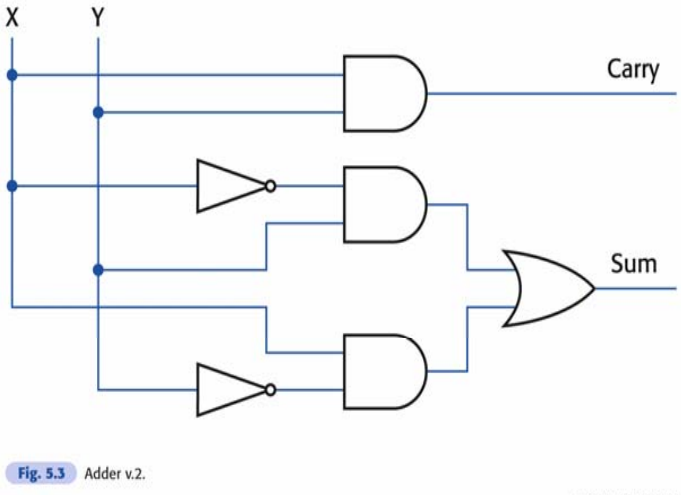



Fig. 5.3 Adder v.2.



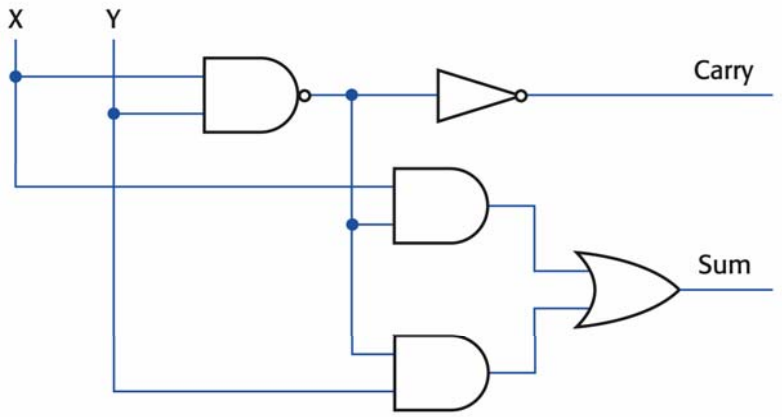



Fig. 5.4 Adder v.3.



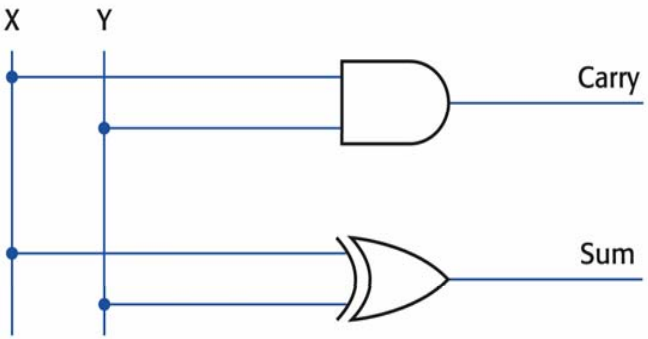



Fig. 5.5 Adder v.4, using an XOR gate for the Sum.

# Full adder

- For the addition of multi-bit binary numbers one needs to deal with carry-in from previous stage, that means the adder should have three inputs.

