CS/ECE 250: Computer Architecture

Basics of Logic Design:
ALU, Storage, Tristate

Benjamin Lee

Slides based on those from Alvin Lebeck, Daniel Sorin, Andrew Hilton, Amir Roth, Gershon Kedem

Administrivia

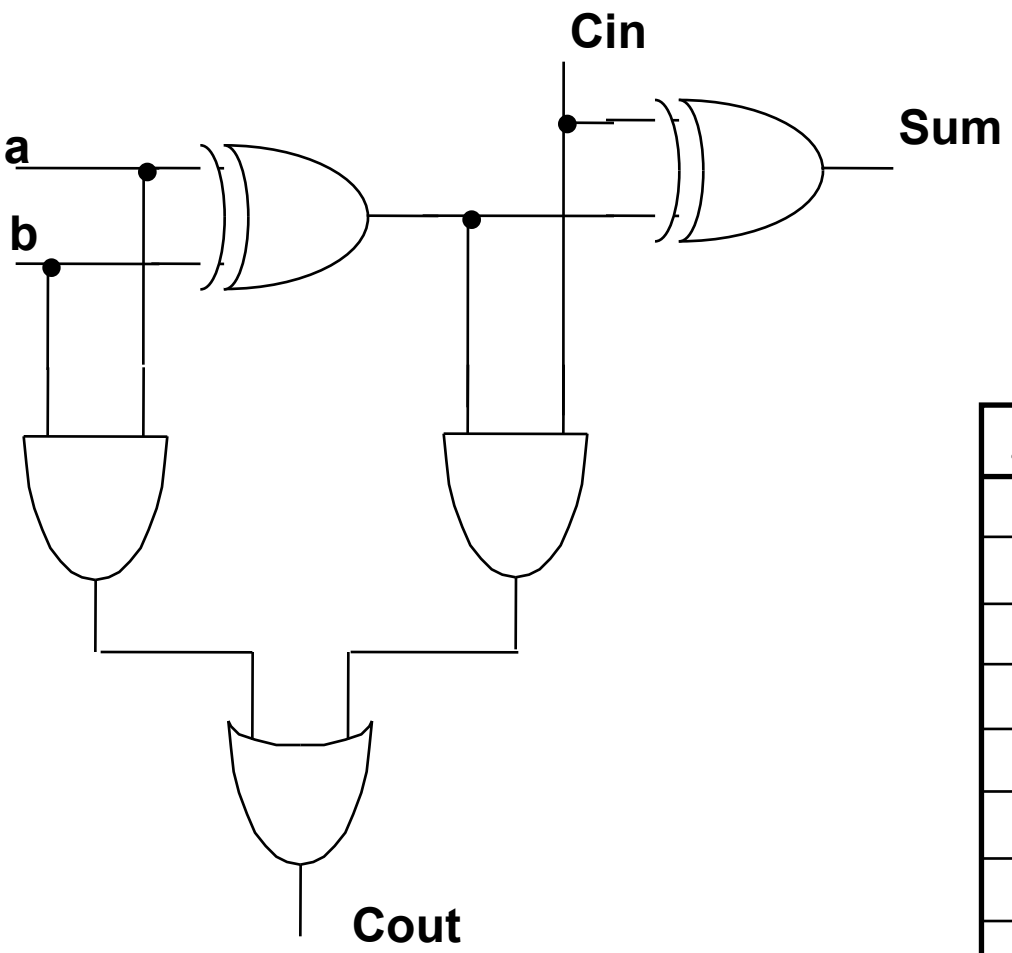
- Homework #3
- Due Mar 7, 11:55pm

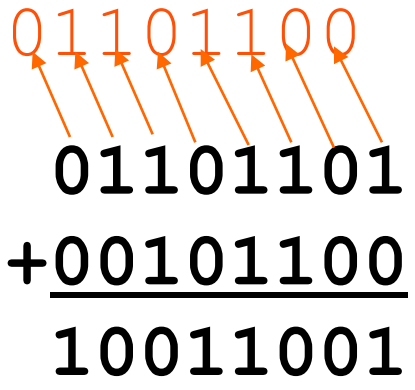
- Readings
  - Pragmatic Logic by William Eccles
  - Linked on web page, skim for what you need.
  - Combinational Circuits Ch 4.1-4.2, Ch 5.3
  - Sequential Circuits Ch 6

Arithmetic and Logical Operations in ISA

- **What operations are there?**
- **How do we implement them?**
  - **Consider a 1-bit Adder**

