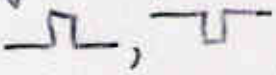
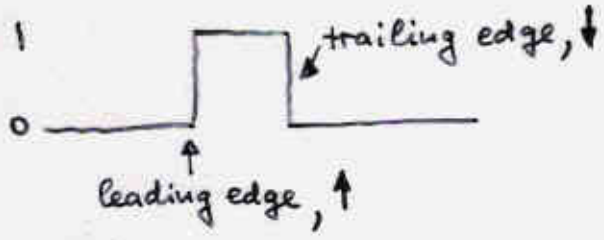
Flip-flops

- So far: <u>combinatorial</u> approach, i.e. given the state of the inputs we can predict the state of the outputs.

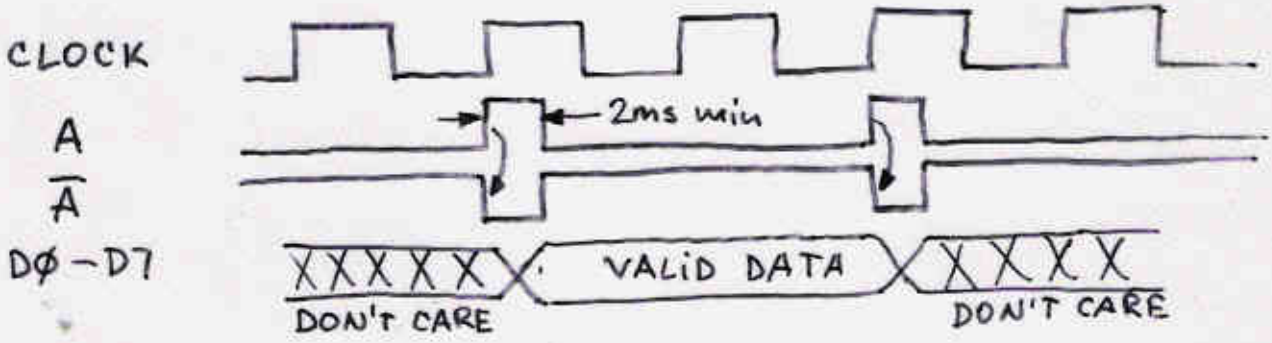
  Also possible: <u>sequential</u> approach, when the outputs may depend on the state of the inputs at <u>some earlier time</u>.

- time dependence: logical levels $\phi, 1$ → logical pulses 

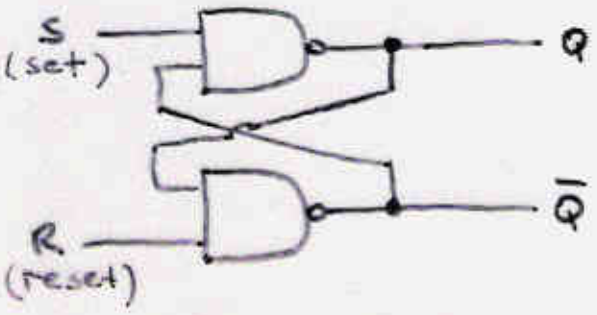
- pulse:



- truth table may now contain, $\phi, 1, \uparrow, \downarrow$, X ≡ don't care
- for full analysis, need a timing diagram



- RS flip-flop



R	S	Q	$\bar{Q}$	$\overline{(S * \bar{Q})}$ = Q	$\overline{(R * Q)}$ = $\bar{Q}$
1	1	Q_i	$\bar{Q}_i$	$\overline{(1 * \bar{Q}_i)} = \bar{\bar{Q}}_i$	$\overline{(1 * Q_i)} = \bar{Q}_i$
ϕ	1	ϕ	1	$\bar{Q} = \bar{1} = \phi$	$\overline{(\phi * Q)} = \bar{\phi} = 1$
1	ϕ	1	ϕ	$\overline{(\phi * \bar{Q})} = \bar{\phi} = 1$	$\overline{(1 * Q)} = \bar{Q} = \bar{1} = \phi$
ϕ	ϕ	?	?	$\bar{\phi} = 1$	$\bar{\phi} = 1$

problem!

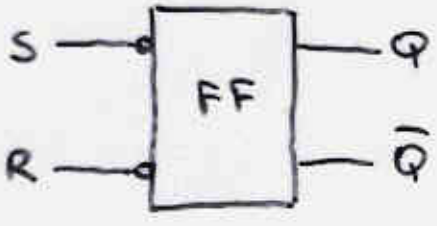
$Q_o = \phi$ or 1

- Implications ?

No matter what state (Q_i $\bar{Q}_i$) we start from...

- if R=1 and S=1 , nothing happens
- if R=1 and S goes from =1 to =Ø : $\begin{cases} Q_i \to 1 \\ \bar{Q}_i \to Ø \end{cases}$
- if S=1 and R goes from =1 to =Ø : $\begin{cases} Q_i \to Ø \\ \bar{Q}_i \to 1 \end{cases}$
- if both S & R go from =1 to =Ø : indeterminate state

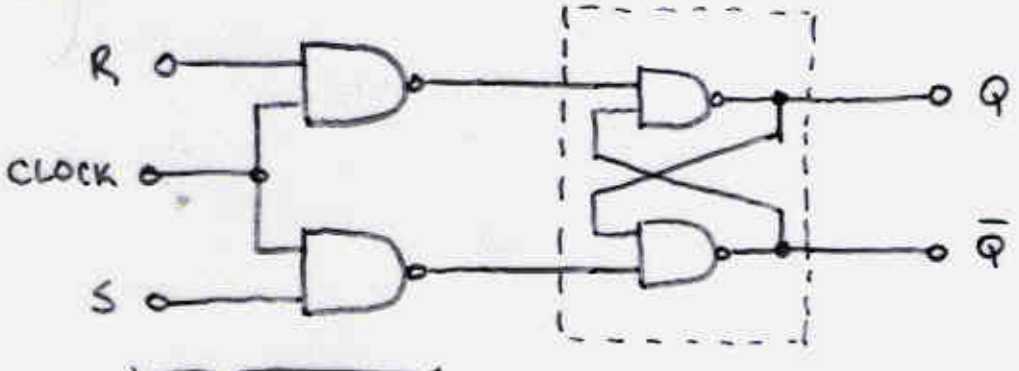
Note: both R & S detect ↓-edges, i.e. they should be properly designated as $\bar{R}$ and $\bar{S}$ (inverting) inputs

- 

R	S	Q	$\bar{Q}$
H	H	Q_i	$\bar{Q}_i$
⌐	H	L	H
H	⌐	H	L
⌐	⌐	indeterminate	

H = high, 1, 5V
L = low, Ø, ØV

- add a front-end, clocked input



CLOCKED RS FF

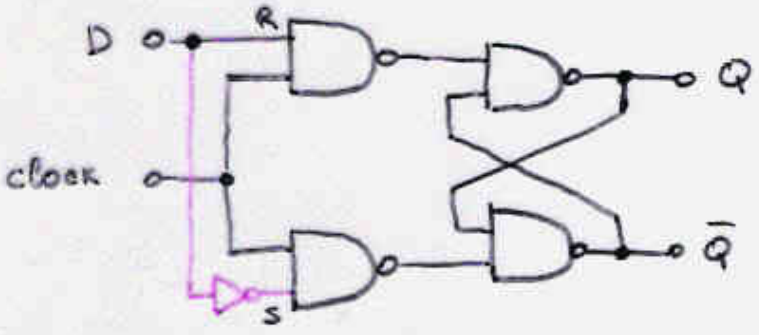
R	S	C	Q	$\bar{Q}$
X	X	L	Q_i	$\bar{Q}_i$
L	L	⌐	Q_i	$\bar{Q}_i$
H	L	⌐	H	L
L	H	⌐	L	H
H	H	⌐	?	?

