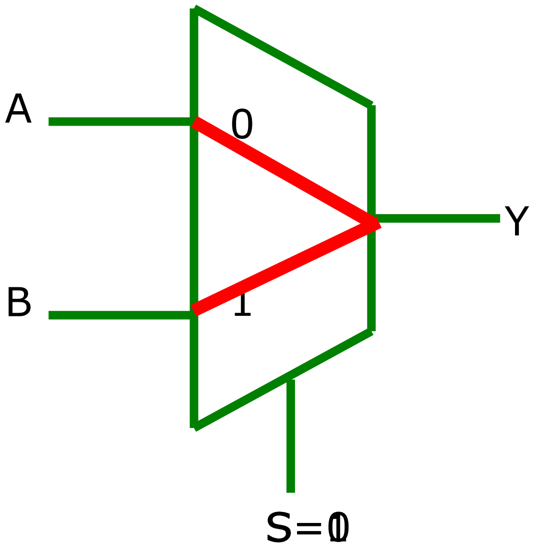
Multiplexor (aka MUX)
An example, yet VERY useful circuit!

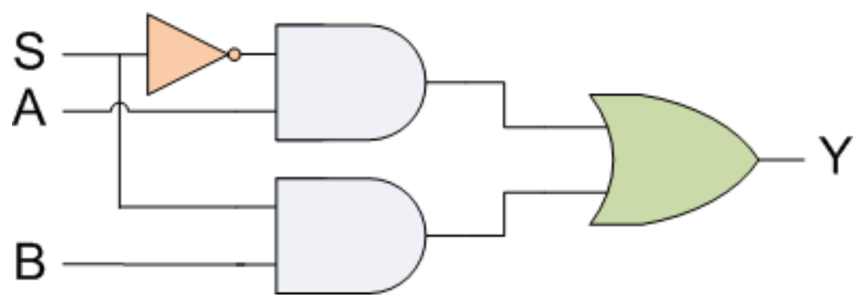


Y = (S) ? B:A;