| X | Y | Z | C | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

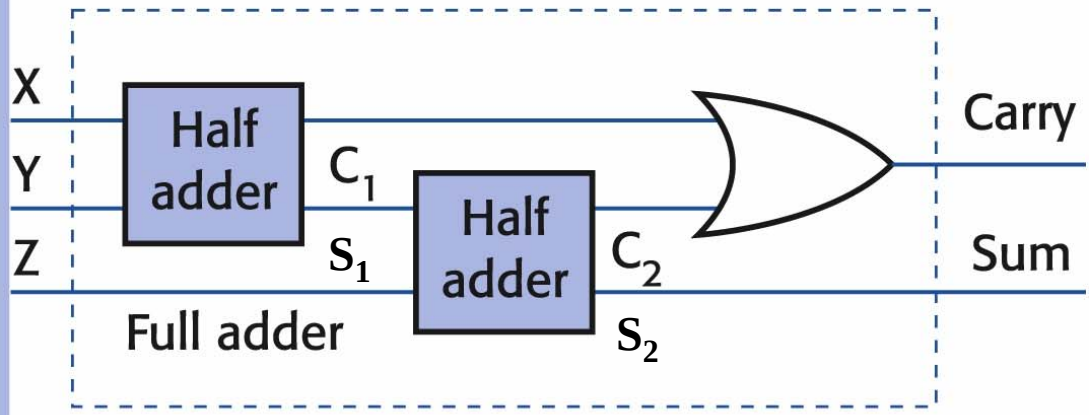



**Fig. 5.6** Full adder.

# Full adder (cont.)

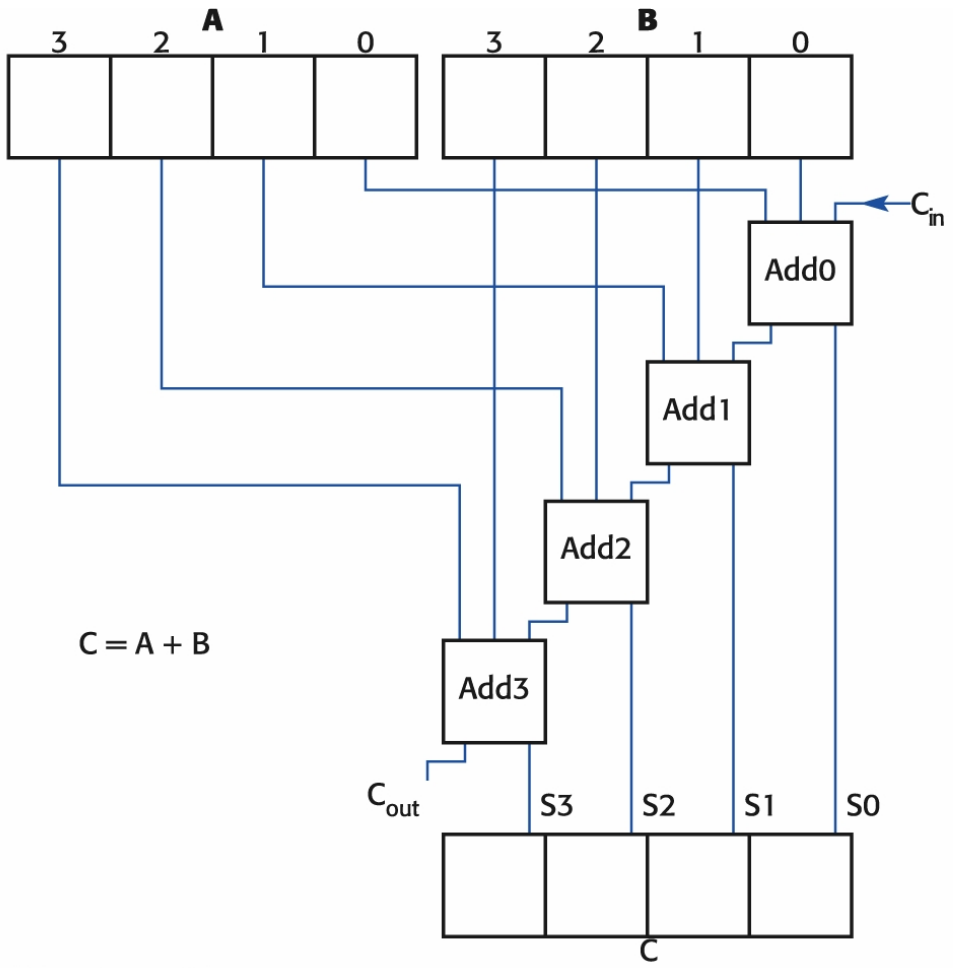



**Fig. 5.7** Paralleladdition for 4 bit numbers.

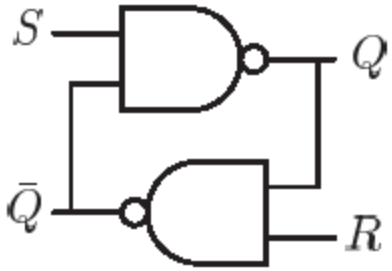
Ex. Extend 4-bit full adder into 8-bit full adder

# Sequential logic circuits

- **Combinational** (combinatorial) logic circuits.
  - In all Boolean circuits that we have seen so far, the output at any moment depends only on the input at the same moment.
  - **No memory** is there.
- **Sequential** logic circuits.
  - The output depends also on the **state** of the circuit. The state of the circuit is somehow stored within the circuit.
  - There is a **memory** inside.

