Lecture 19 Flip Flops

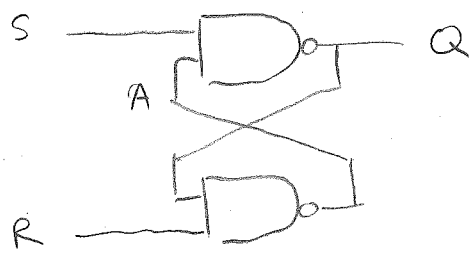
Two classes of logic circuit

Combinatoric: output determined by input

Sequential: output depends on history

Considered combinatoric circuits last week

Simple sequential circuit:



Analyze:

Suppose $S = L$

Then $Q = H$, regardless of A

If $S = H$, then Q depends on A

$Q = L$ if $A = H$

$Q = H$ if $A = L$

Now if $R = L$, then $A = H \Rightarrow Q = L$

If $R = H$, two possibilities:

if $Q = H$ then $A = L \Rightarrow Q = H$

if $Q = L$ then $A = H \Rightarrow Q = L$

Both consistent, circuit is bistable

S	R	Q
L	L	H
L	H	H
H	L	L
H	H	Q

So if $S = R = H$, Q holds whatever value it has

This type of circuit often called a latch:

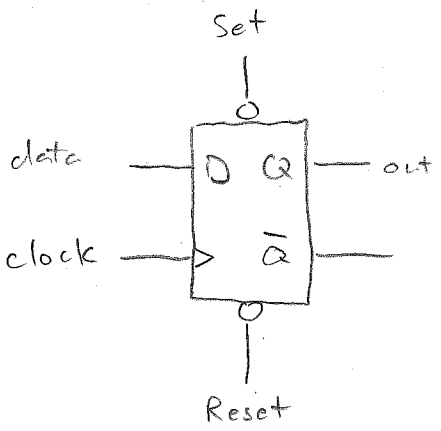
use S & R to put Q in a given state, then it stays there

= simple form of memory

Example:

- Set R low
- Apply data D to S: $Q \to \bar{D}$ (whatever D is)
- Set R & S high, Q holds value $\bar{D}$ indefinitely

More convenient version: D-type flip flop



Idea:

When clock signal rises: ⌐¯

value at D is stored in Q

Otherwise D and clock are ignored

R & S force Q to L or H, indep of D & clock

Truth table:

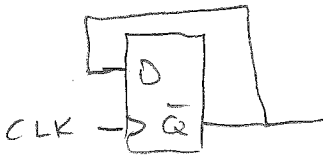
$\bar{S}$	$\bar{R}$	CLK	D	Q
L	X	X	X	H
H	L	X	X	L
H	H	↑	D	D
H	H	X	X	Q

"X" = doesn't matter

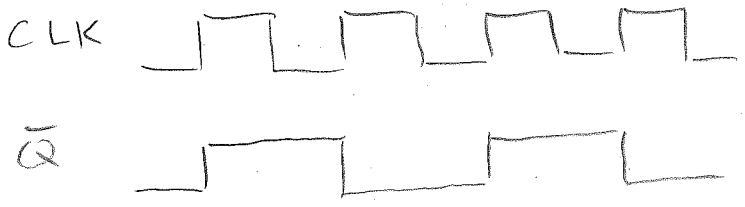
Using clock signal as trigger makes it easy to define when data is saved.

Useful for many things

Simplest:



Every time CLK ↑, output changes state



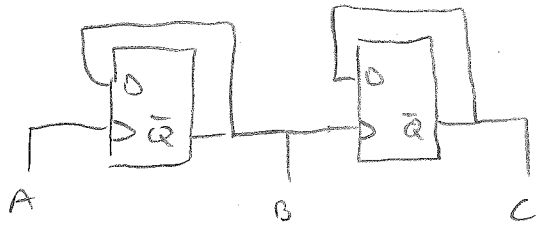
Divides clock frequency in half

Alternately, forms two-bit counter:

Cycle	$\bar{Q}$	CLK	Binary
	0	0	0
	1	1	3
	1	0	2
	0	1	1
	0	0	0

Counts down from 3 and then repeats
(to count up, invert both bits)

Can cascade circuits to make bigger counter:

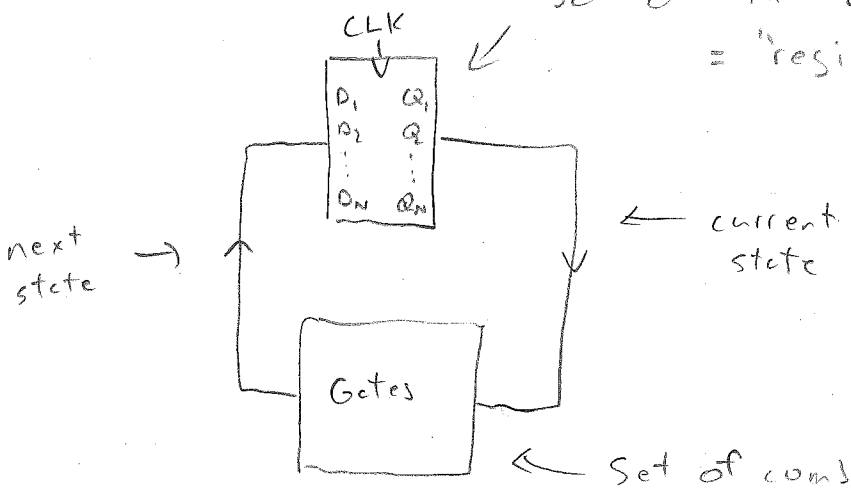


C	B	A	binary
1	1	1	7
1	1	0	6
1	0	1	5
1	0	0	4
0	1	1	3
0	1	0	2
0	0	1	1
0	0	0	0
1	1	1	7

Counters are important: have lab on them next week

More generally, counters are example of state machine
= triggered circuit that moves from one state
to another in a defined way

Generally:



Can use state machine as general purpose computer

For instance, if you want to add two numbers

1) Load numbers as initial data
2) Cycle through sequence of steps leading to
3) Output = sum of inputs

Can implement any sequence of logical steps
= program

Discuss further next time.