when S =
0: output A
1: output B

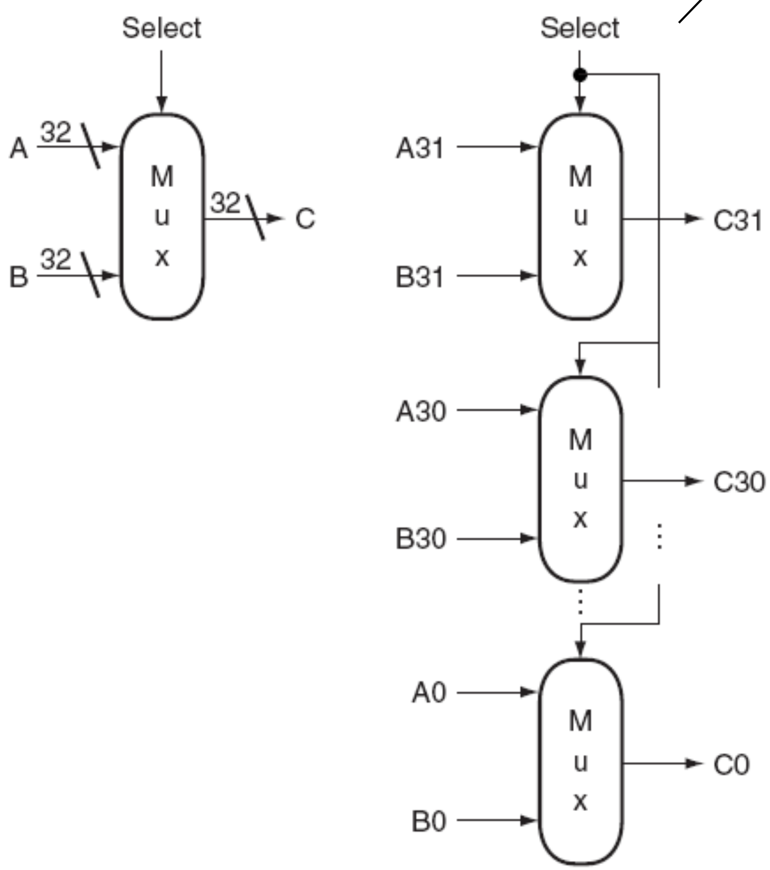
S	A	B	Y
0	0	x	0
0	1	x	1
1	x	0	0
1	x	1	1

Y=S'A+SB



A 32-bit MUX

Use 32 1-bit muxes
Each mux selects 1 bit
S is connected to each mux

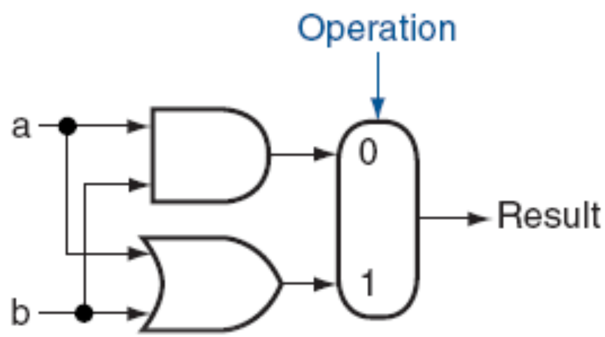


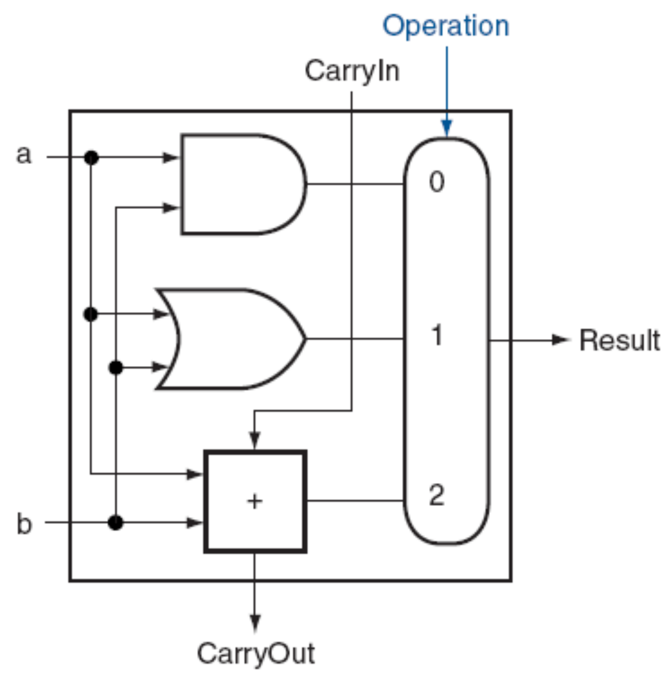
a. A 32-bit wide 2-to-1 multiplexor

b. The 32-bit wide multiplexor is actually an array of 32 1-bit multiplexors

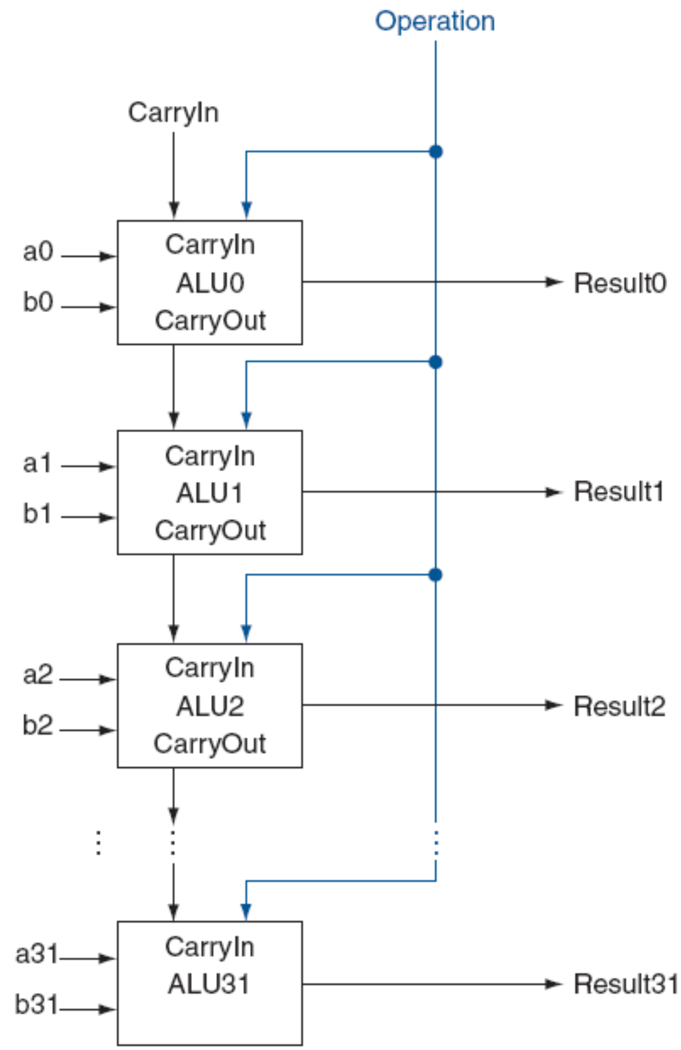
Building a 1-bit ALU

- ALU = arithmetic logic unit = arithmetic unit + logic unit



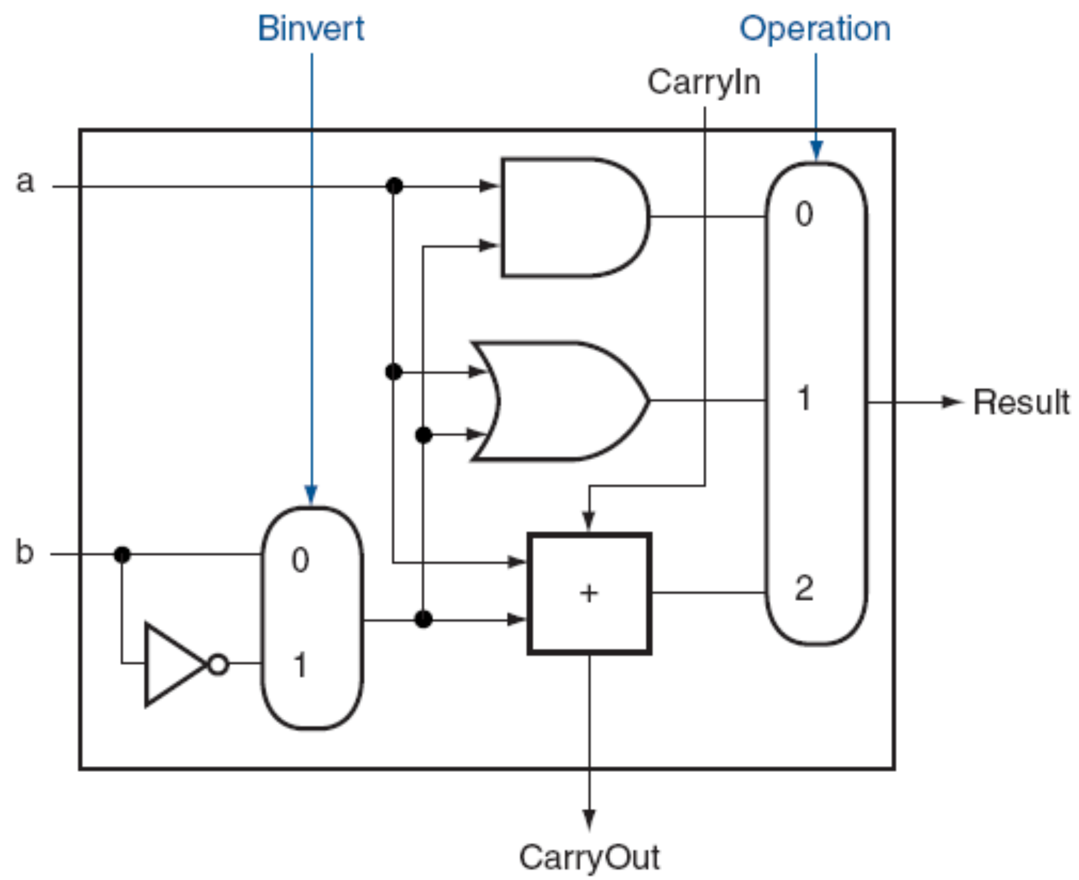


Building a 32-bit ALU



Implementing “sub”

Binvert=1
CarryIn=1 for 1st 1-bit ALU
Operation=2

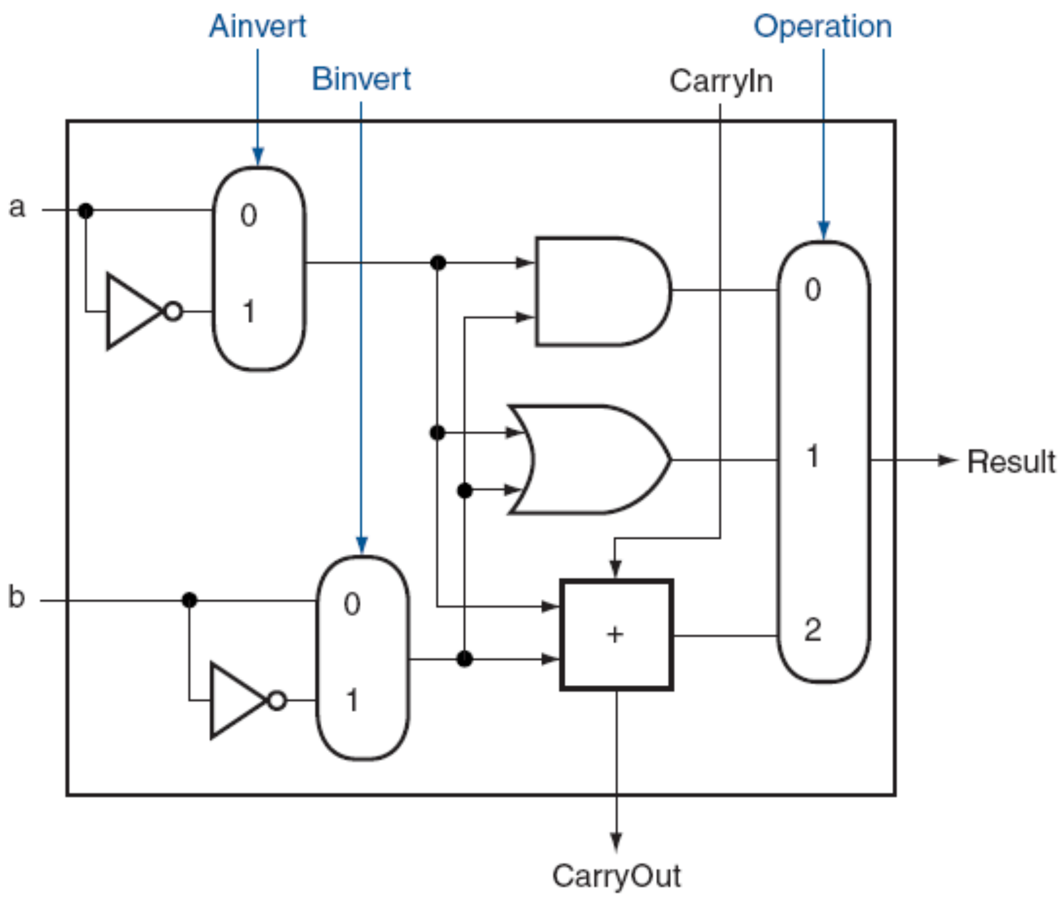


Implementing NAND and NOR

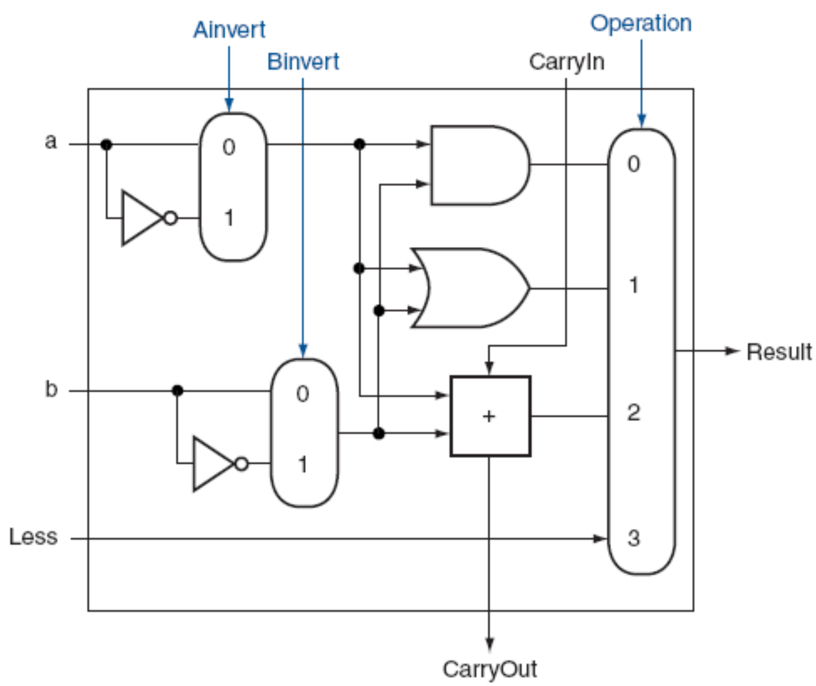
NOR:
NOT (A OR B)
by DeMorgan's Law:
(NOT A) AND (NOT B)

Thus,
Operation=0,
Ainvert=1,
Binvert=1

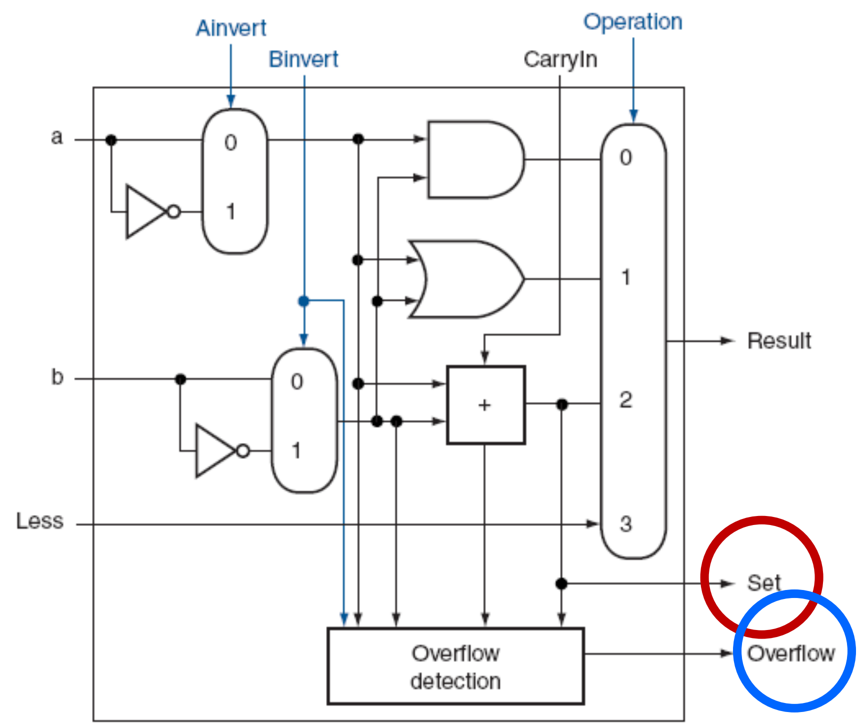
And, NAND???