A 1-bit Full Adder

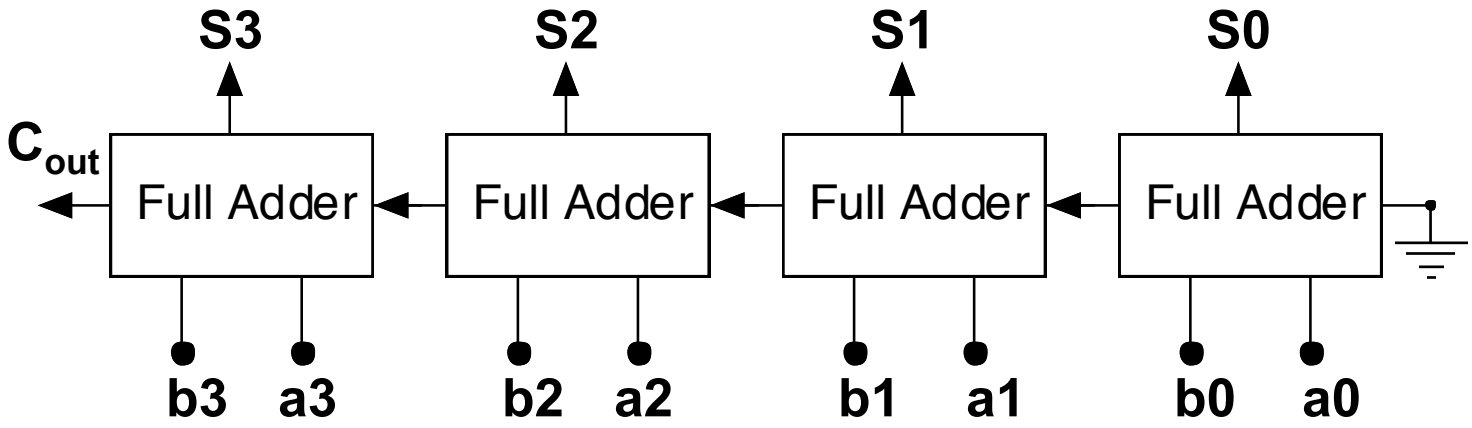




a	b	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

© Alvin Lebeck, from Hilton, Sorin

Example: 4-bit Ripple Carry Adder



Subtraction

- **How do we perform integer subtraction?**
- **What is the HW?**

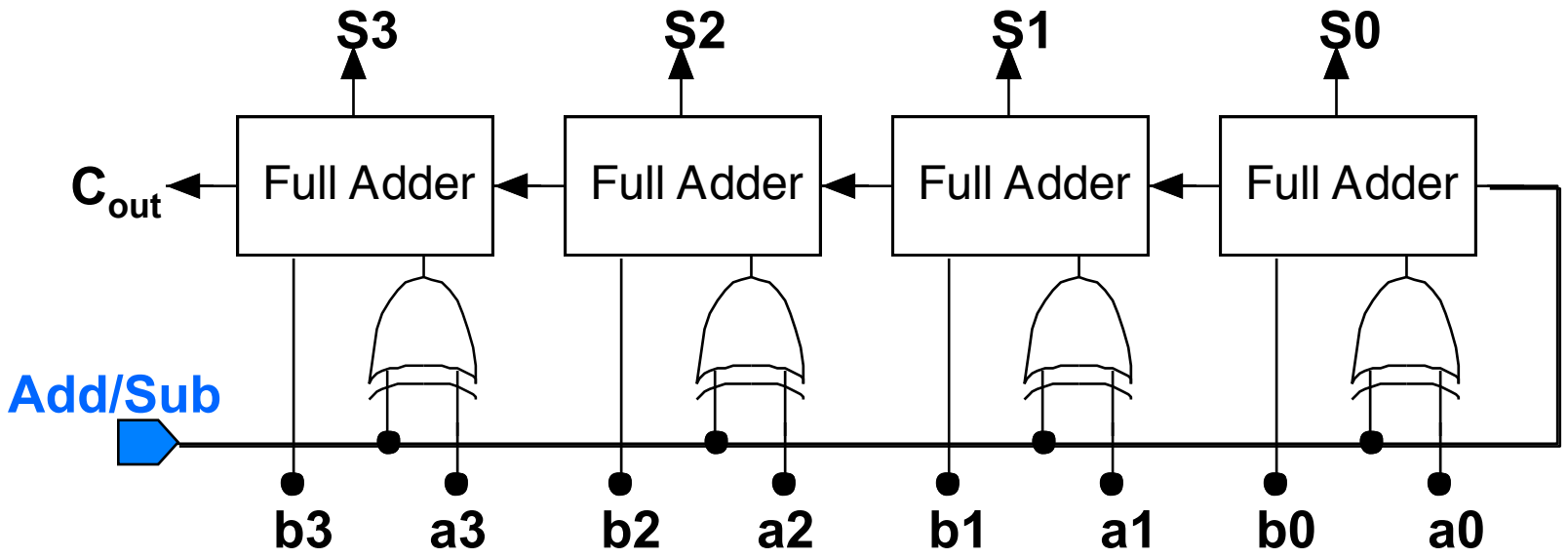
- **Remember: Subtraction is just addition**

  X – Y =

  X + (-Y) =

  X + (~Y +1)