ignores any signals from R,S unless CLOCK=1
⇒ controls time at which circuit switches state
This does not solve the problems, but good for sequential operation

Note: as CLOCK goes low, FF "memorizes" its state

• D Flip-flop [D = data]

- To eliminate the problem of an indeterminate state of a FF, need to disallow R = 1 and S = 1 state:



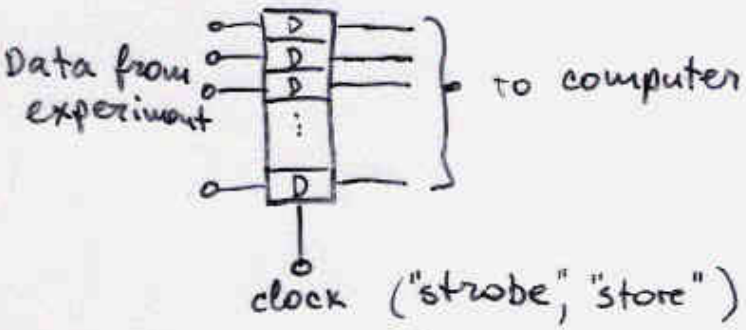
R = 1, S = 1 } disallowed
R = ϕ, S = ϕ }

⇒

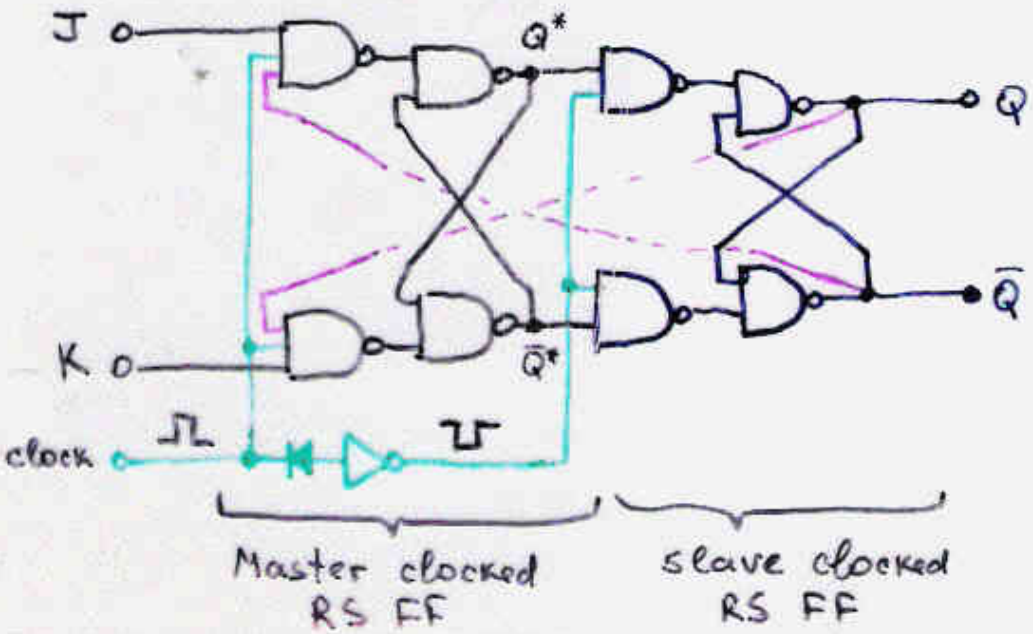
D	C	Q	$\bar{Q}$
X	L	Q_i	$\bar{Q}_i$
H	⊓	H	L
L	⊓	L	H

- After the clock goes low, this FF will remember what D was when the clock was high. → a data latch

- Can be used as a data register: preserve the state of the data at a certain time for later processing.



• JK Flip-flop

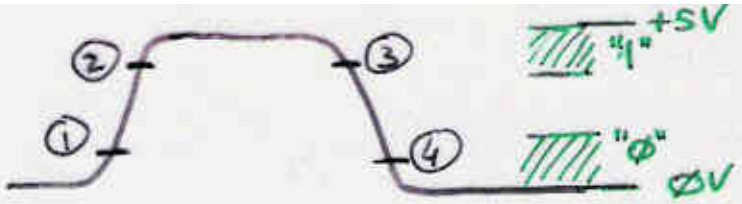


J	K	C	Q^*	$\bar{Q}^*$		Q	$\bar{Q}$
X	X	L	— no change —				
L	L	⊓	— no change —				
H	L	⊓	H	L	⊓	H	L
L	H	⊓	L	H	⊓	L	H
H	H	⊓	....			$\bar{Q}_i$	Q_i

toggle!

state @ current clock pulse ⊓ | state after clock pulse ⊓

Clock pulse:
(when J=H, K=H)



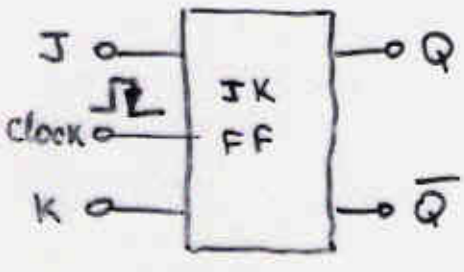
① Because of the diode, the inverted clock at slave FF has fallen enough to isolate the slave from the master. From now on, any change in the master does not get passed to the slave

② Master's clock has entered high, i.e. master FF is completely enabled and will latch in $(Q_i \bar{Q}_i)$ from the slave: $Q^* = \bar{Q}_i$, $\bar{Q}^* = Q_i$ ← note the sign!

③ Master's clock is out of high ⇒ master RS FF is isolated from the influence of its inputs

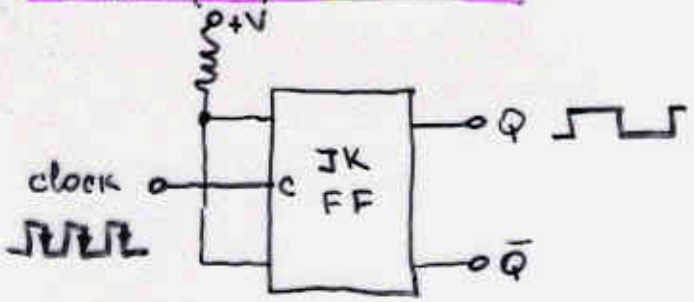
④ Slave RS FF is enabled and clocks in the new state of its inputs: $Q = Q^* = \bar{Q}_i$, $\bar{Q} = \bar{Q}^* = Q_i$ } toggle!

