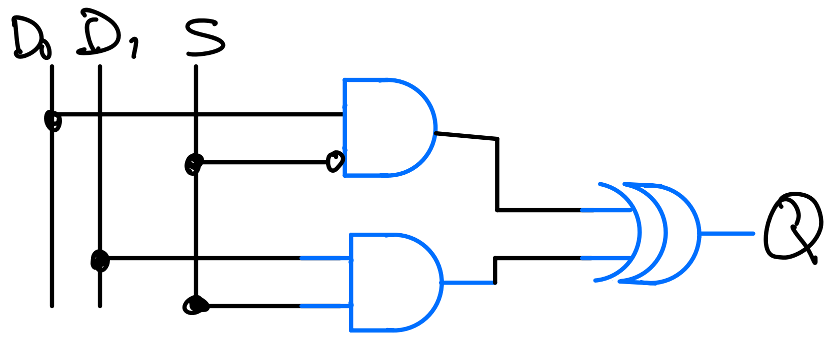
Some useful primitives

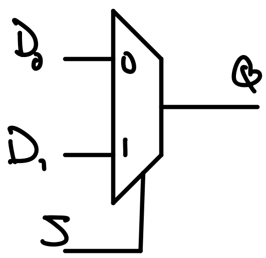
Multiplexor (aka "mux")

2 single-bit inputs D_0, D_1
1 "selector" S
1 output Q

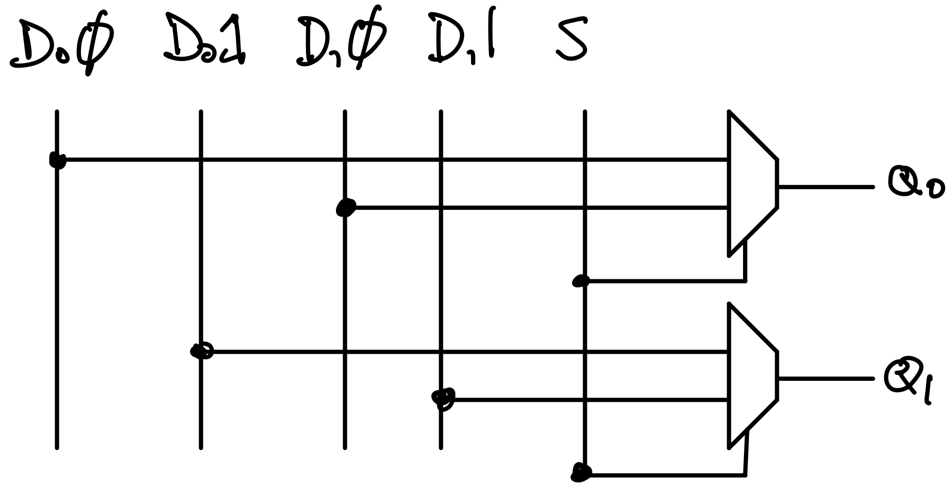
controller determines which input bit → Q
⇒ 2 inputs ∴ = 1 bit (2 "states")
so $Q = \bar{S} D_0 + S D_1$



primitive:



if inputs are 2 bits each, then Q is also 2 bits
but S is still 1-bit ⇒ just use 1-bit mux for each bit D, Q



4-bit mux D_0, D_1, D_2, D_3

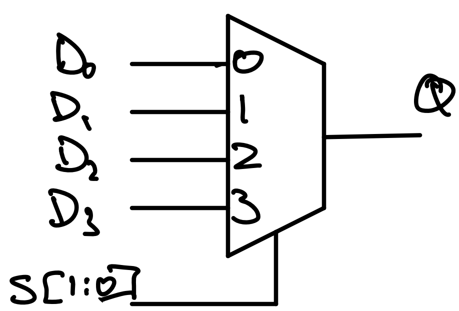
Q is just 1-bit

S has to have 4 states to differentiate which input is switched to Q so S[1:0]

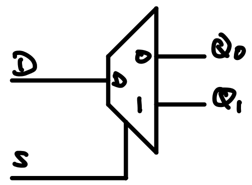
truth table

S_1	S_0	Q
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

so $Q = D_0 \overline{S_0}\,\overline{S_1} + D_1 S_0 \overline{S_1} + D_2 \overline{S_0} S_1 + D_3 S_0 S_1$