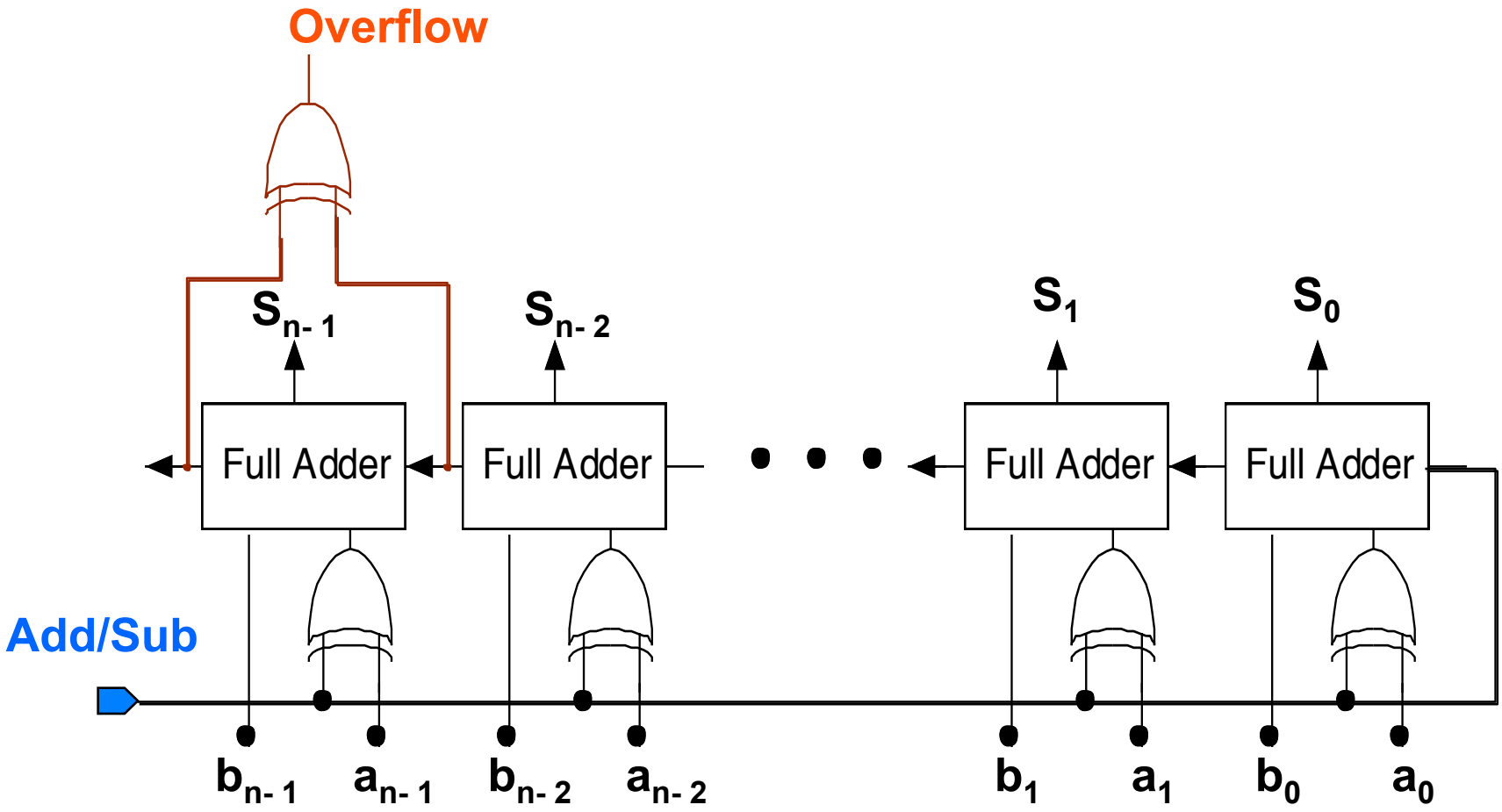
Example: Adder/Subtractor



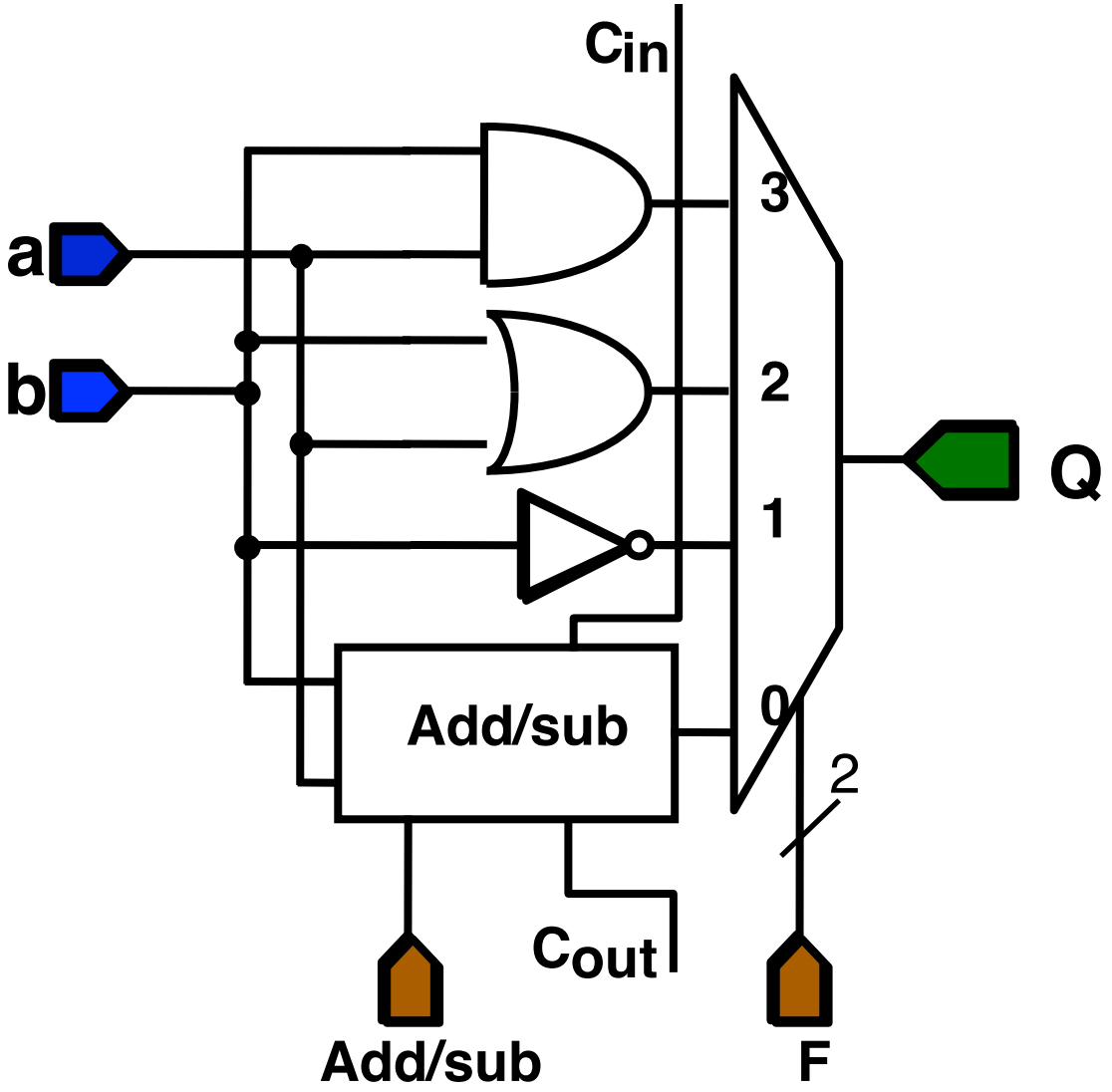
Overflow

- How would we detect signed overflow?
  - See if CI != CO
  - 1-bit != is implemented with XOR
- If CI = 0 and CO=1
  - Sum must produce the carry
  - CO=1 only if A=1 and B=1
  - Adding two negative numbers to produce positive number
- If CI = 1 and CO=0
  - Sum must consume the carry
  - CO=0 only if A=0 and B=0
  - Adding two positive numbers to produce a negative number