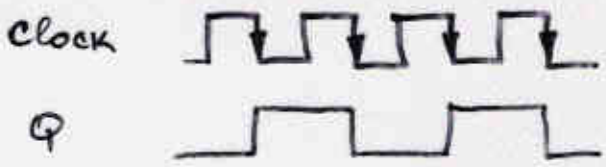
Features:
- diode causes early response in one FF vs the other
- the change of the output for the whole JK FF occurs at the <u>trailing edge</u> of the clock



J	K	C	Q	$\bar{Q}$	
X	X	L	Q_0	$\bar{Q}_0$	
L	L	H	Q_0	$\bar{Q}_0$	
H	L	H	H	L	} outputs track inputs
L	H	H	L	H	
H	H	↓	$\bar{Q}_0$	Q_0	– toggle

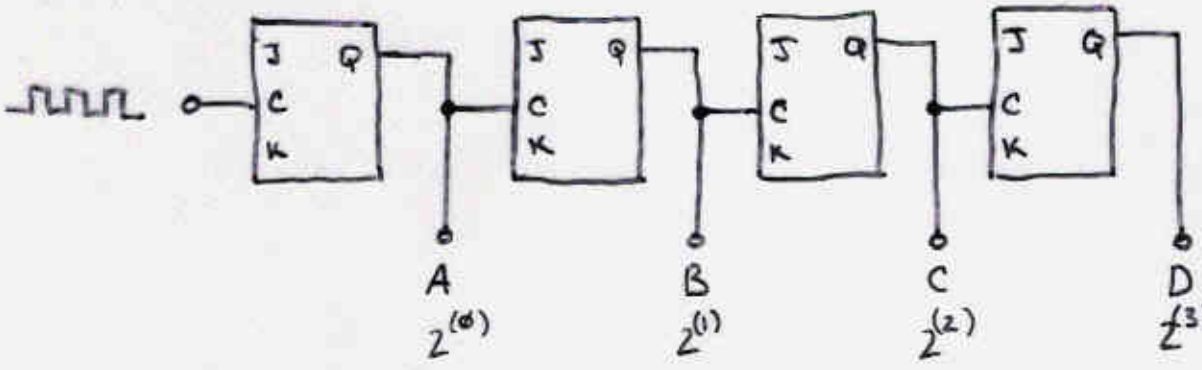
Flip-flop circuits





⇒ Q changes @ $\frac{1}{2}$ the clock frequency

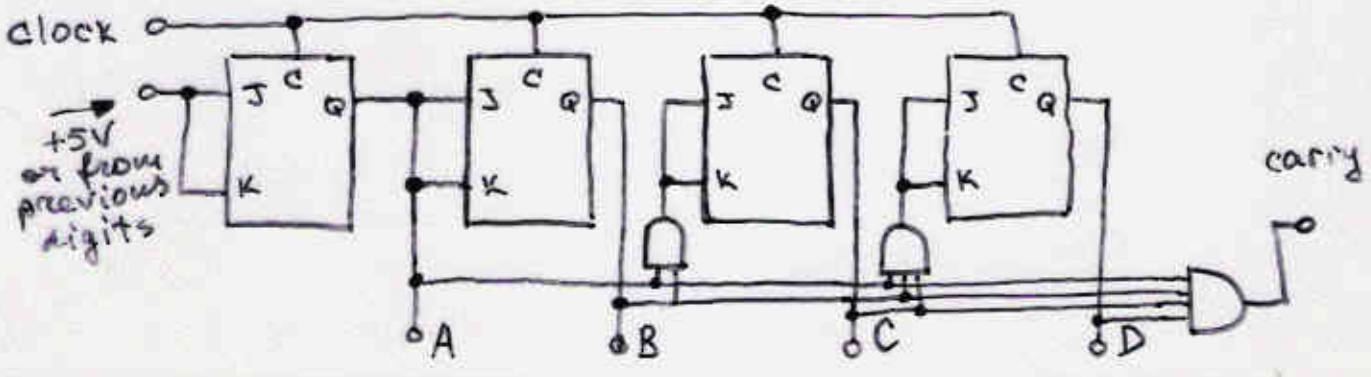
- Binary counters



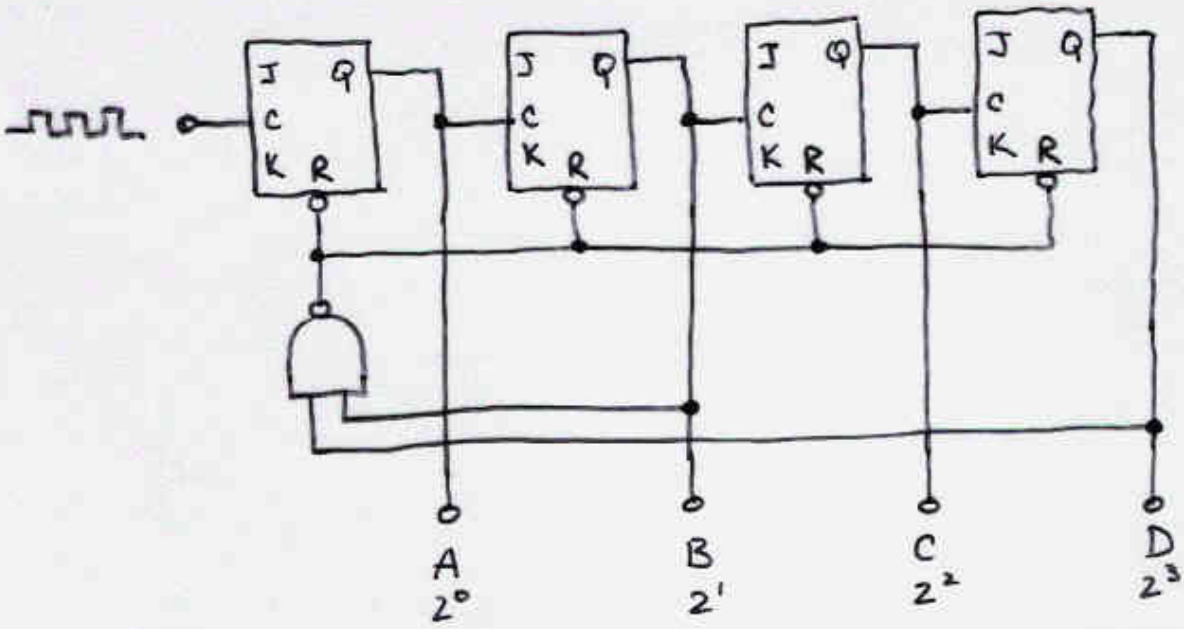
count = $= (DCBA)_2$

Asynchronous counter : e.g. when $(DCBA) = \emptyset 111$ the next clock pulse will propagate across each FF in ~20ns, i.e. the final state of $(DCBA) = 1\emptyset\emptyset\emptyset$ will not be reached until $4 \times 20 = 80$ns later, a propagation delay

Synchronous counters are also possible : the same clock signal gets routed to all FF's, and the JK inputs are used to inhibit unwanted toggles.



• modulo-n counters by direct reset technique



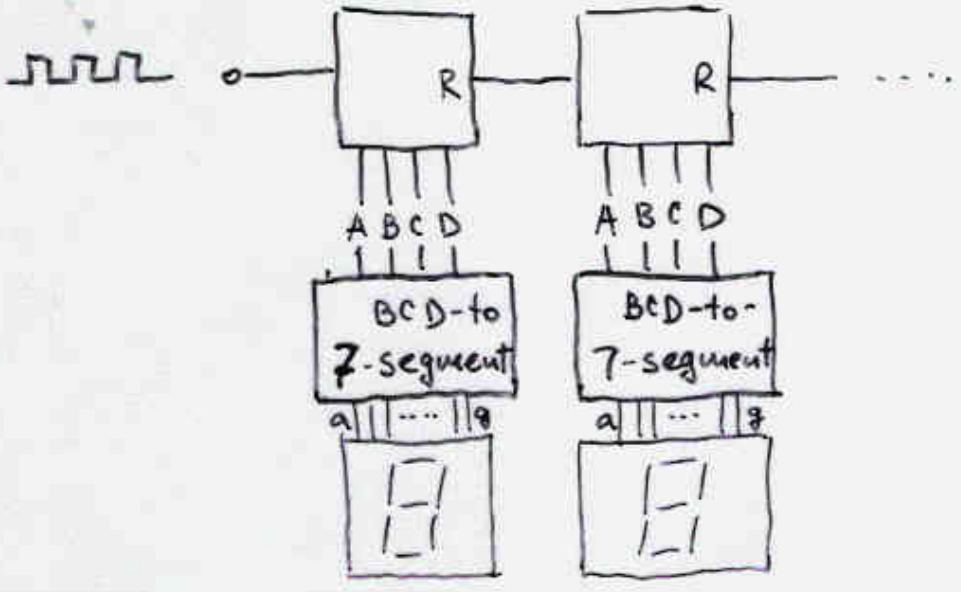
R = reset to
$\begin{cases} Q = \emptyset \\ \overline{Q} = 1 \end{cases}$