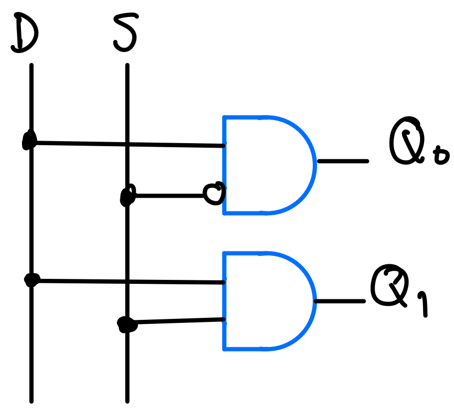


Demux → opposite of mux : D → Q_0 or Q_1 depending on S



$Q_0 = D\overline{S}$

$Q_1 = DS$



Decoder :

any binary # can be encoded into a bus

ex: 3 encodes to 11 for 2-bit bus D[1:0]

⇒ D[1:0] has 4 states

construct 4 outputs Q_0, Q_1, Q_2, Q_3 such that each output is "asserted" (is 1) depending on what number is encoded

D1	D0	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

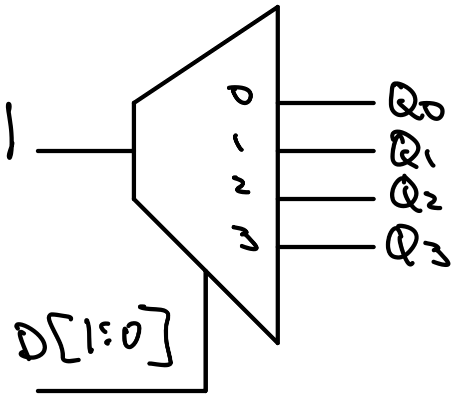
$Q_0 = \overline{D0}\,\overline{D1}$

$Q_1 = D0\,\overline{D1}$

$Q_2 = \overline{D0}\,D1$

$Q_3 = D0\,D1$

note: this is equivalent to using 4-bit demux with a constant input 1



Comparitor: test on 2 signals A, B

can be $A=B$, $A<B$, $A>B$, $A \neq B$

A B	A=B	A≠B	A>B	A<B
0 0	1	0	0	0
0 1	0	1	0	1
1 0	0	1	1	0
1 1	1	0	0	0

$A=B$: $Q_= = \bar{A}\bar{B} + AB = \overline{A \oplus B}$

$A \neq B$: $Q_{\neq} = A \oplus B$

$A>B$: $Q_> = A\bar{B}$

$A<B$: $Q_< = \bar{A}B$

$$Q = Q_= + Q_{\neq} + Q_> + Q_<$$

4 possible combs of A, B, ∴ 4 outputs that are mutually exclusive

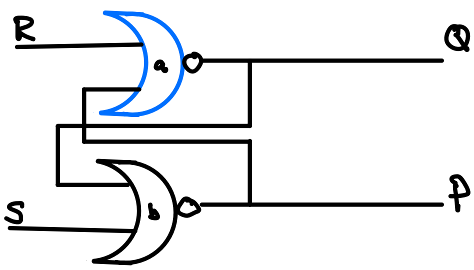
So far we have learned about "sequential logic"
⇒ inputs flow thru some sequence of gates

note: the outputs follow inputs at <u>all</u> times
what seq logic lacks:

1. ability to specify the time for things to happen
2. " " remember anything (memory)

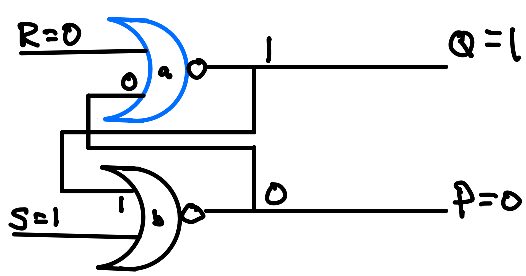
Memory ⇒ controlled feedback

try this circuit



Hard to tell what this will do!