Add/Subtract With Overflow Detection



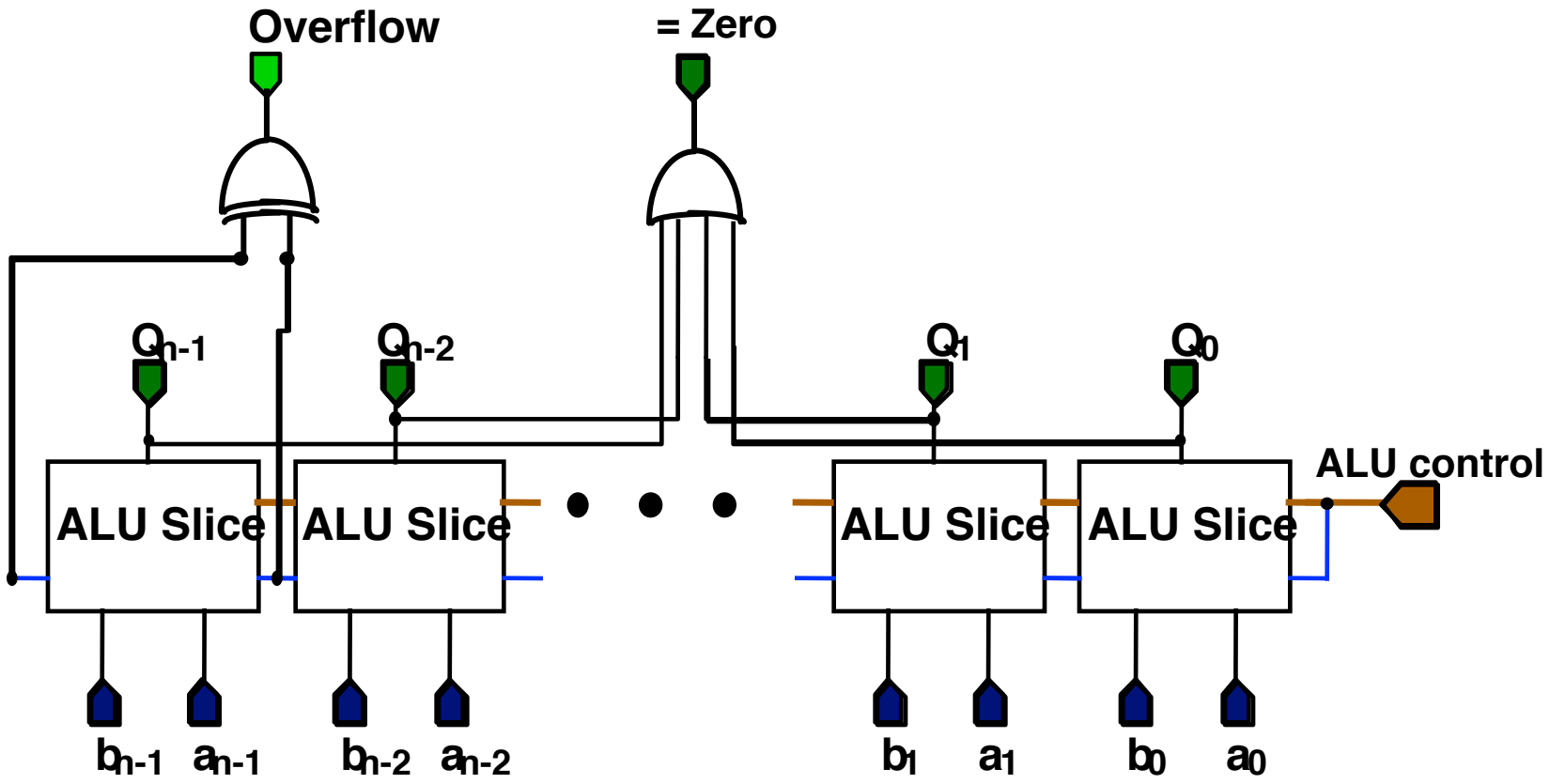
ALU Slice



A	F	Q
0	0	a + b
1	0	a - b
-	1	NOT b
-	2	a OR b
-	3	a AND b

© Alvin Lebeck, from Hilton, Sorin

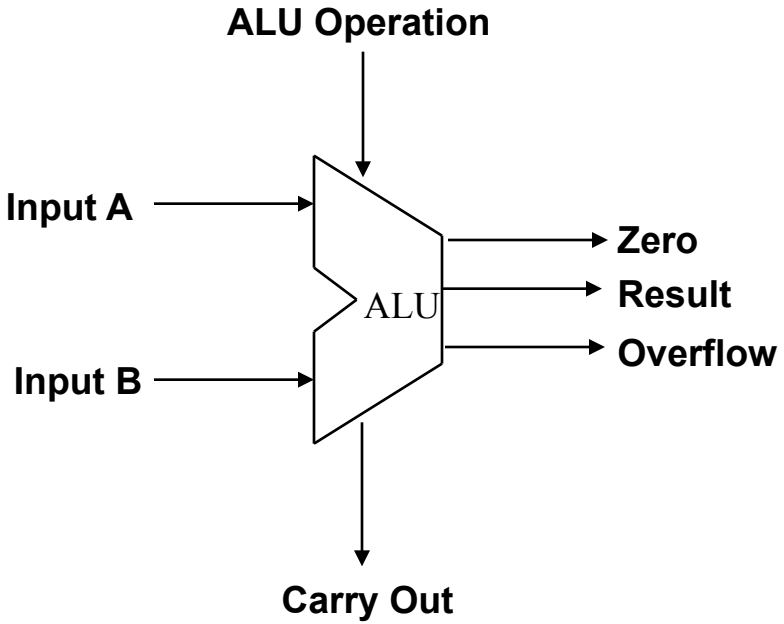
The ALU



© Alvin Lebeck, from Hilton, Sorin

Abstraction: The ALU

- General structure
- Two operand inputs
- Control inputs



- We can build circuits for
  - Multiplication
  - Division
  - They are more complex

© Alvin Lebeck, from Hilton, Sorin

Shifts

- **Remember the << and >> operations?**
  - **Shift left/shift right?**
  - **How would we implement these?**