(DCBA) = count. When $DCBA = 1010_2 = 10_{10}$ → reset to ∅

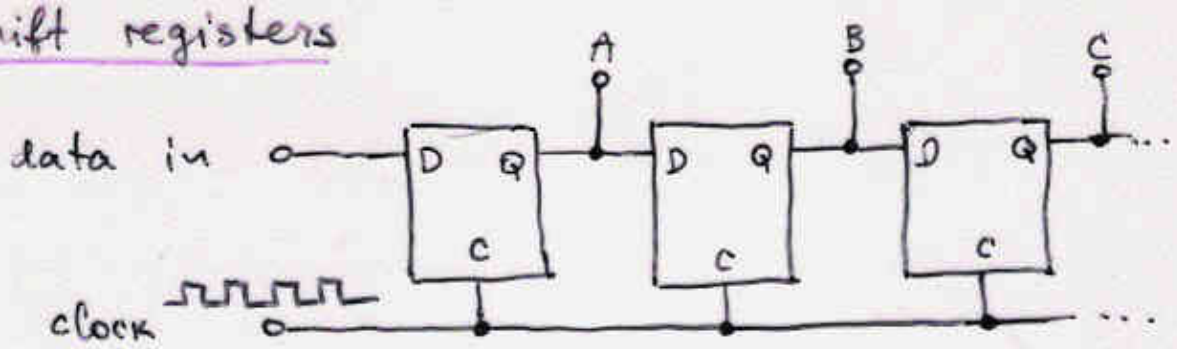
i.e. the counter will count ∅, 1, 2, 3 .... 8, 9, ∅, 1 ....

⇒ a modulo-10 counter (count-by-10 counter)

- other n possible, by an appropriate combination of ANDed bits.
- route the Reset signal as the clock into the next significant digit ⇒ decimal (BCD) counter.



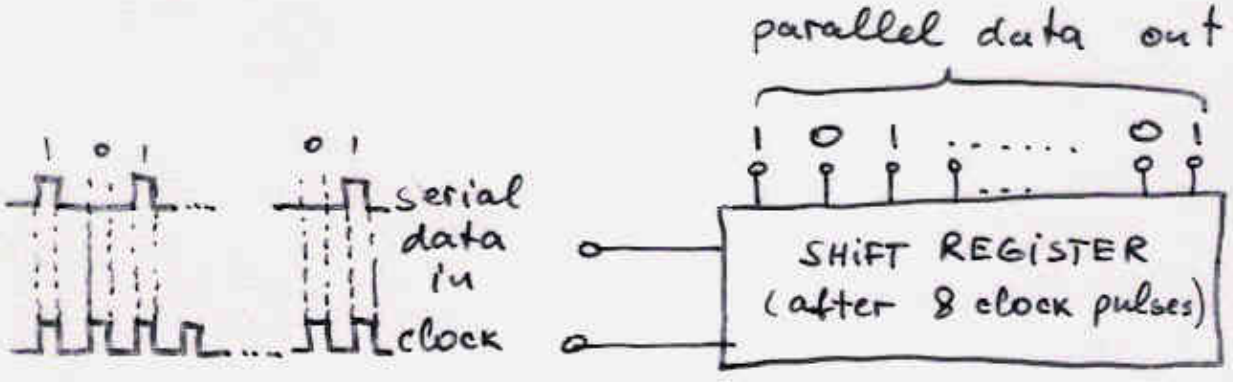
- shift registers



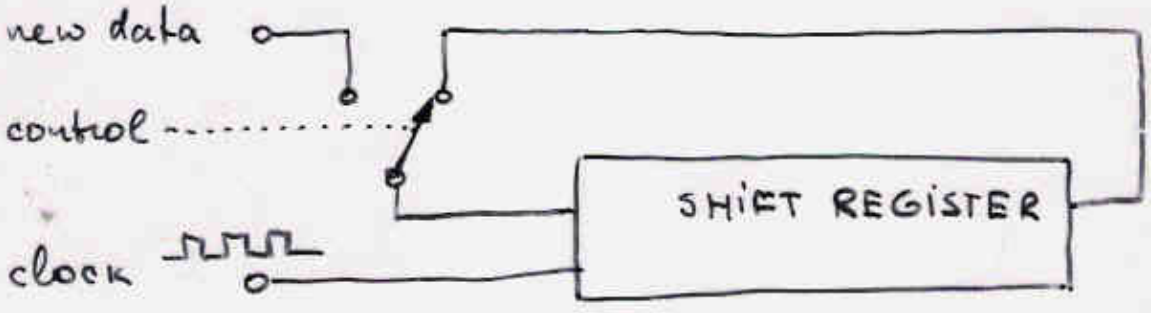
- at each stage: the clock pulse locks in the value of the output from the previous stage

⇒ the data steps to the right! — shift register

- use to decode serial data transmission:



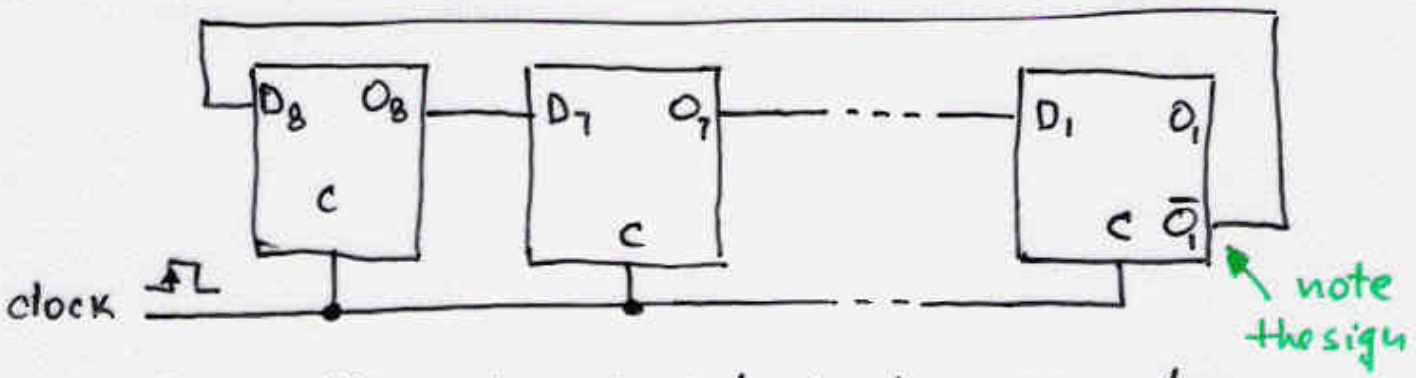
- use as FIFO (first-in, first-out) buffer



this is a recirculating SR: by controlling the switch we can either take in new data, or keep recycling what is already in the SR.