⇒ Specify initial states of R & S, see how evolves
S=1, R=0



S=1 means gate b will give ϕ on output due to ouput inverter

⇒ P=0

R=0 and red input=0 means gate a will be 1

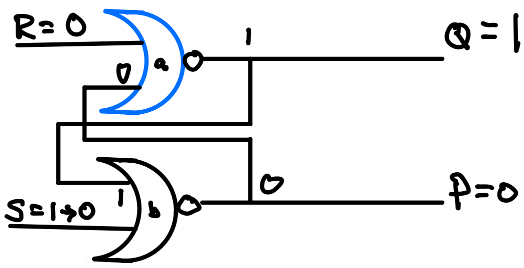
$\Rightarrow Q=1$

R	S	Q	P	state	
0	1	1	0	"set"	$S\bar{R}$

what happens when $S \to 0$ here?

$\Rightarrow$ nothing b/c gate a is an OR & the other input=1

$\Rightarrow$ remembers!

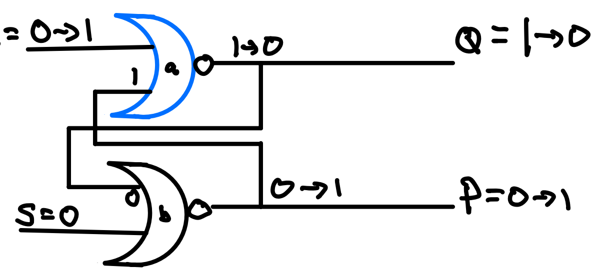


$S\bar{R} \to \bar{S}\bar{R}$

next transition from $\bar{S}\bar{R}$ to $\bar{S}R$ (asserting R)

this changes output of gate a to 0 (Q=0)

but since S=1, gate b doesn't change



$R \to 1$ turns off gate a (NOR), and S=0 so gate b turns on

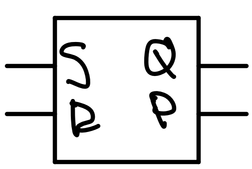
this changes outputs from "set" (QP=10) to "reset" (QP=01)

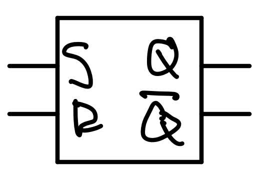
R	S	Q	P	state	
0	1	1	0	"set"	$S\bar{R}$
0	0	1	0	"hold"	$\bar{S}\bar{R}$
1	0	0	1	"reset"	$R\bar{S}$

if you now transition $S \to 1$ (RS state)
then gate b $\to 0$, $P \to 0$
but since $R=1$ already, no change in $Q=0$

	R	S	Q	P	state
$\bar{R}S$	0	1	1	0	"set"
$\bar{R}\bar{S}$	0	0	1	0	"hold"
$R\bar{S}$	1	0	0	1	"reset"
RS	1	1	0	0	$P \neq \bar{Q}$

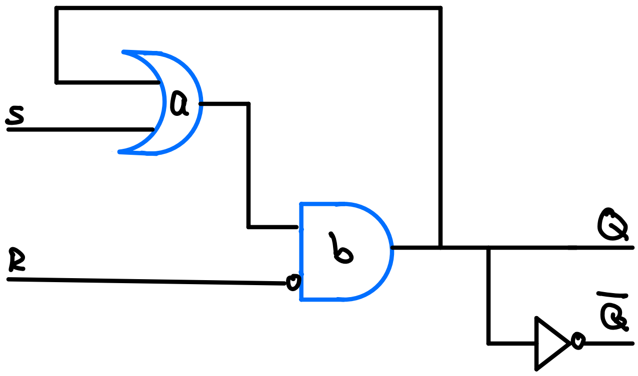
this is called SR "latch"
works great for somethings although $P \neq \bar{Q}$ is maybe a problem

primitive 

or 

since $P = \bar{Q}$ (mostly)

Another way to make RS latch that is more well behaved:



here $P = \bar{Q}$ explicitly, so outputs will never be equal

set $S\bar{R} \Rightarrow a=1, b=1, Q=1, \bar{Q}=0$ "set"

$S\bar{R} \to \bar{S}\bar{R}$ doesn't change Q b/c a is OR "hold"

$\bar{S}\bar{R} \to \bar{S}R$ turns off b, $Q \to 0$ "reset"

R	S	Q
0	1	set
1	X	hold
0	0	reset