Suppose an 8-bit number

$b_7b_6b_5b_4b_3b_2b_1b_0$

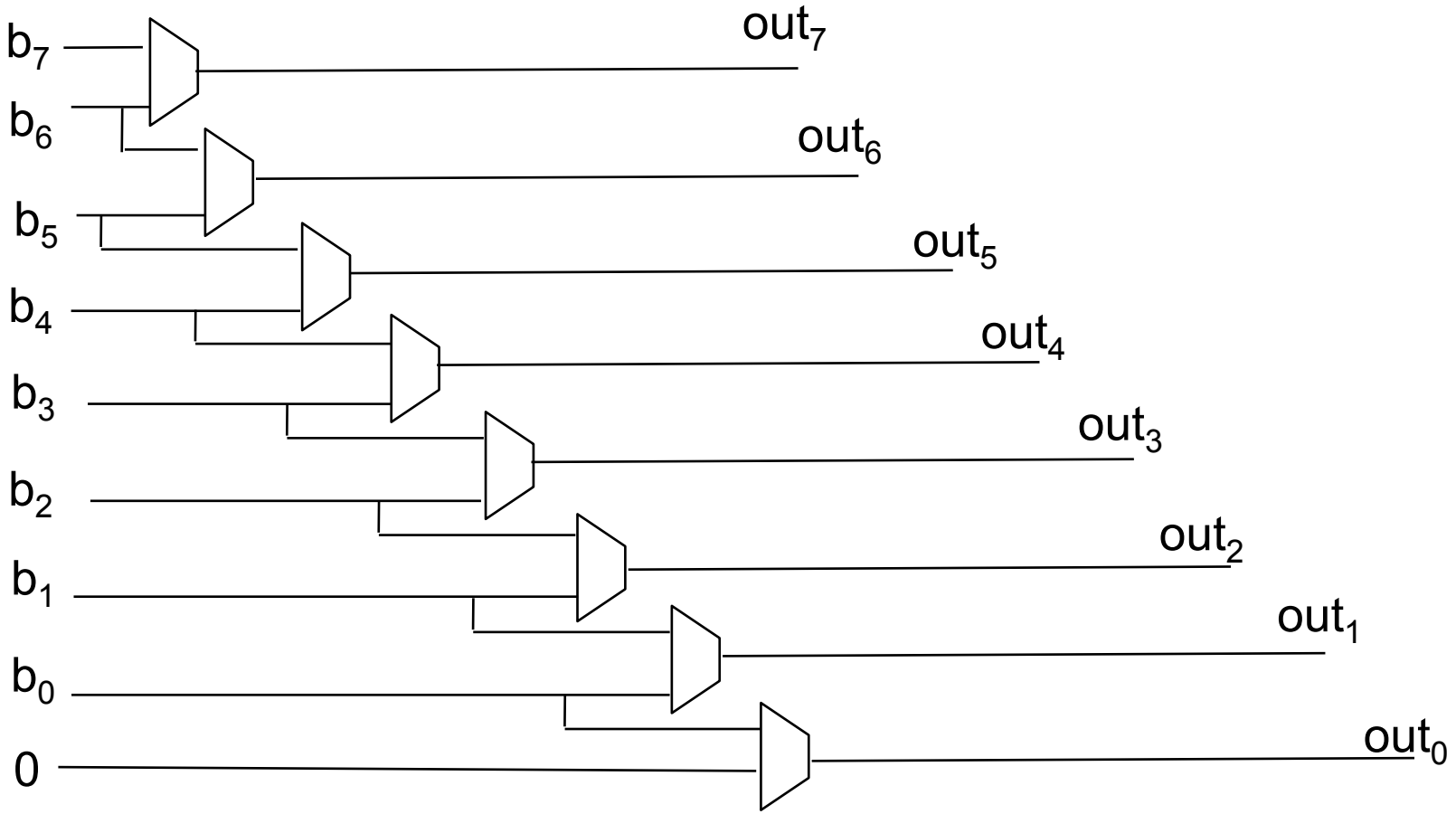
Shifted left by a 3 bit number

$s_2s_1s_0$

- **Option 1: Truth Table?**
  - **2048 rows? Not appealing**

Let's simplify

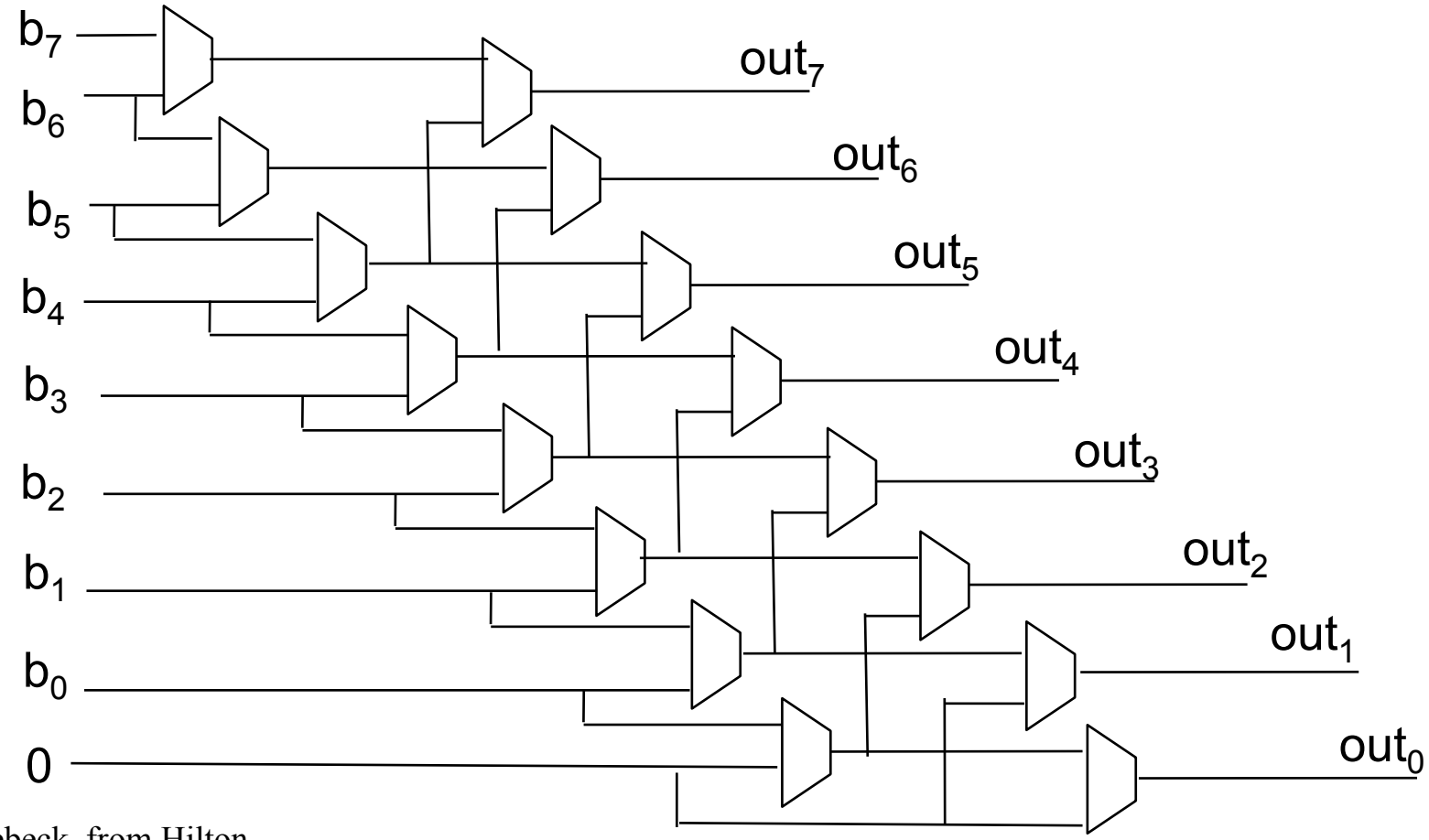
- **Simpler problem: 8-bit number shifted by 1 bit number (shift amount selects each mux)**