Implementing SLT (set-less-than)



1-bit ALU for bits 0~30



1-bit ALU for bit 31

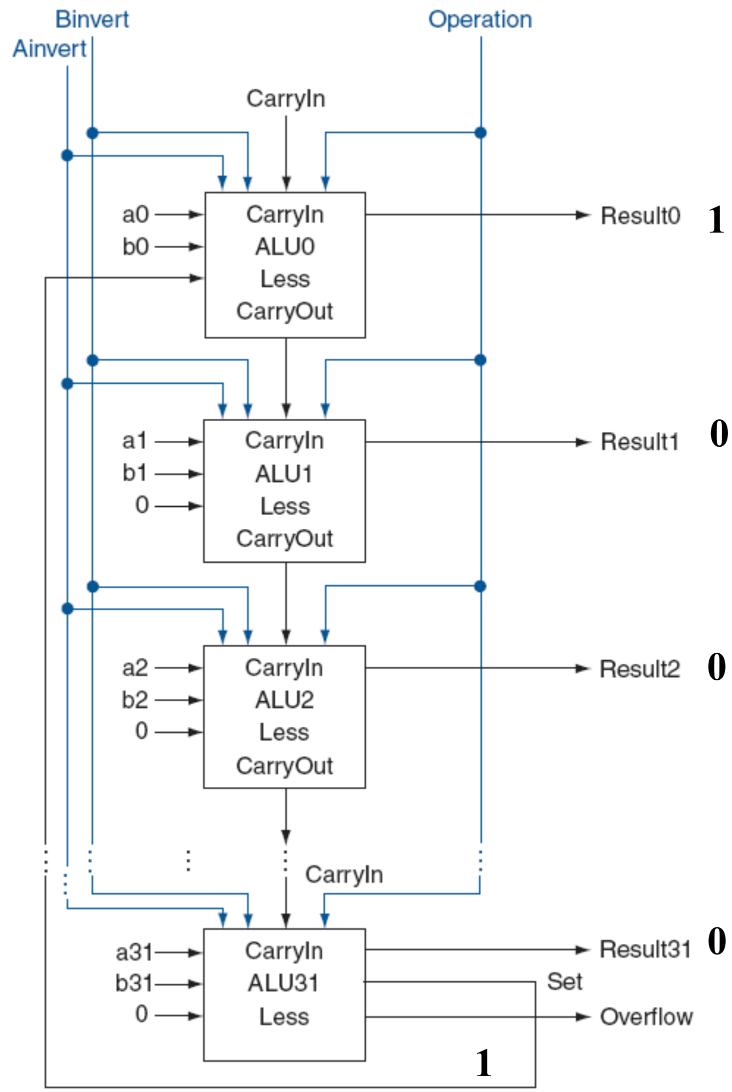
Implementing SLT (set-less-than)

<u>SLT uses subtraction</u>
slt \$t0,\$t1,\$t2
\$t1<\$t2: \$t1-\$t2 gives negative result
set is 1 when negative

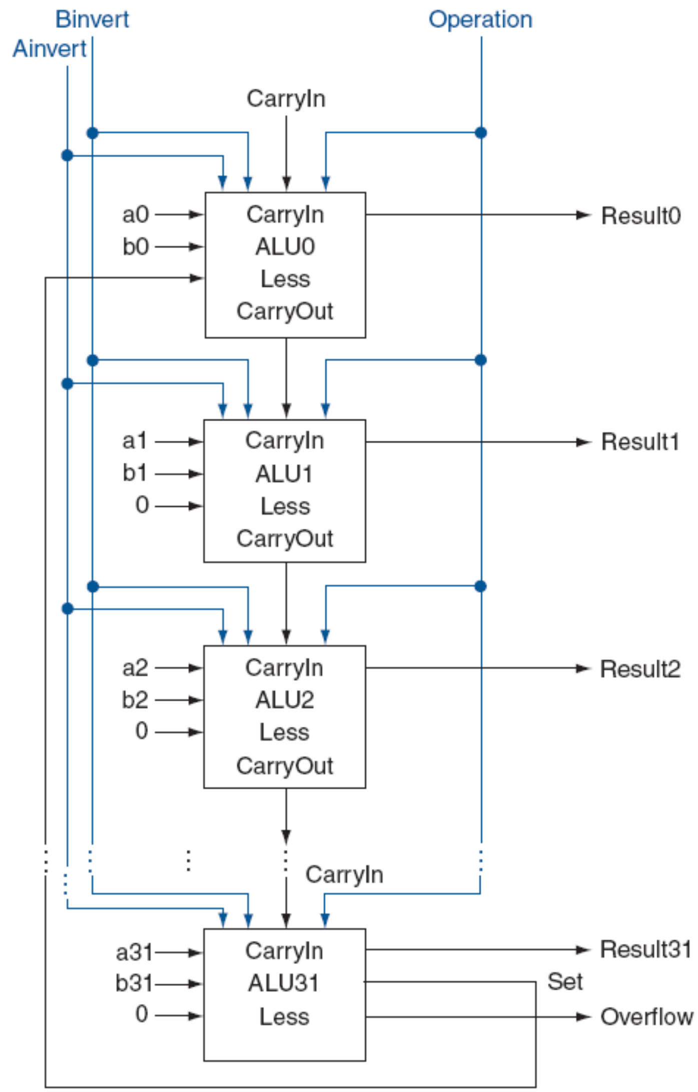
<u>Setting the control</u>
perform subtraction (Cin=1,Binvert=1)
select Less as output (Operation=3)
ALU31's Set connected to ALU0 Less

<u>Consider</u>
Suppose \$t1=10 and \$t2=11

\$t1 - \$t2 = -1 = 1111…1 binary
\$t0 = 0000...1



Implementing SLT (set-less-than)



SLT uses subtraction

slt \$t0,\$t1,\$t2

\$t1<\$t2: \$t1-\$t2 gives negative result

set is 1 when negative

Setting the control

perform subtraction (Cin=1,Binvert=1)

select Less as output (Operation=3)

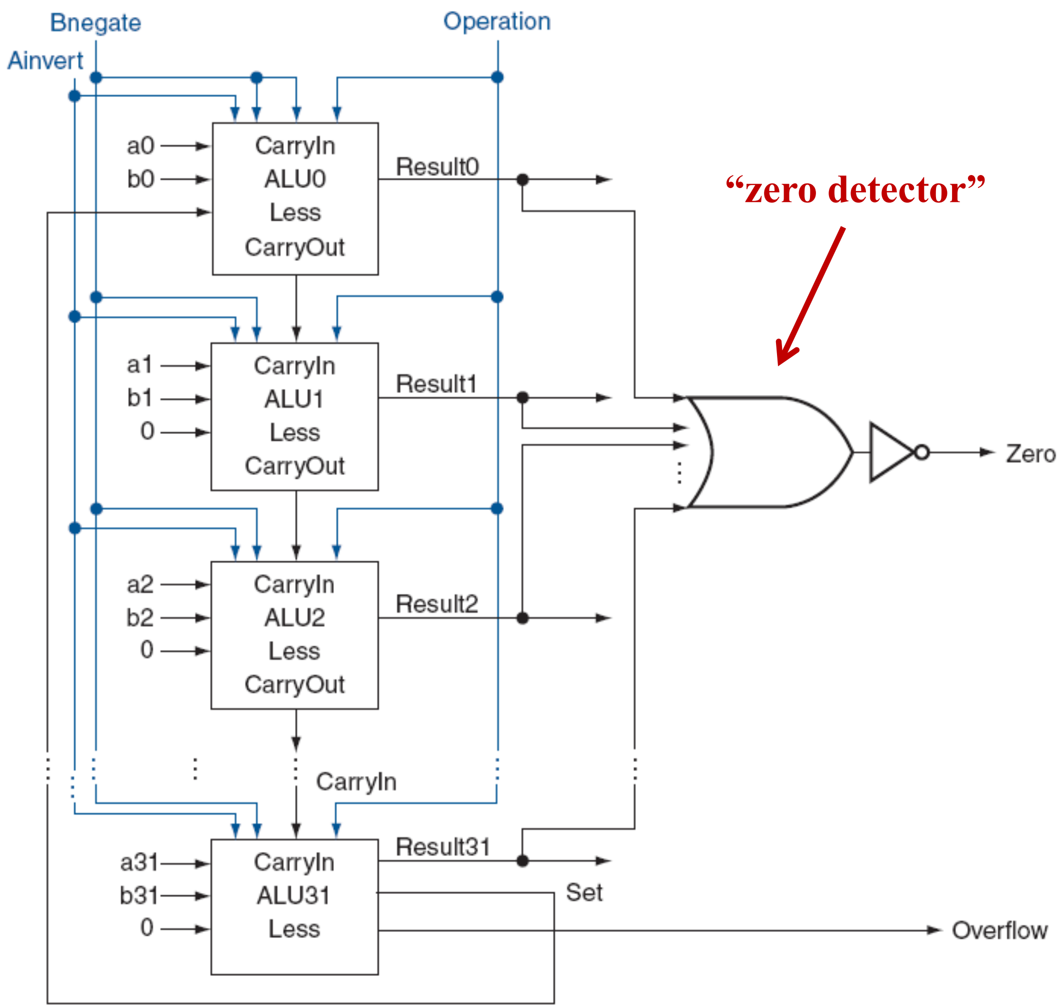
ALU31's Set connected to ALU0 Less

Why do we need Set? Could we use just the Result31?