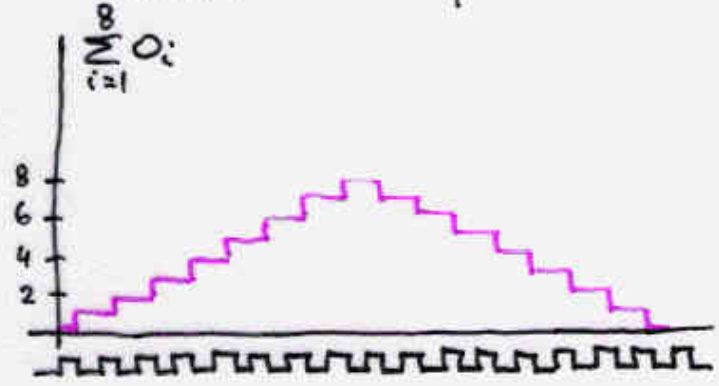
⇒ a form of data storage (memory)

Ex The Johnson counter



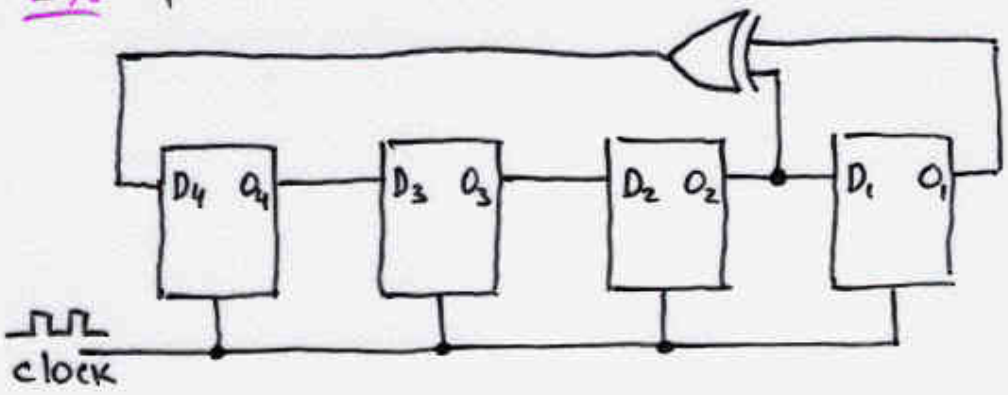
also called the twisted ring counter

After....	O_8	O_7	O_6	O_5	O_4	O_3	O_2	O_1
initial state	0	0	0	0	0	0	0	0
1 pulse	1	0	0	0	0	0	0	0
2 pulses	1	1	0	0	0	0	0	0
....	.....							
7 pulses	1	1	1	1	1	1	1	0
8 pulses	1	1	1	1	1	1	1	1
9 pulses	0	1	1	1	1	1	1	1
10 pulses	0	0	1	1	1	1	1	1
.....	.....							
14 pulses	0	0	0	0	0	0	1	1
15 pulses	0	0	0	0	0	0	0	1
16 pulses	0	0	0	0	0	0	0	0
17 pulses	1	0	0	0	0	0	0	0
.....								



← A triangular digital "wave"

Ex pseudo-random number generator



	Q_4	Q_3	Q_2	Q_1
"seed"	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	1	0	0	1
4	1	1	0	0
5	0	1	1	0
6	1	0	1	1
7	0	1	0	1
8	1	0	1	0
9	1	1	0	1
10	1	1	1	0
11	1	1	1	1
12	0	1	1	1
13	0	0	1	1
14	0	0	0	1
15	1	0	0	0

no repeats !

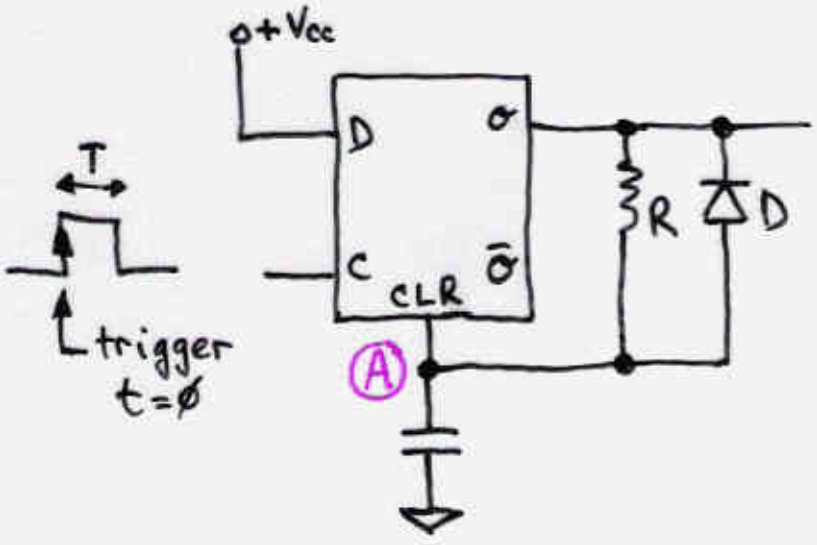
$N=4 \Rightarrow 2^N-1 = 15$ different N-bit numbers are generated, no repetitions

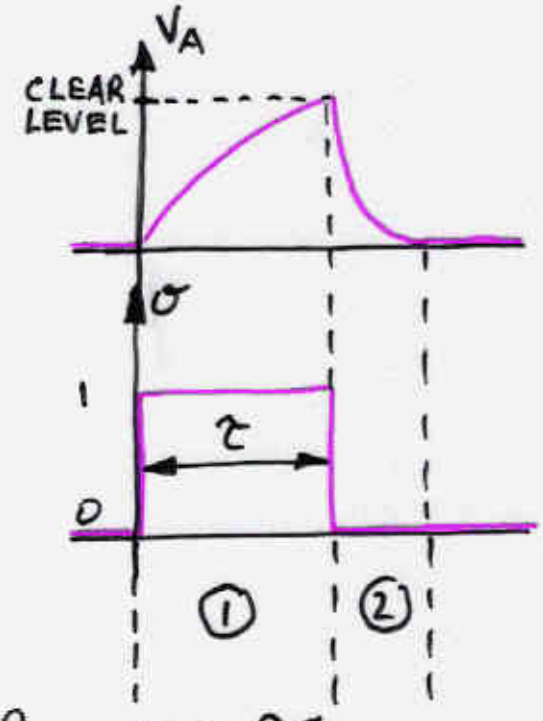
N	# states	Cycle time @ 100kHz i.e. $T = 10\mu s$
4	15	$150\mu s$
16	65,535	$655.35ms$
64	1.845×10^{19}	$1.845\times10^{14}s \simeq 7\times10^{6}$ years

Eg. MM5837 : 17-bit digital noise source
MM2533 : 1024-bit shift register

Digital clocks and monostable multivibrators (one-shots)

- a D-flip-flop one-shot





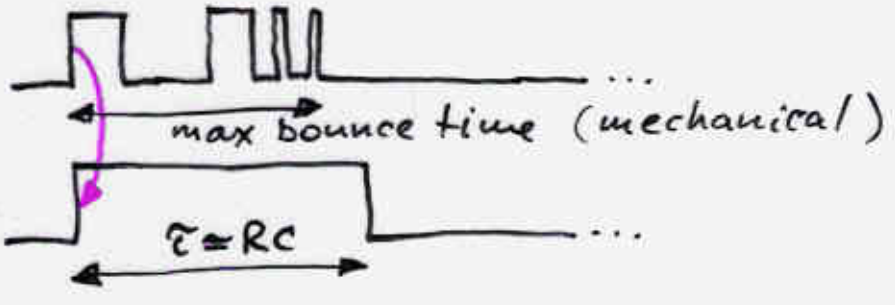
① Charge up C through R, $\tau \sim RC$

② Discharge C through D, fast(or)