© Alvin Lebeck, from Hilton, Sorin

Let's simplify

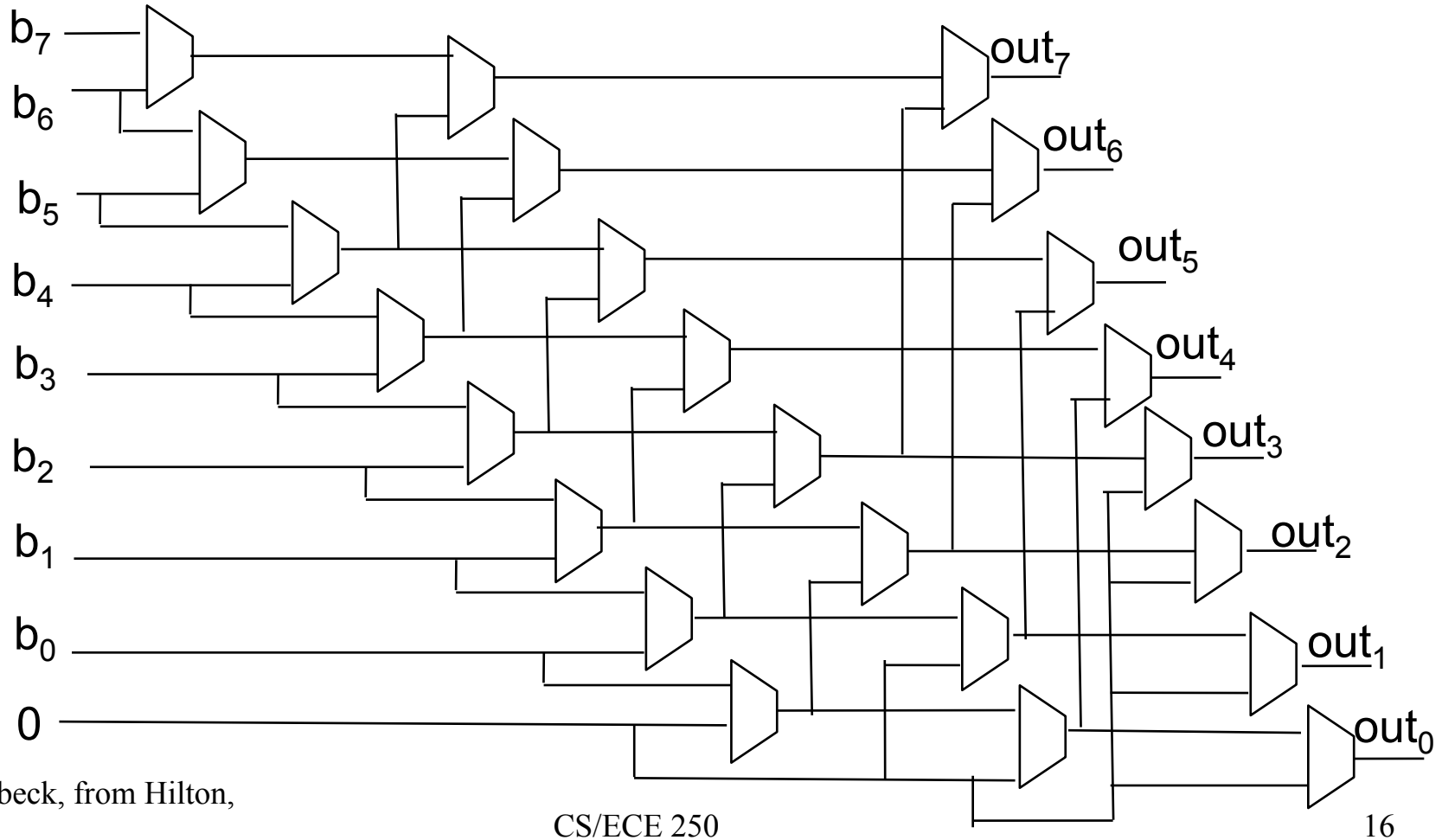
- **Simpler problem: 8-bit number shifted by 2 bit number (new muxes selected by 2nd bit)**