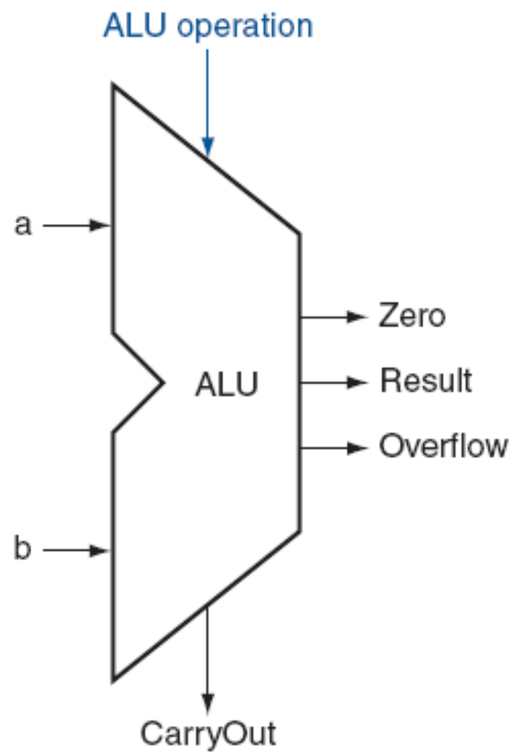
Supporting BEQ and BNE

<u>BEQ uses subtraction</u>
beq \$t0,\$t1,LABEL
perform \$t0-\$t1
result=0 ➔ equality

<u>Setting the control</u>
subtract (Cin=1,Binvert=1)
select result (operation=2)
detect zero result



Abstracting ALU



- Note that ALU is a combinational logic

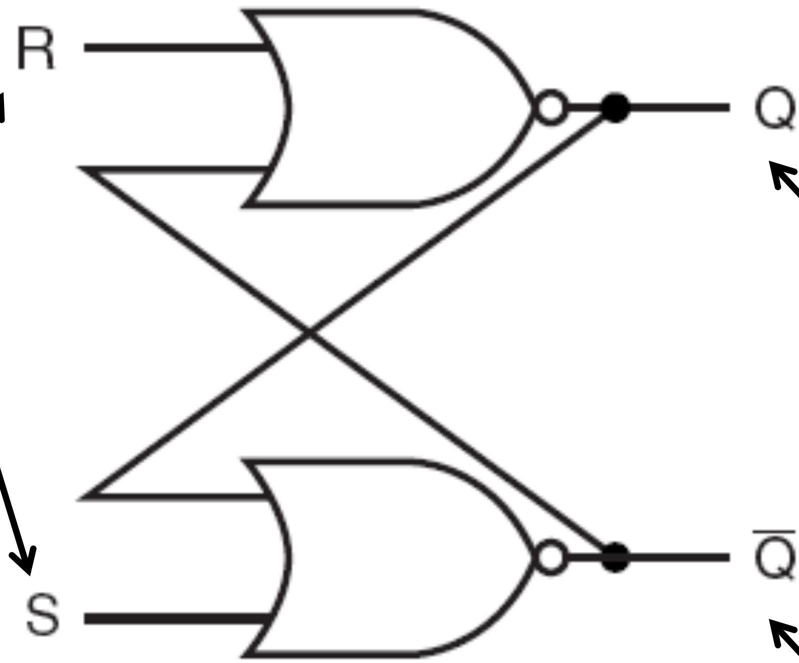
Sequential Logic

- Output depends on ***sequence of previous*** inputs
  - "Sequence of previous inputs" – this is history
  - History is a state that captures how you get here
    - E.g., 25 cents vending = 10 cents + 10 cents + 5 cents
    - Or, 25 cents + 10 cents = 35 cents. Multiple ways are possible.
  - State requires memory – remembering the past…
- Memory in logic
  - Smallest element is 1 bit of memory
  - Use logic gates to create a 1-bit memory
  - Yet, combinational logic (using gates) depends on present inputs!
- Fundamental building block: "RS latch"
  - 1 bit of history through feedback of gates

RS latch

Two NOR gates
With feedback

Inputs
RS control writing
a 0 or 1 in state

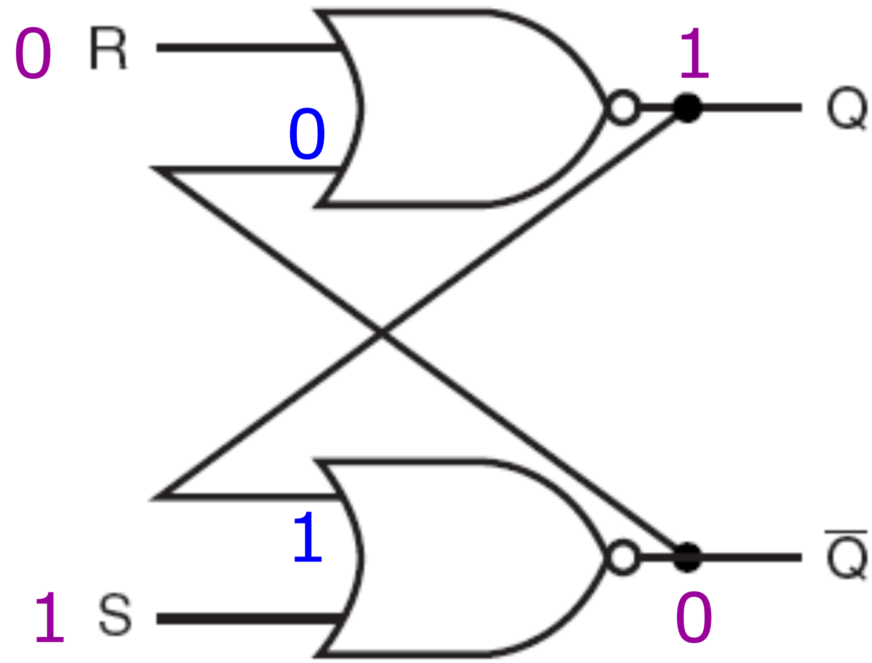


Output
Current state (0 or 1)

Complement of current state

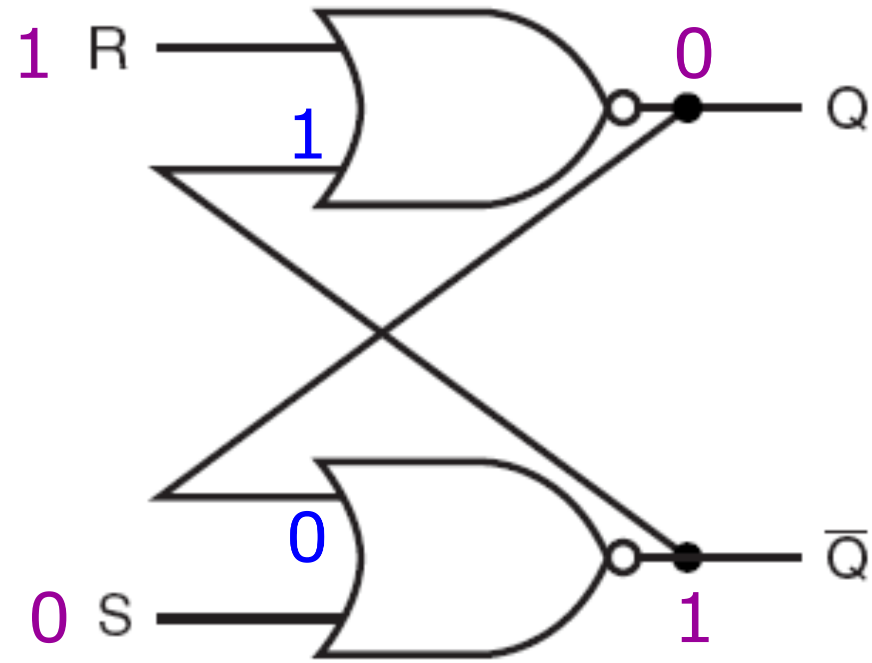
- Beware of the feedback!