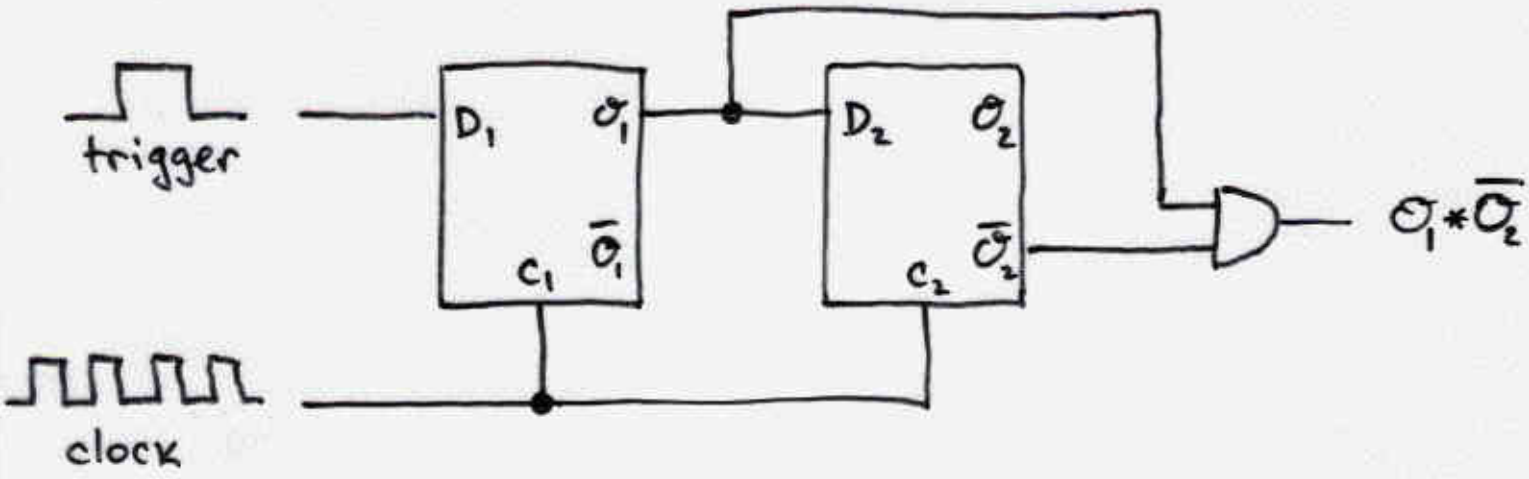
Note: $\tau \sim RC$, asynchronous of the clock period, T

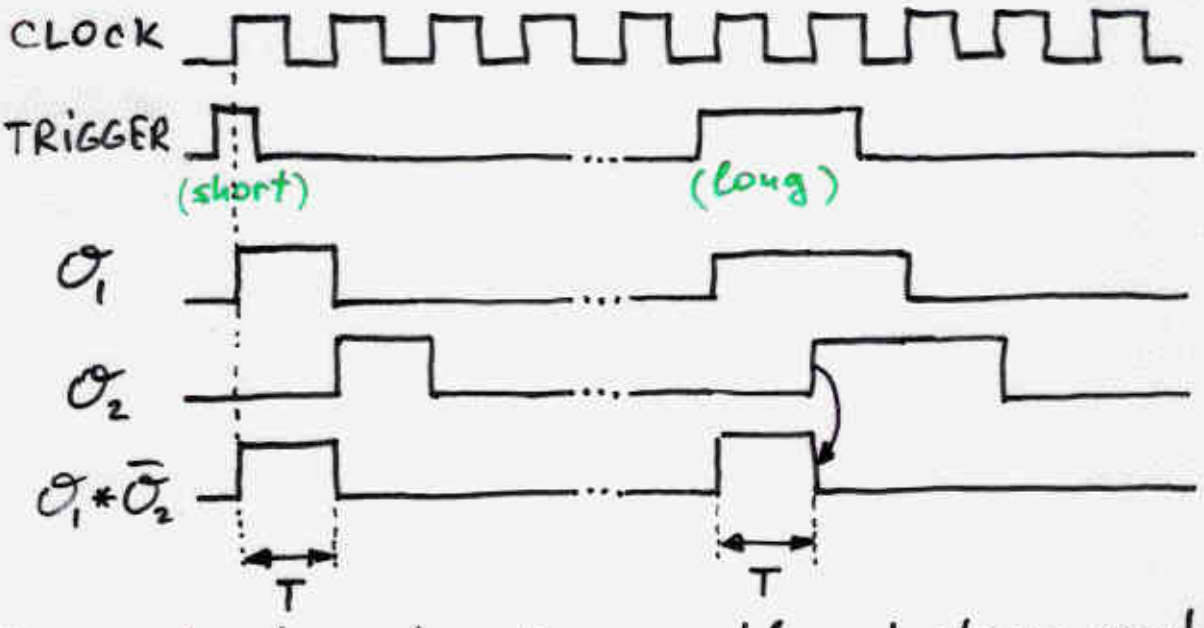
monostable: one stable state (ø), one temporary state (1)

Ex debounce a mechanical switch



- a synchronous one-shot



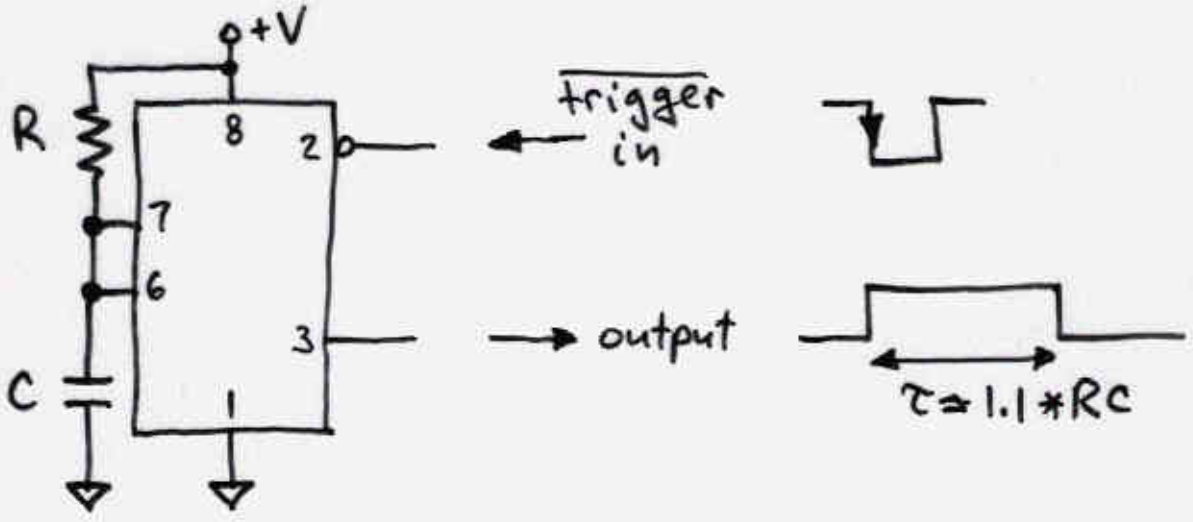


- the output pulse is <u>exactly</u> 1 clock cycle long, whether the trigger pulse is short or long (longer than the clock cycle!)