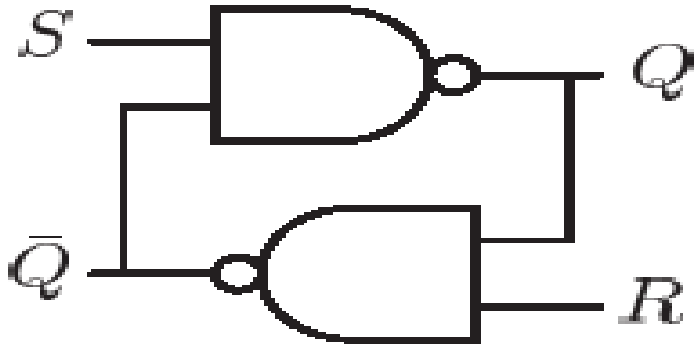
# Flip-flops, or latches

- The basic memory element of the sequential circuits is called a **flip-flop**, or **latch**.
- The simplest flip-flop is made up of two NAND gates. It is called set-reset (SR) flip-flop.




- Inputs are $S$ and $R$ while outputs are $Q$ and $\bar{Q}$.

# SR flip-flop




- Suppose that S and R are both initially set to 1. Can we determine two outputs $Q$ and $\bar{Q}$ ? **No.** Two variants are possible:

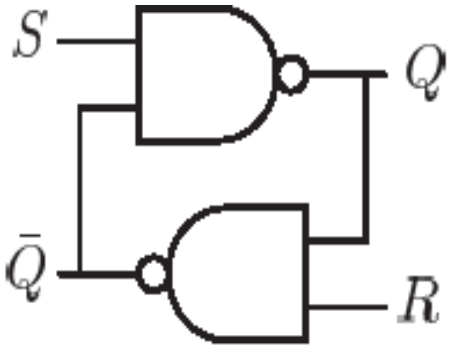
$Q = 1$ and $\bar{Q} = 0$, or

$Q = 0$ and $\bar{Q} = 1$

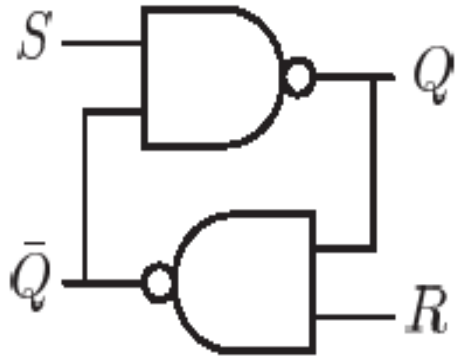
- Everything is consistent with any of these variants.

# SR flip-flop (cont.)

- If we fix both inputs to be 1, then the circuit may have one of the two possible **states**. The circuit is **stable** as long as R and S remain at 1.



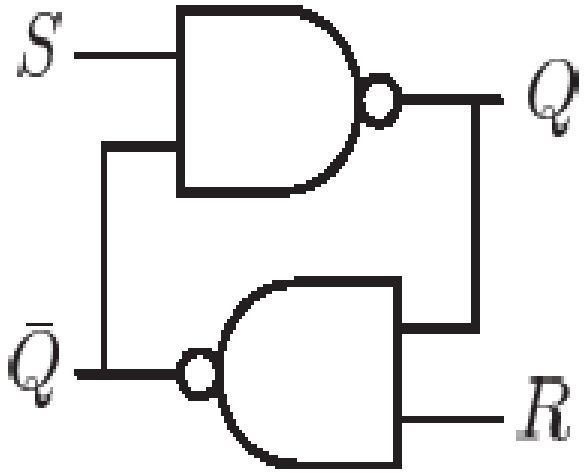
# SR flip-flop (cont.)




- Let's assume that flip-flop is in a stable state with Q=1.
- Suppose, R momentarily becomes 0. This forces changing the value of Q to 0. The circuit will stay in this state even after input R returns to 1. The flip-flop **switched** the states.
- If then input S momentarily becomes 0 the system will switch the state back.