RS latch



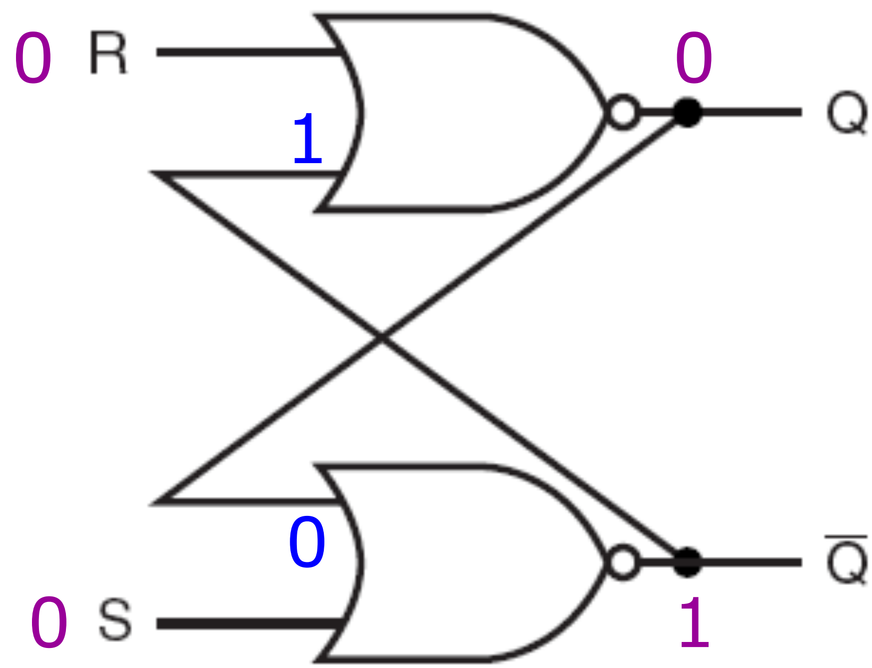
- When R=0, S=1

RS latch



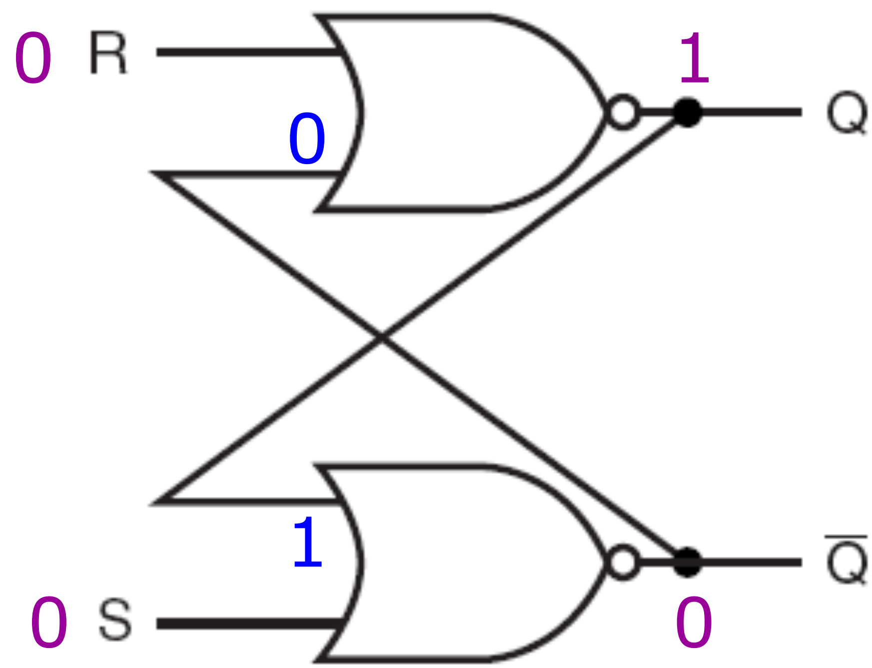
- When R=1, S=0

RS latch



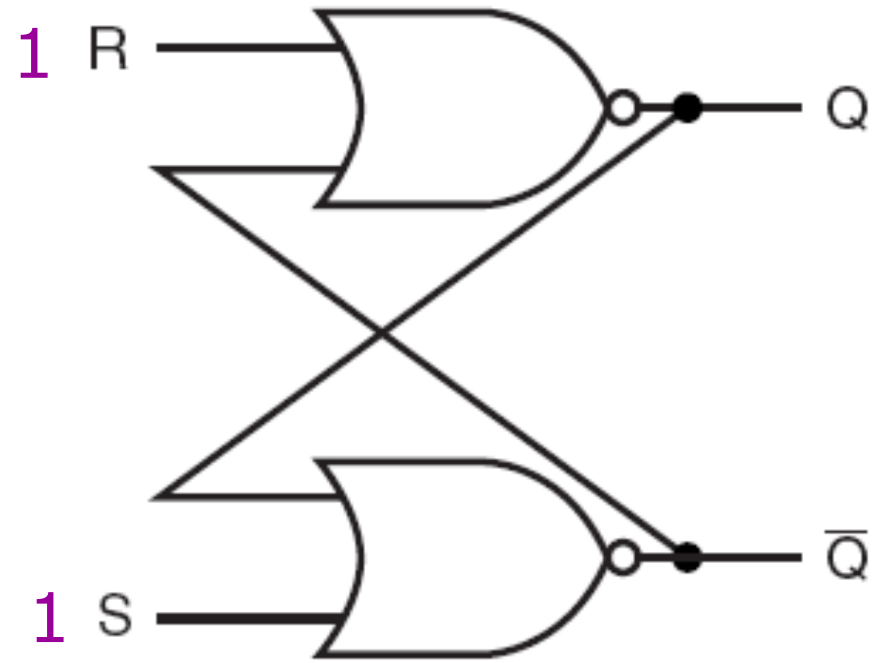
- When R=0, S=0, and Q was 0

RS latch



- When R=0, S=0, and Q was 1

RS latch



- What happens if R=S=1

RS latch truth table

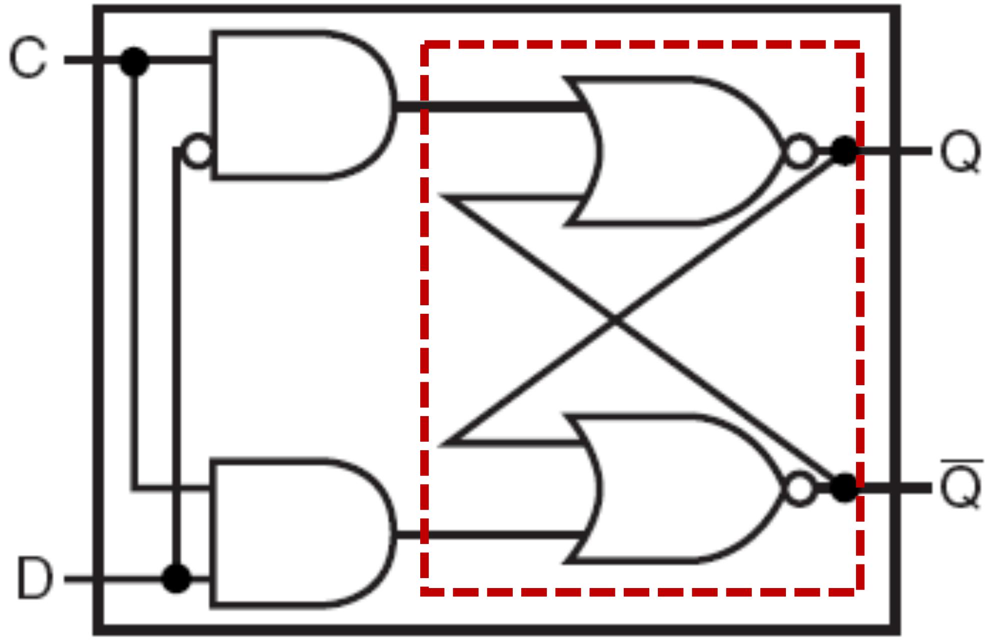
R	S	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	Invalid
1	1	1	Invalid

- **Storage (R=0, S=0)**
- **Set to 1 (S=1)**
- **Reset to 0 (R=0)**

Outputs will track any changes in the inputs!
R=1, S=1 must be avoid.

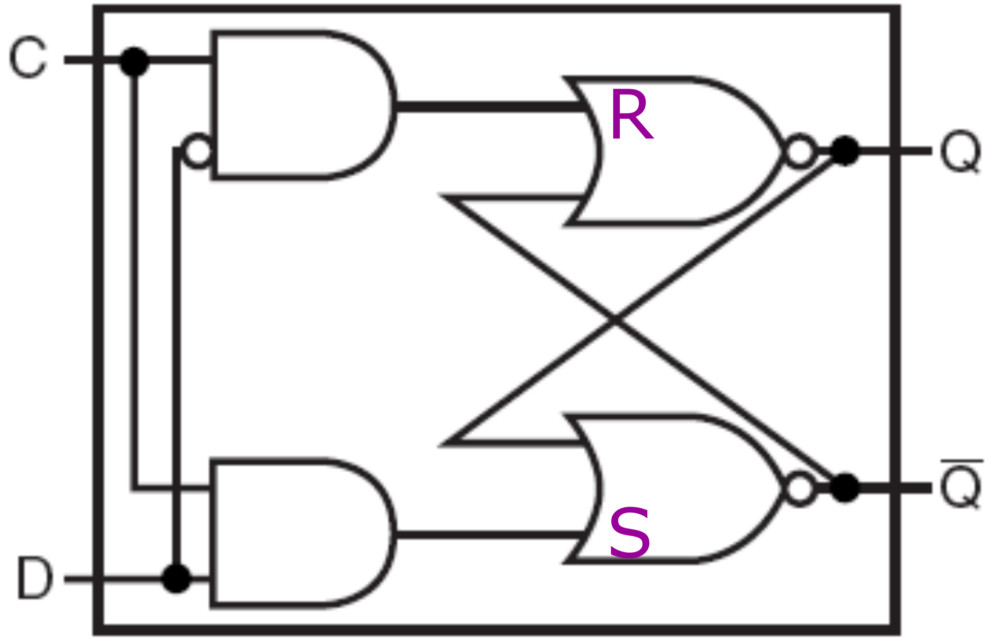
Desirable to control when to capture input state.

D latch



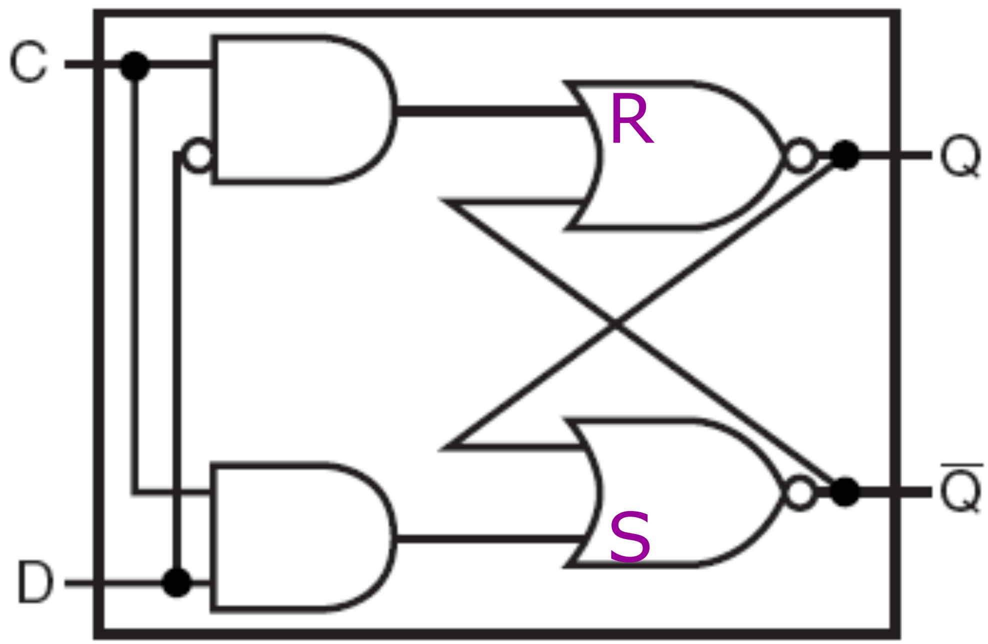
- Note that we have an RS latch in the back-end of this design

D latch



- Note that R, S inputs always get opposite values when C=1
- When C=0, S=R=0 ⇒ RS latch remembers the previous value

D latch

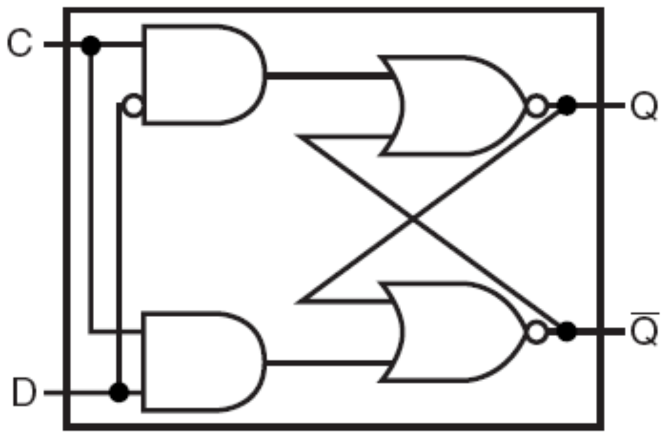


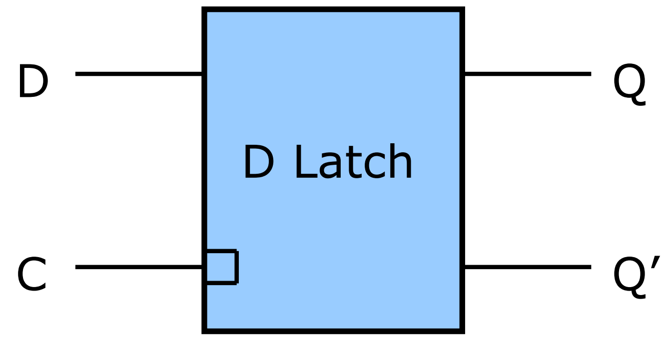
"latched mode"