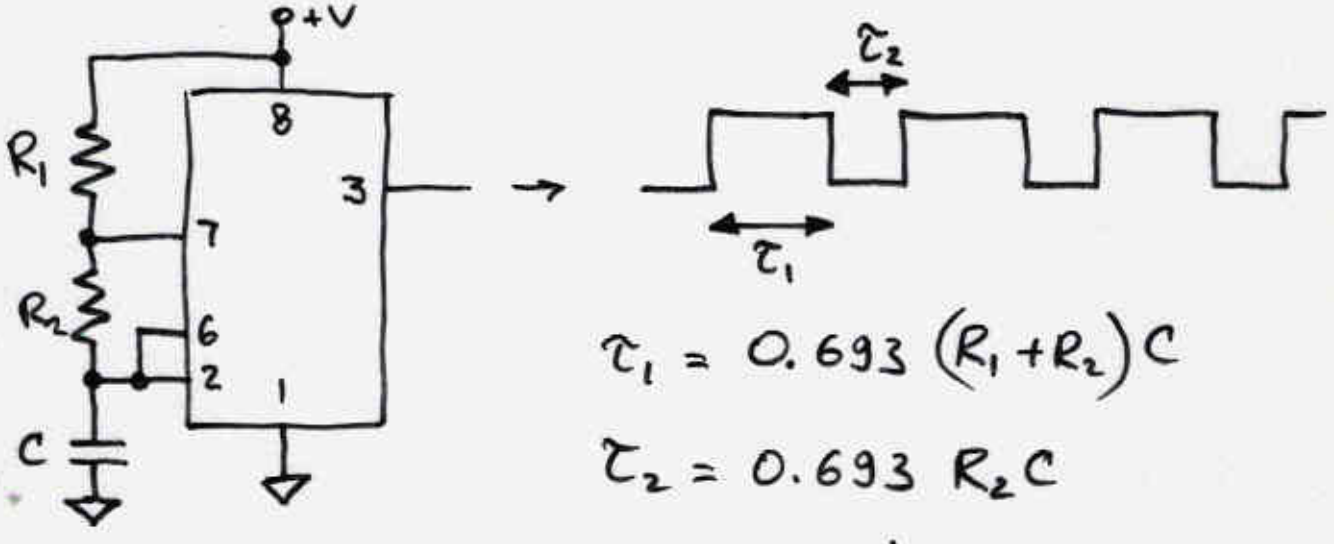
- in-phase with the clock, even if the clock frequency changes

- combination digital / analog (linear) clock circuits

Ex a 555, monostable (one-shot) operation



Ex a 555, astable (oscillator) operation



$$\tau_1 = 0.693\,(R_1 + R_2)\,C$$

$$\tau_2 = 0.693\,R_2\,C$$

$$f = \frac{1}{0.693\,(R_1 + 2R_2)\,C}$$

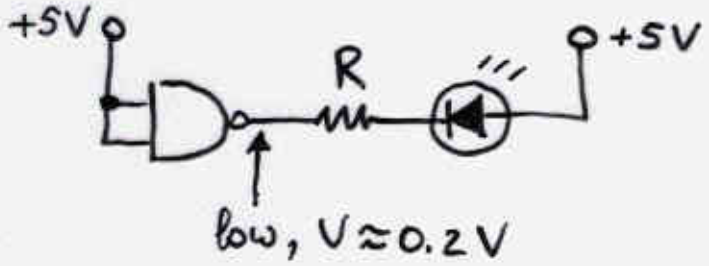
Note: the duty cycle $= \frac{t_{on}}{t_{on} + t_{off}} = \frac{R_1 + R_2}{R_1 + 2R_2} \neq 50\%$

i.e. cannot get a symmetrical wave.

As $R_1 << R_2$, D.C. $\rightarrow 50\%$

Three-state logic

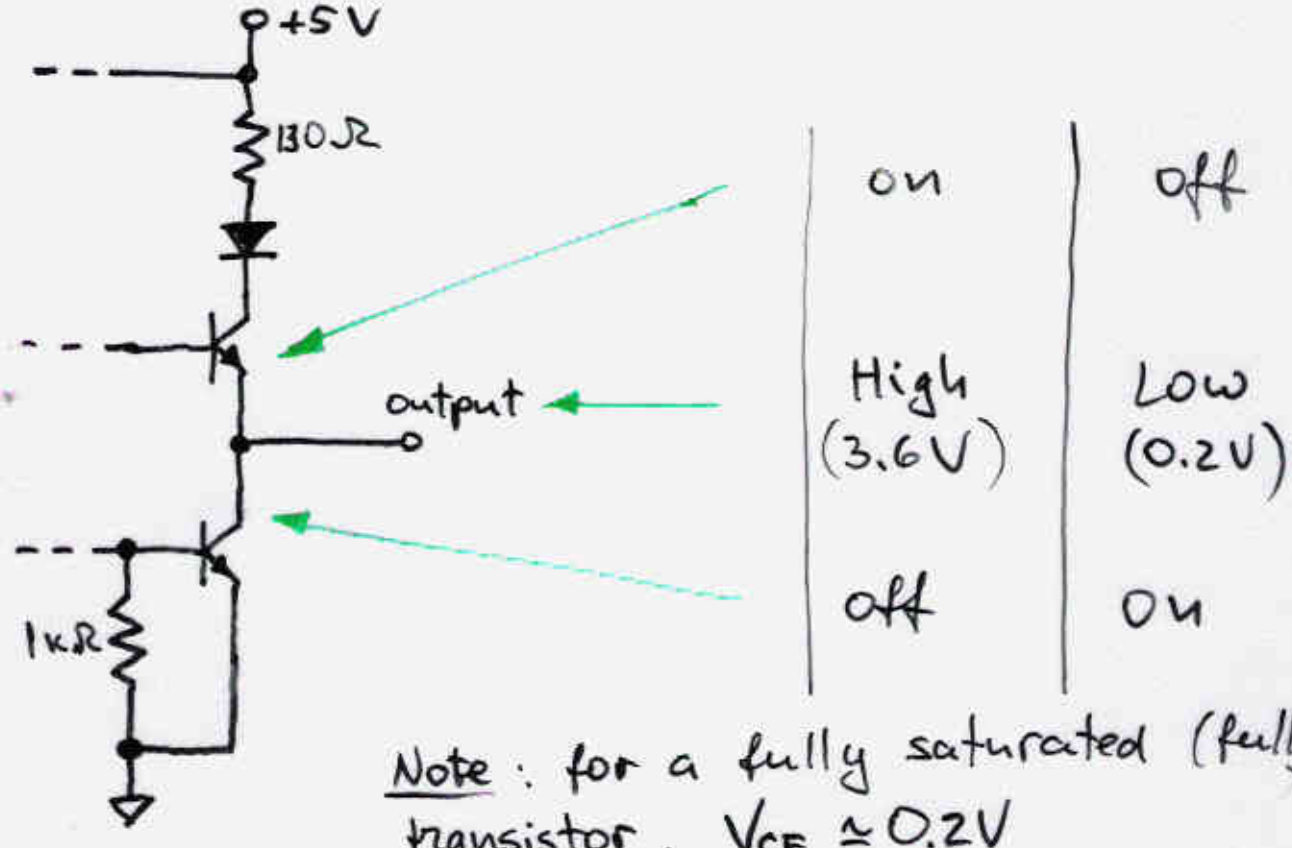
- A typical TTL gate can <u>sink</u> up to 16mA of current (8mA for LS TTL)