© Alvin Lebeck, from Hilton, Sorin

Now shifted by 3-bit number

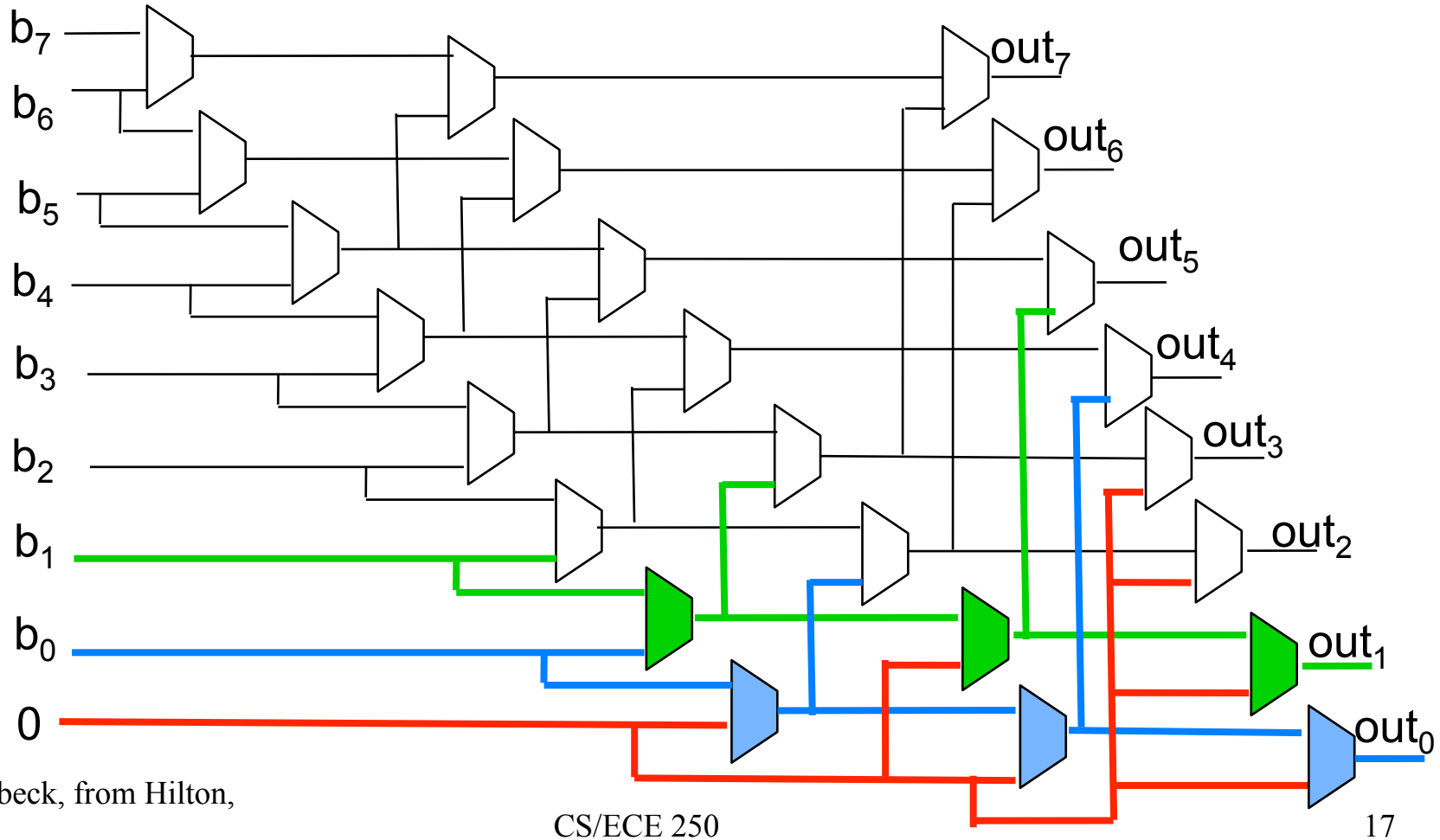
- **Full problem: 8-bit number shifted by 3 bit number (new muxes selected by 3rd bit)**