# SR flip-flop (cont.)

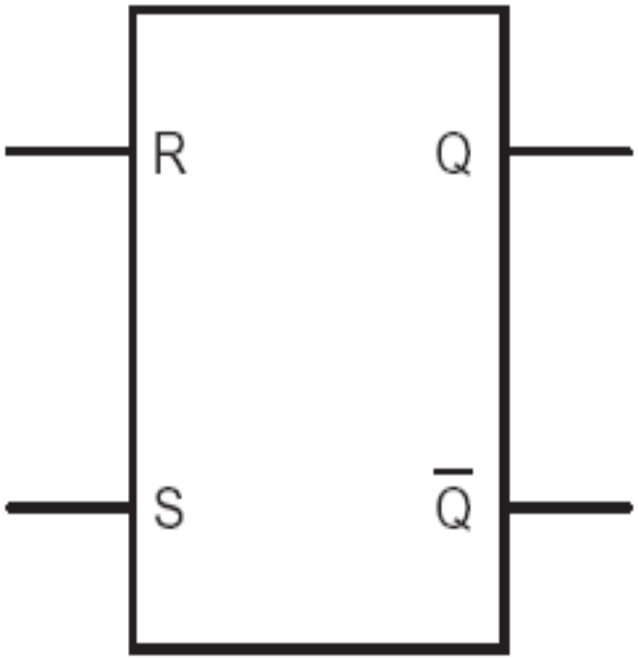
- Thus, SR flip-flop **remembers** which input was set (momentarily) to 0 last.
- One constraint: the logic surrounding this flip-flop must avoid situations where **both** R and S are 0 at the same time.



# State table of SR flip-flop

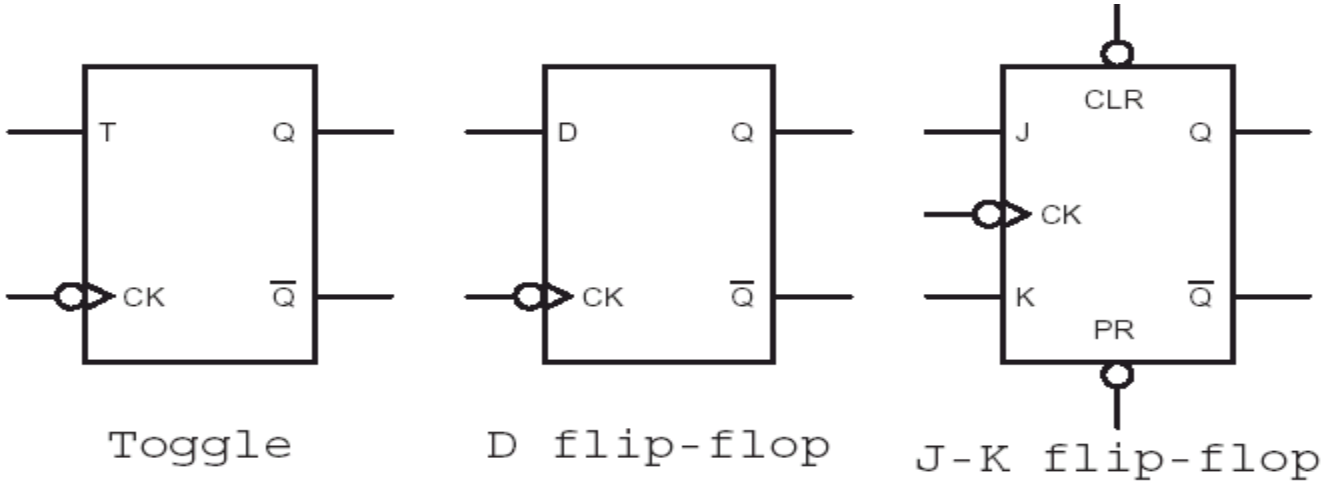
| R | S | Q |
|---|---|---|
| 0 | 0 | ? |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $Q_{prev}$ |