E.g. for a LED with a forward $\Delta V = 2V$:

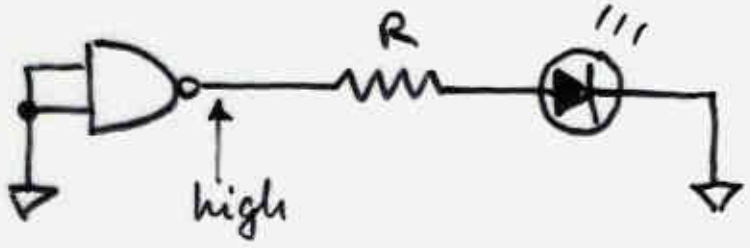
$$R = \frac{5V - 2V - 0.2V}{16 \times 10^{-3} A} = \frac{2.8}{16 \times 10^{-3}} \Omega = 175 \Omega$$

- "totem pole" output of a typical TTL gate



<u>Note</u>: for a fully saturated (fully ON) transistor, $V_{CE} \simeq 0.2V$

- If the same TTL gate is used to <u>source</u>, up to 16 mA in a high state, i.e:



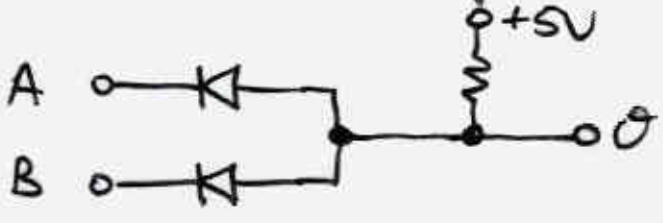
$\Delta V_{LED} = 2V$

$$\frac{5V - 0.7V - 0.2V - 2V}{R + 130\Omega} \leq 16\,mA$$

<u>Note 1</u>: it is better to <u>sink</u> than to <u>source</u>

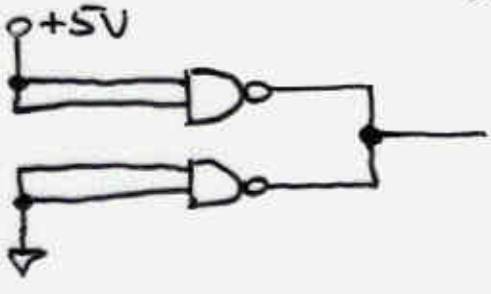
<u>Note 2</u>: cannot get the maximum brightness of an LED ($I_{LED} \simeq 30mA$)
⇒ use driver IC's or transistors

- Recall: diode logic



A	B	O
0V	0V	0.7V
0V	5V	0.7V
5V	0V	0.7V
5V	5V	5V

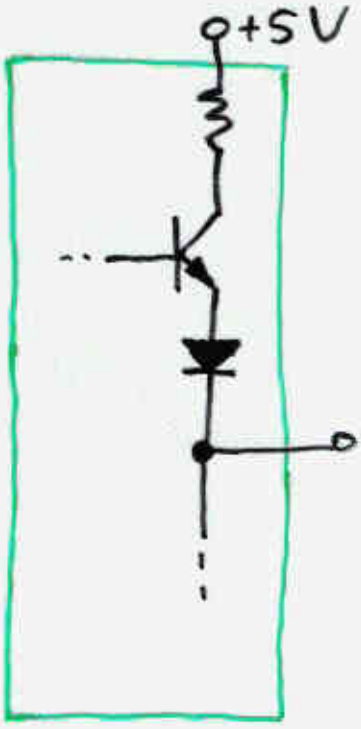
In general, cannot connect two outputs together (e.g., to increase current to an LED).



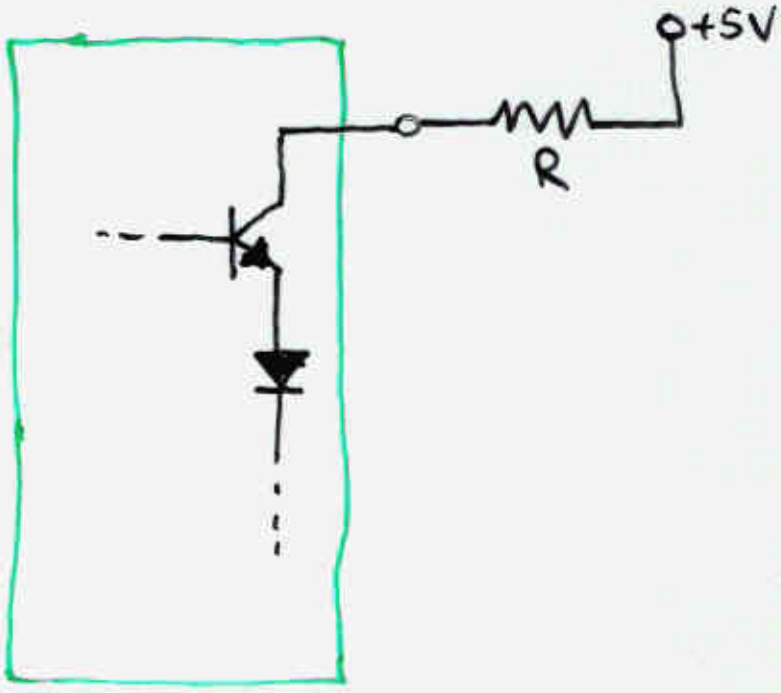
which one wins?

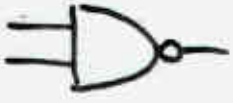
- one possibility: open-collector logic

Instead of:

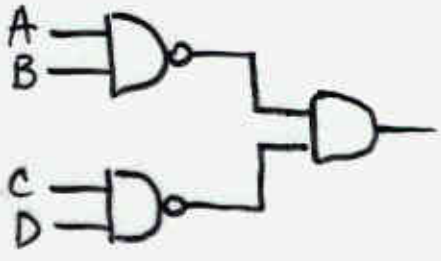


Use external R:

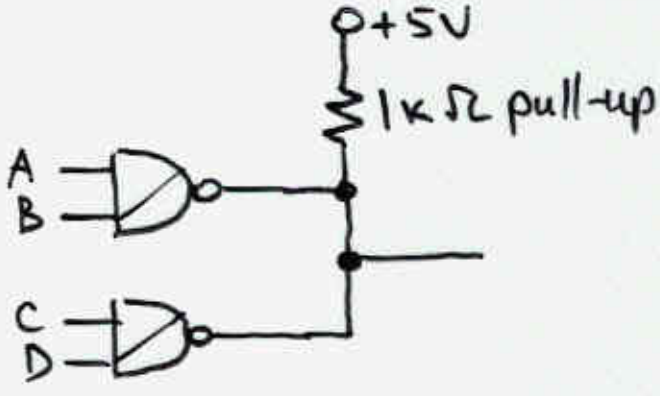




Compare: $(\overline{A \cdot B}) \cdot (\overline{C D})$



vs.



Disadvantages of OC logic: slow to switch, as the RC time constant is larger [if we make the external R smaller, power is drawn in the low-output state!]

- a better solution: three-state logic
  ("tri-state"™ National Semiconductor)

3 states: high / low / disabled (O.C.)

Ex a bidirectional data bus