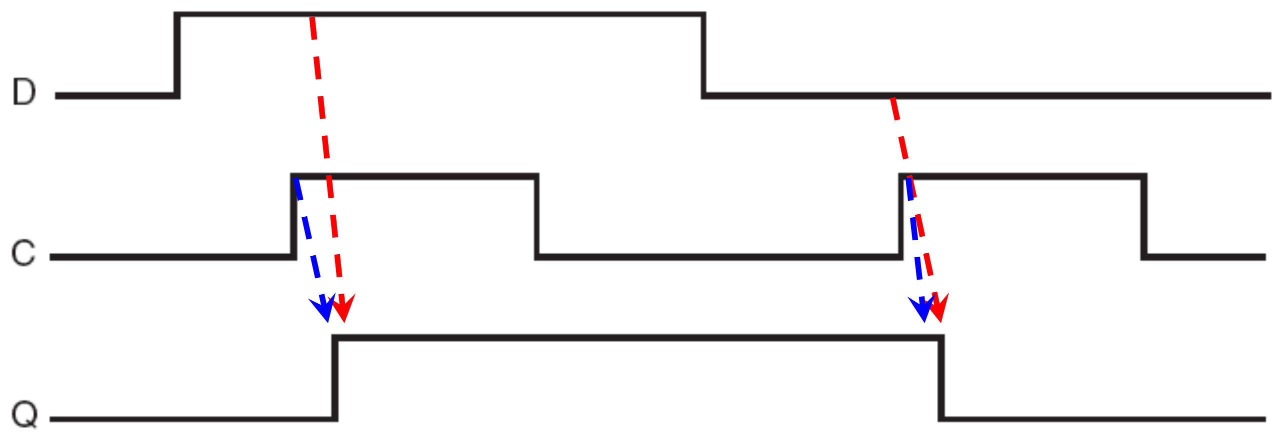
C	D	Q(t)
0	0	Q(t-1)
0	1	Q(t-1)
1	0	0
1	1	1

"transparent mode"

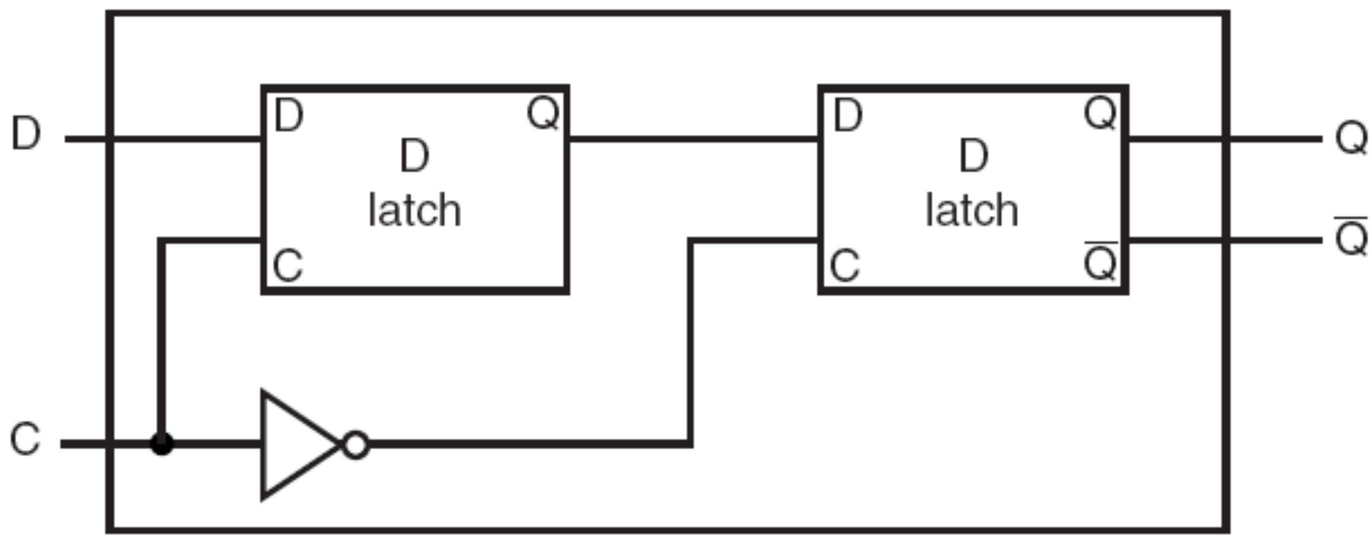
D latch





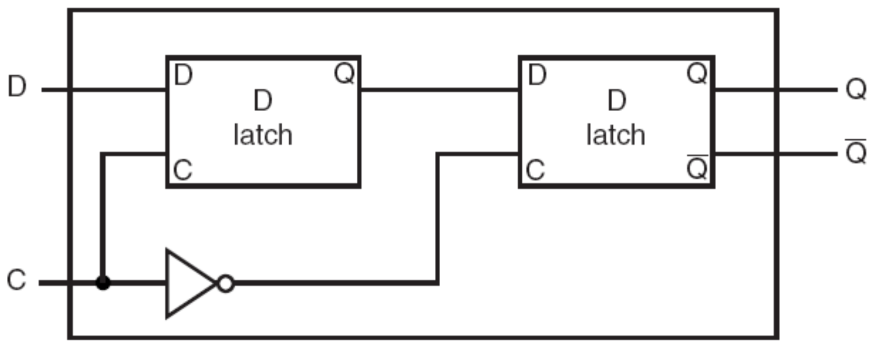


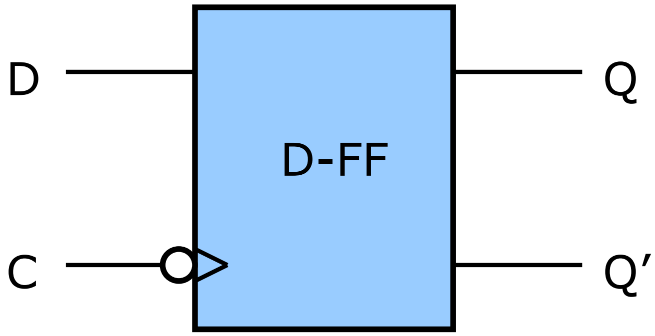
D flip-flop (D-FF)

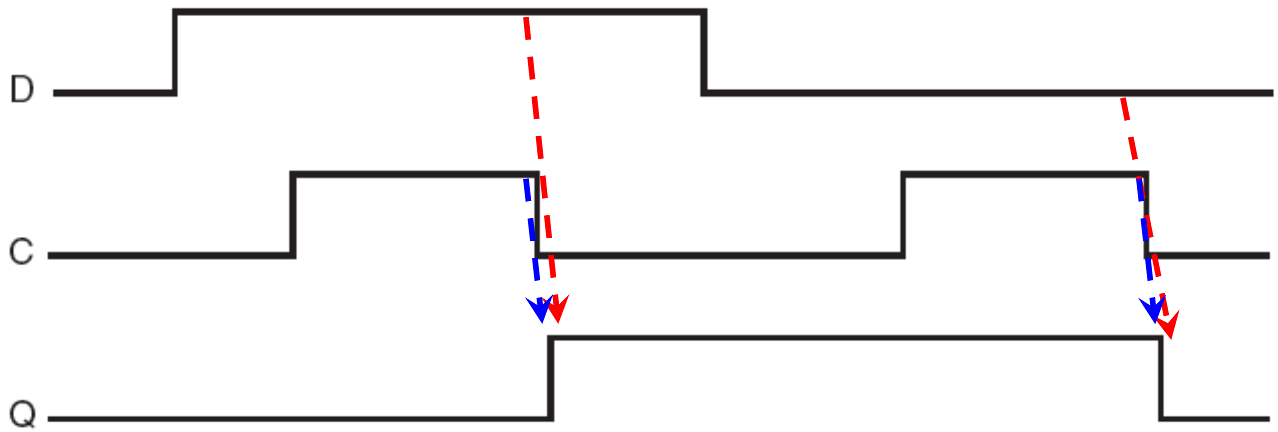


- Two cascaded D latches; C input of the second is inverted
- This is a negative edge (aka "falling edge") triggered D-FF