# SR representation



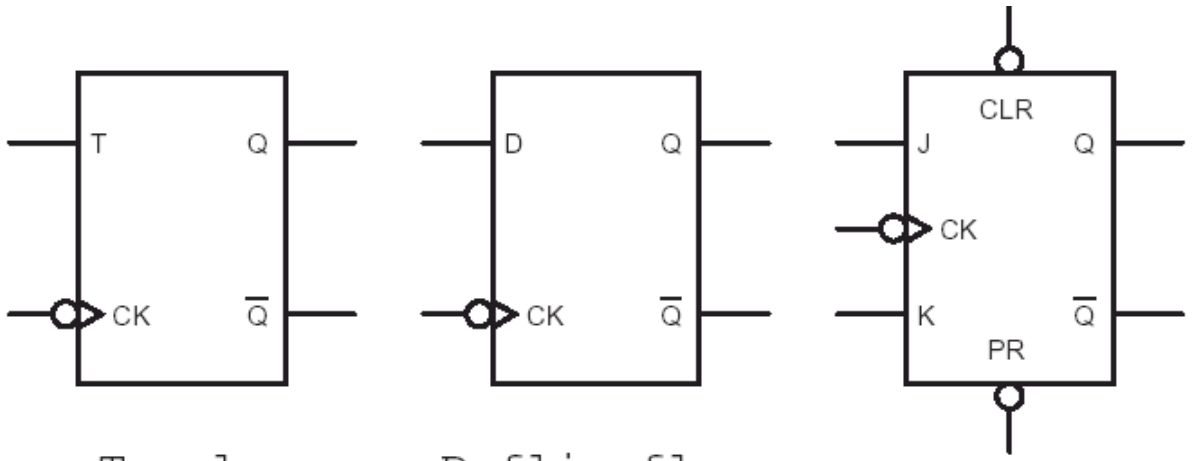

FlipFlop(SR)

# Other types of flip-flops




- All these flip-flops has an input marked for **clock**. The new output occurs when the clock is pulsed, i.e. momentarily changed from 1 to 0.
- CLR (clear) and PR (preset) inputs are used to initialise the flip-flop to known value ( 0 and 1, respectively).

# Other types of flip-flops (cont.)



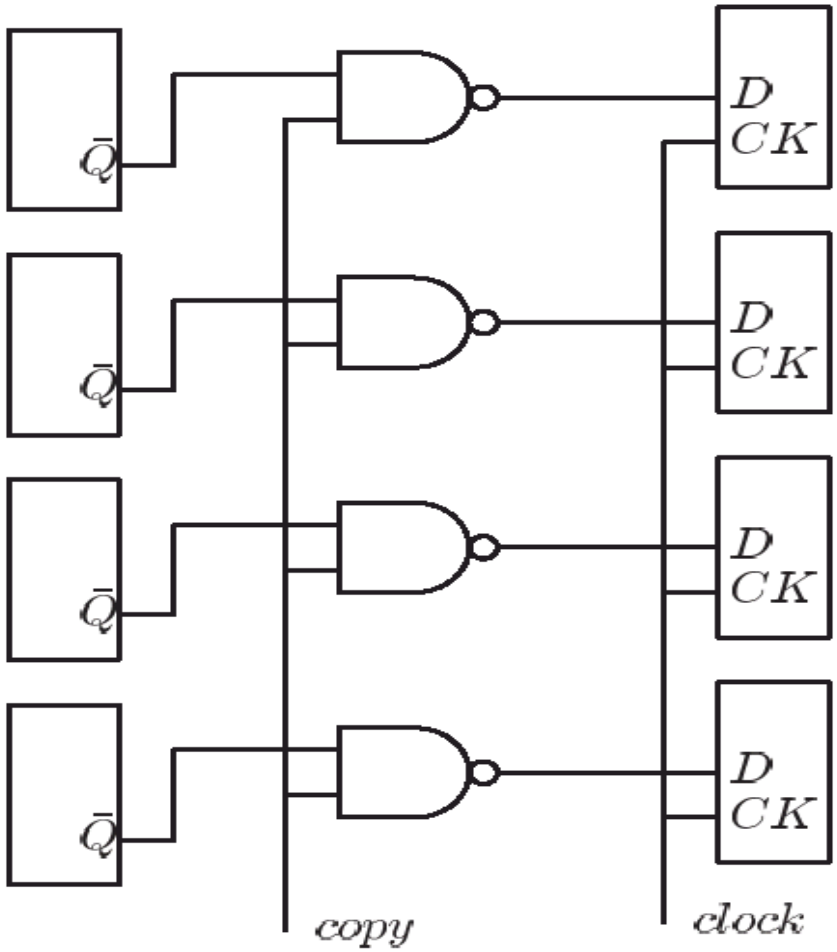

Toggle D flip-flop J-K flip-flop

| T | Q |
|---|---|
| 0 | $Q_{prev}$ |
| 1 | $\bar{Q}_{prev}$ |

| D | Q |
|---|---|
| 0 | 0 |
| 1 | 1 |

| J | K | Q |
|---|---|---|
| 0 | 0 | $Q_{prev}$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $\bar{Q}_{prev}$ |

# Use of D flip-flops - Copying data




Circuit to copy data from one register to another.

- Q. What type of flip-flop has an illegal state?

- Q. What type of flip-flop allows us to copy data?

- Q. What type of circuit has ‘memory’?

# Readings

- [Wil06] Sections 5.1, 5.2, 6.1.