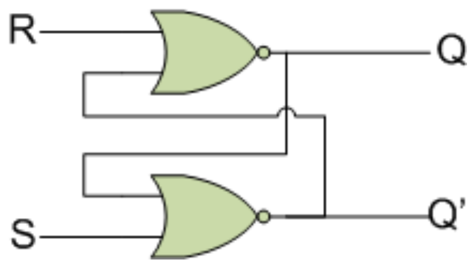
D flip-flop

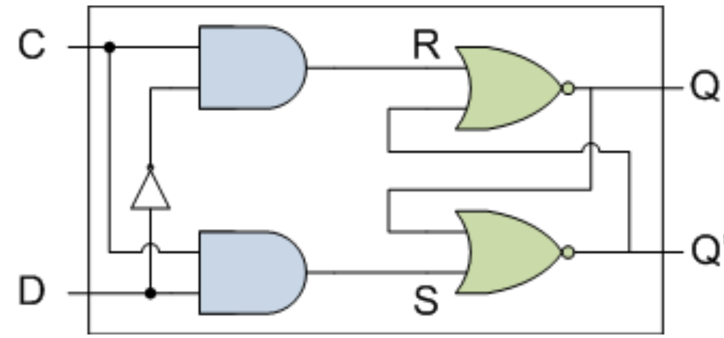






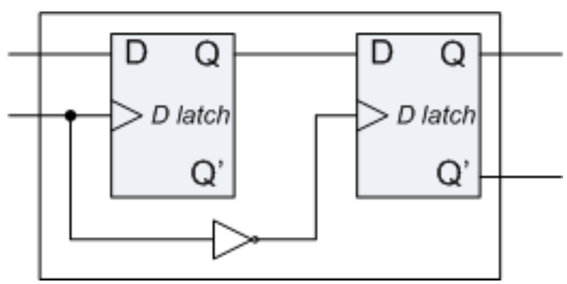
State Elements



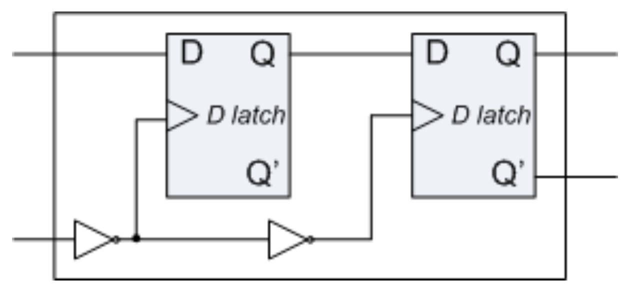


<u>RS latch</u>
R,S control mode (reset, set, storage)
Q,Q' track R and S
R=1, S=1 invalid

<u>D latch</u>
C controls mode (0=latched, 1=transparent)
D is data input ("copied" during transparent)
Signal value triggered: Q,Q' track D when C=1
Guarantees R=1,S=1 can not be done

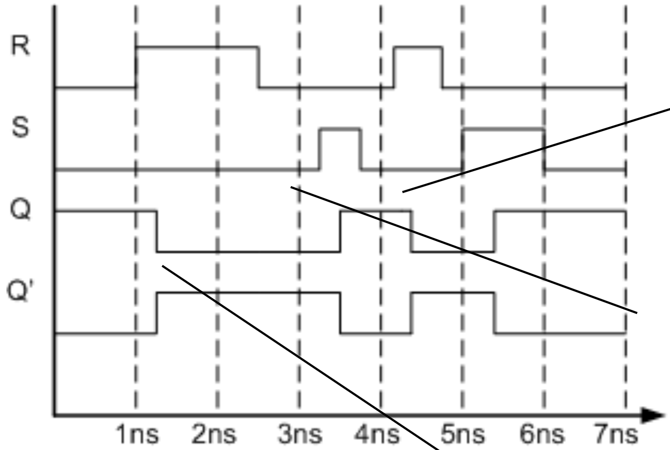


<u>D flip-flop (falling or negative edge triggered)</u>
Two cascaded D latches
C=1 means 1st latch transparent, 2nd latched
C=0 means 1st latch latched, 2nd transparent
Output changes on falling edge (C: 1=>0)



<u>D flip-flop (rising or positive edge triggered)</u>
Same as falling edge triggered
Output changes on rising edge (C: 0=>1)

Signaling Behavior



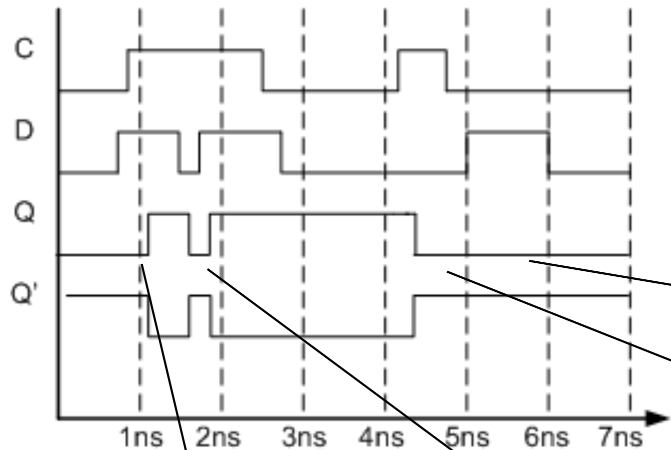
RS latch behavior

Q goes high R=0,S=1

Storage mode

Q goes low R=1,S=0

Signaling Behavior



D latch behavior

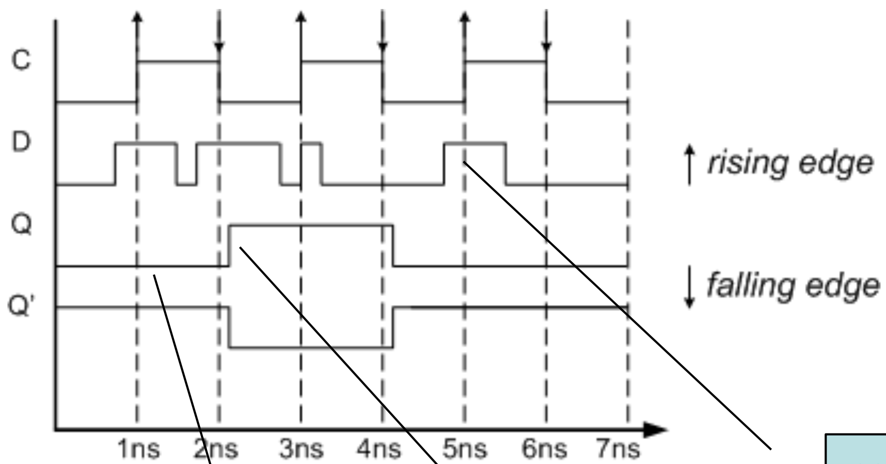
Q stays low b/c C=0

Q went low C=1,D=0

Q tracked D when C=1

Q goes C=1, D=1

Signaling Behavior



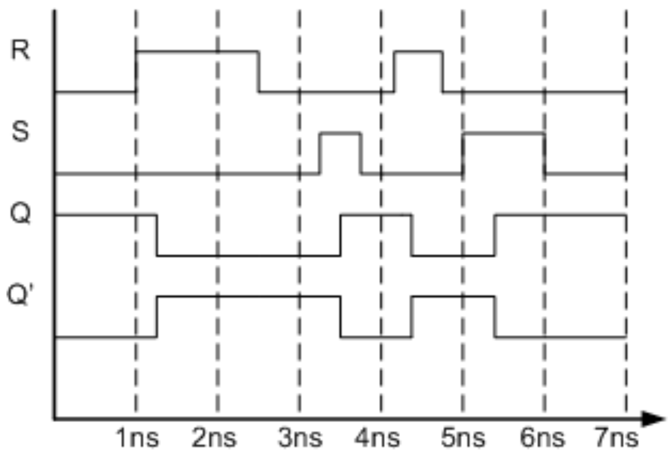
D flip-flop (falling edge triggered)

no change b/c not a falling edge

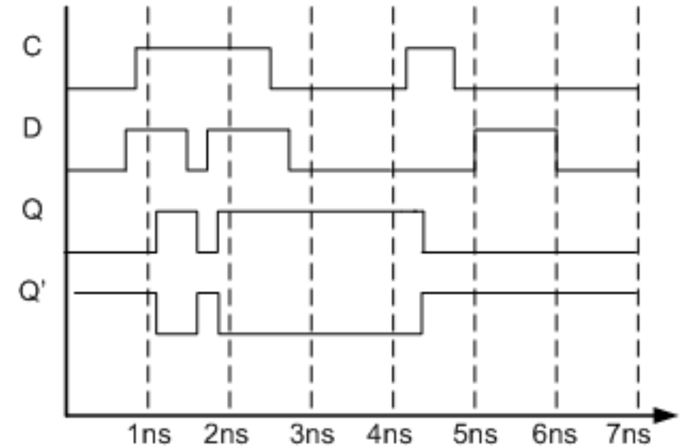
tracks D at time of falling edge

no change b/c not falling edge

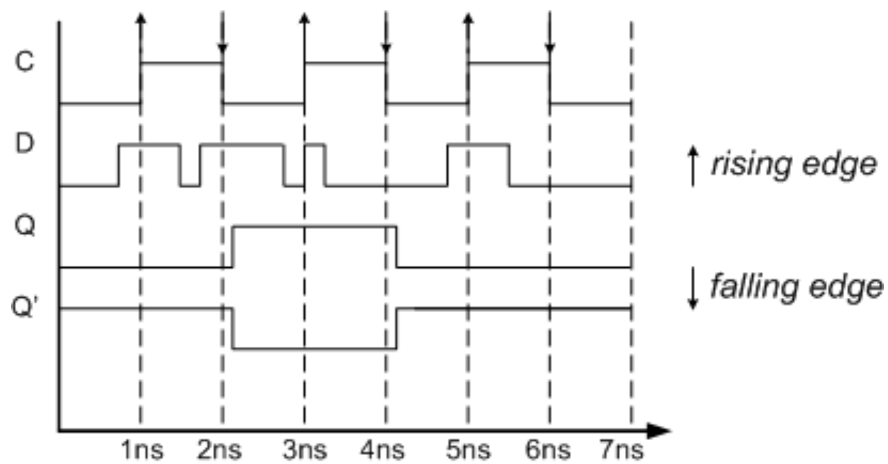
Signaling Behavior



RS latch behavior



D latch behavior



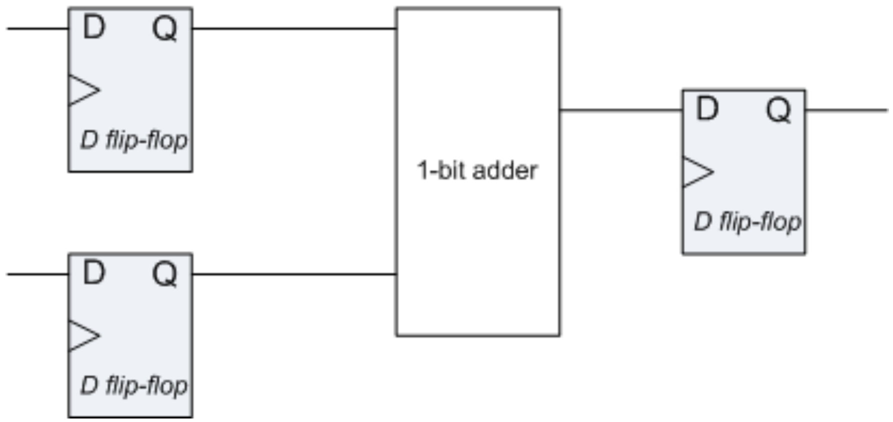
D flip-flop (falling edge triggered)

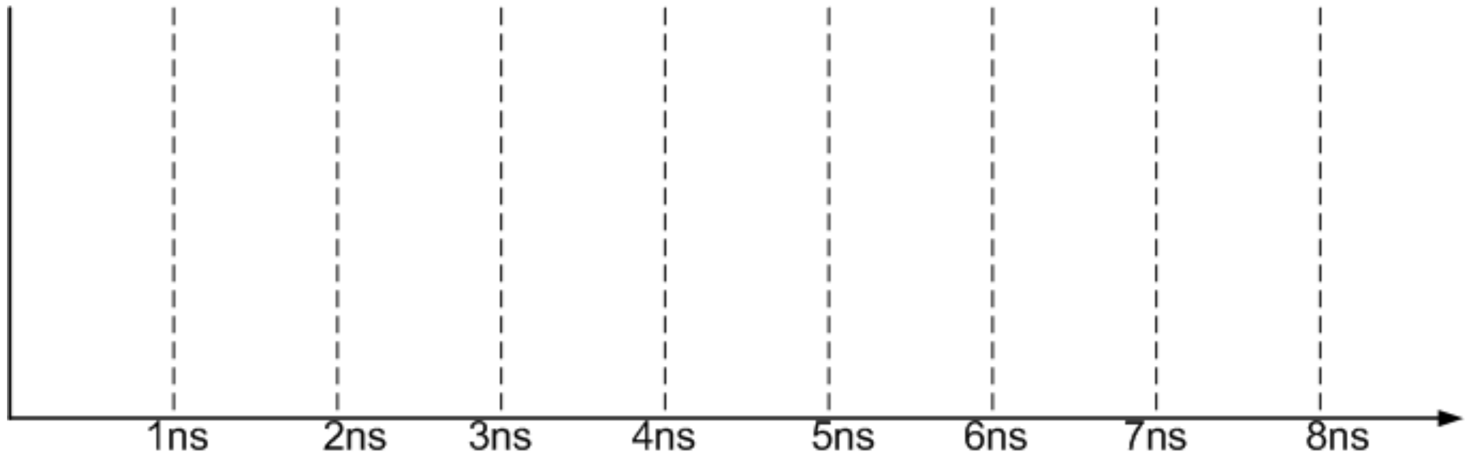
Example circuits and clocking

- Suppose we want to:
  - 1-bit value A stored in a D flip-flop
  - 1-bit value B stored in a D flip-flop
  - 1-bit value C stored in a D flip-flip
  - Do 1-bit addition of A and B, producing C

- C = A + B
  - What is the circuit?
  - Need three D flip-flops
  - Need one 1 bit adder

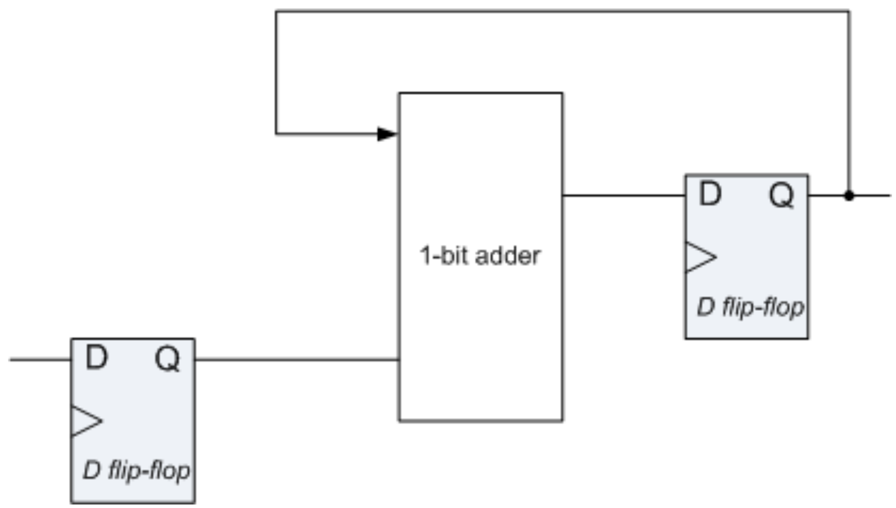
Example circuits and clocking





Example circuits and clocking

- Is there any difference in the delay with this one?



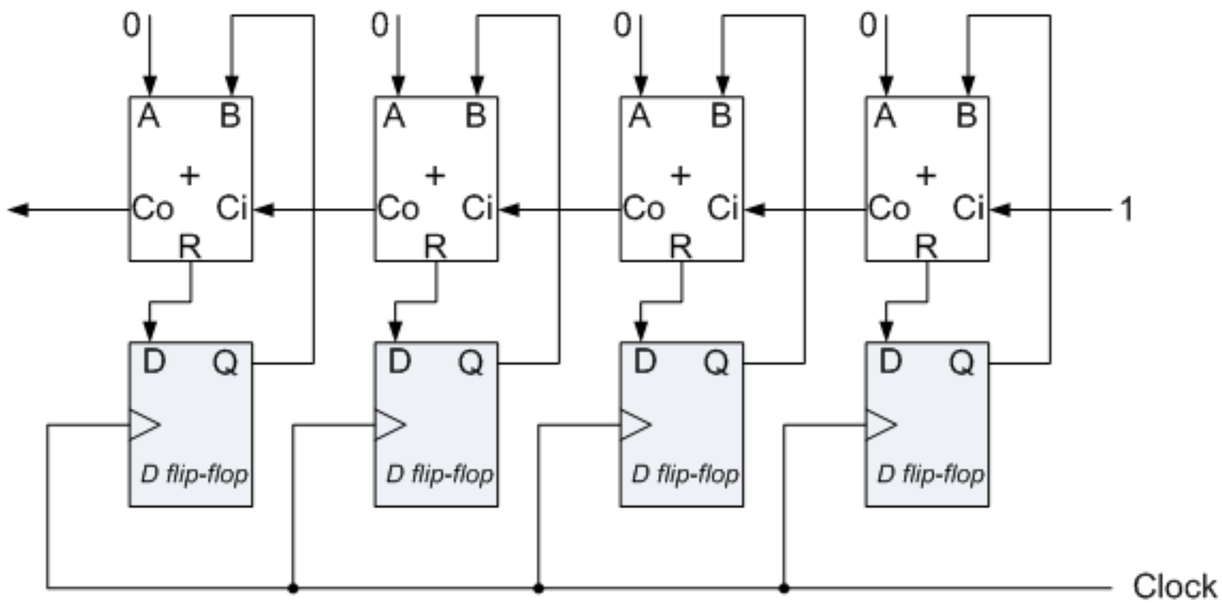
- In fact, sequential logic often looks like this….

Example circuits and clocking

- Now, suppose we want to build a 4-bit counter?
  - Counter increments by 1 for a clock pulse (falling edge event)
  - 4 1-bit adders
  - 4 1-bit D flip-flops

- What's the circuit?
- How often to "pulse" the clock (increment counter)?

Example circuits and clocking



Recall: The flip-flops are edge triggered -- assuming falling edge (negative)

How often can an edge event happen?

No more frequent than the maximum propagation delay
Let's compute the delay -- assume 2ns for latch to stabilize and 4ns for adder

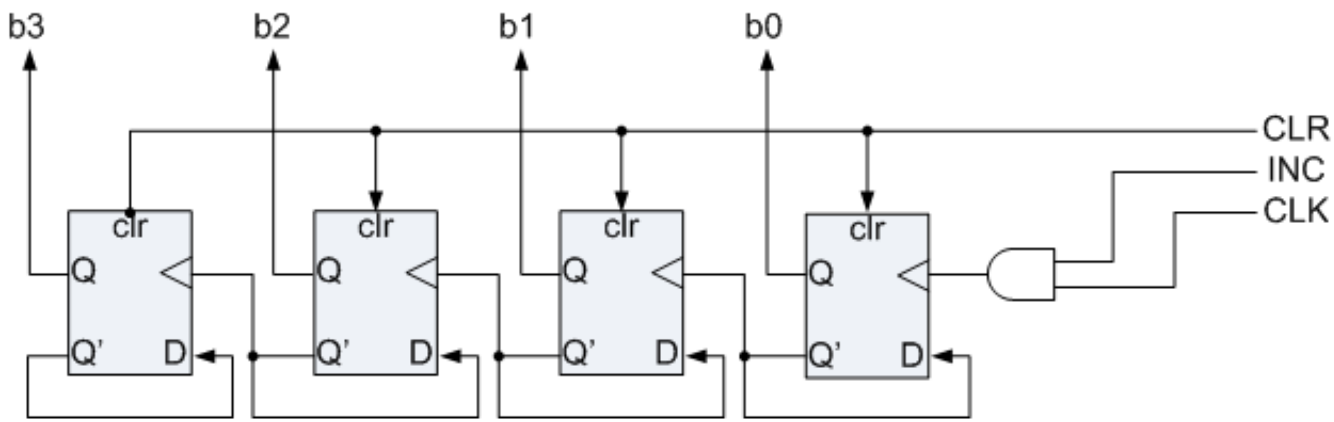
Example circuits and clocking

- Values of output bits ***must all be stable***
  - I.e., can't pulse the clock (increment) until all four bits are computed

- Adder circuit is ripple-carry: Must wait for carries
  - 4ns per adder
  - 4-bit adder
  - thus, 4 * 4ns = 16ns for the adder
- Flip-flops
  - Must wait for 1st latch of last bit to stabilize (others done in parallel)
  - Must wait for 2nd latch of all bits to stabilize (all done in parallel)
  - thus, 2ns + 2ns = 4ns
- Overall delay = 16ns + 4ns = 20ns. Clock pulse is 20ns.

Example circuits and clocking

Can we build a counter with just flip-flops?



What's the propagation delay?