© Alvin Lebeck, from Hilton, Sorin

Now shifted by 3-bit number

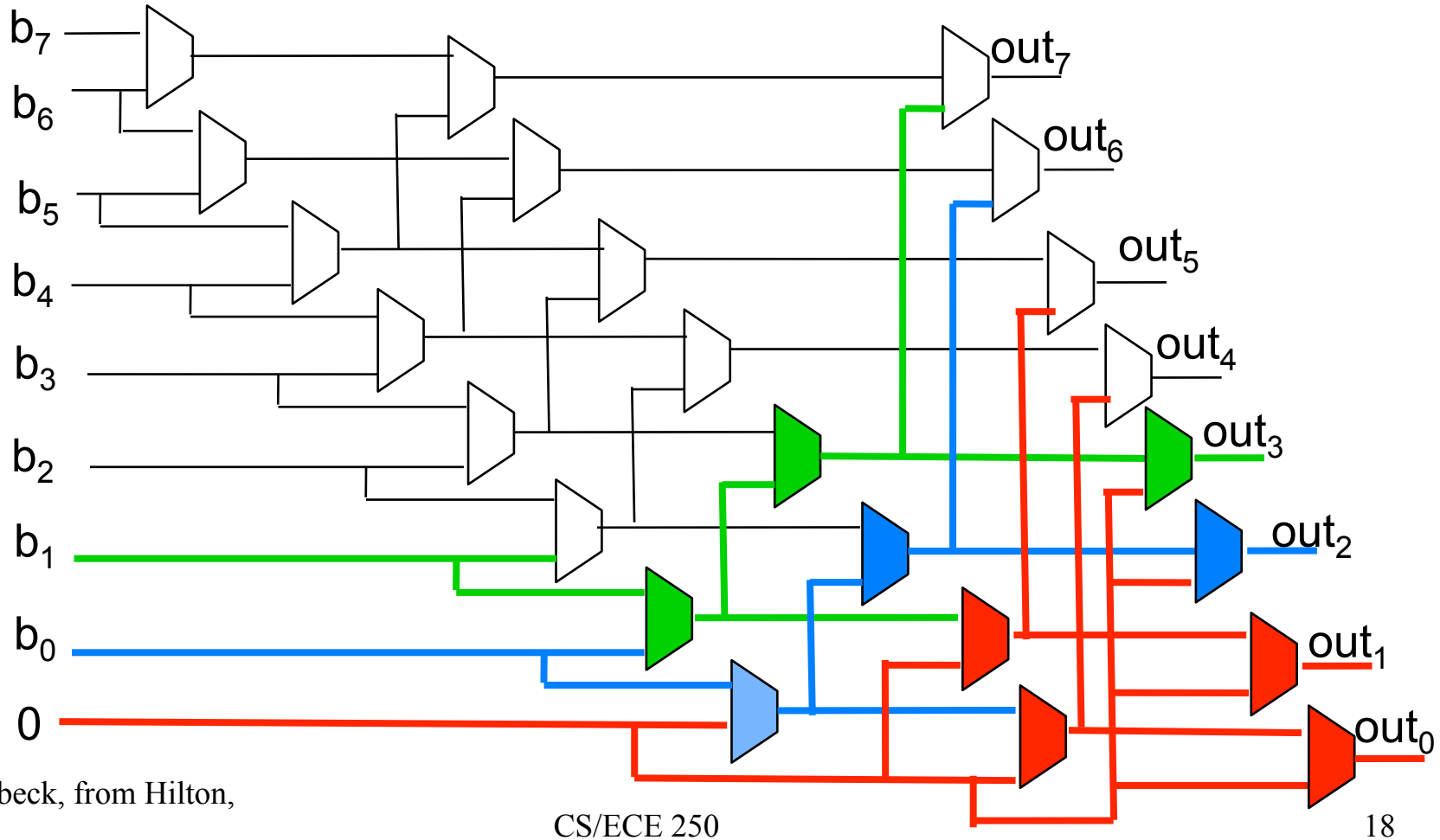
- **Shifter in action: shift by 000**



© Alvin Lebeck, from Hilton, Sorin

Now shifted by 3-bit number

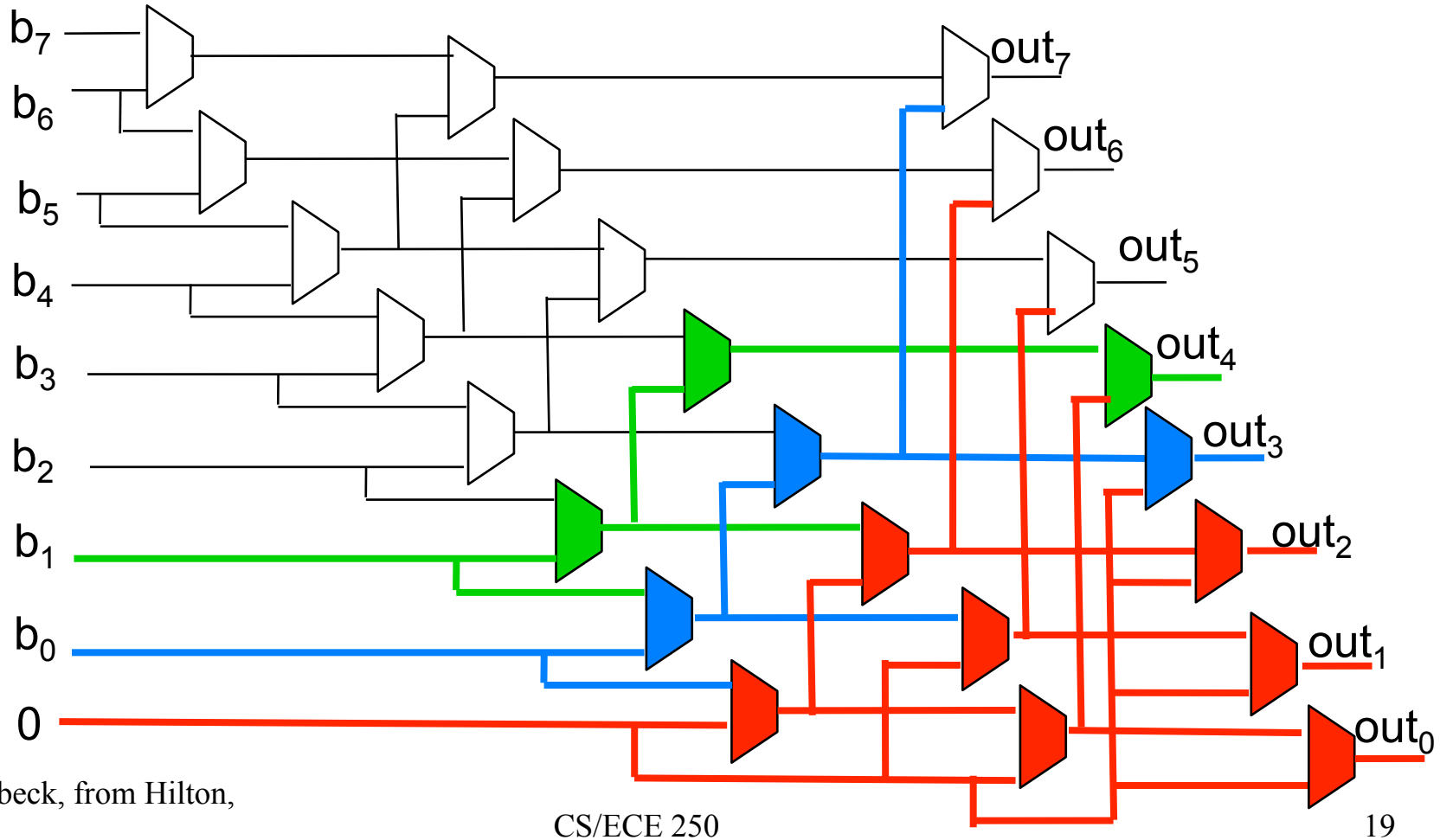
- **Shifter in action: shift by 010**



© Alvin Lebeck, from Hilton, Sorin

Now shifted by 3-bit number

- **Shifter in action: shift by 011**



© Alvin Lebeck, from Hilton, Sorin

So far…

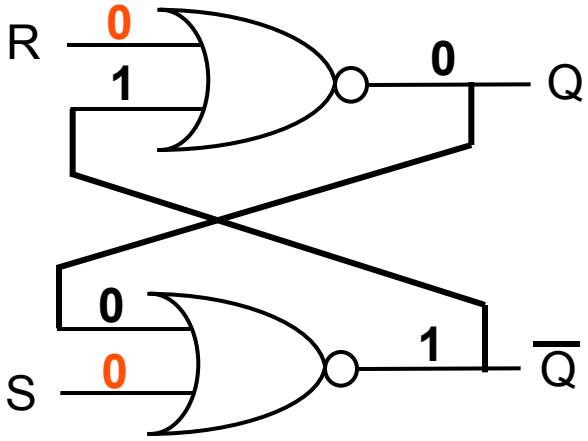
- We can make logic to compute "math"
  - Add, subtract,… (we'll see multiply/divide later)
  - Bitwise: AND, OR, NOT,…
  - Shifts
  - Selection (MUX)

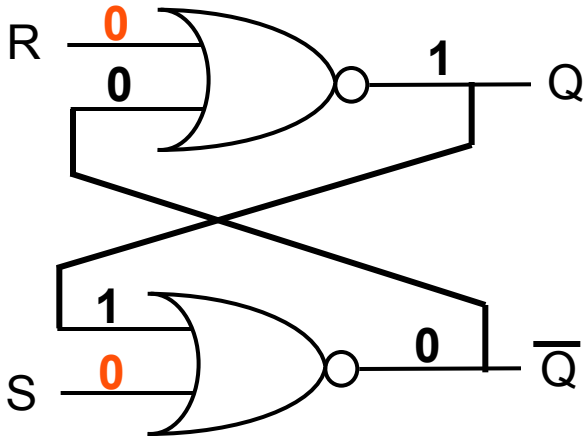
- But processors need state (hold value)
  - Registers

Memory Elements

- All the circuits we looked at so far are combinational circuits: the output is a Boolean function of the inputs.
- We need circuits that can remember values (registers, memory)
- The output of the circuit is a function of the input and a stored value (state)
- Circuits with memory are called sequential circuits
- Key to storage: loops in circuit from outputs to inputs

NOR-based Set-Reset (SR) Latch



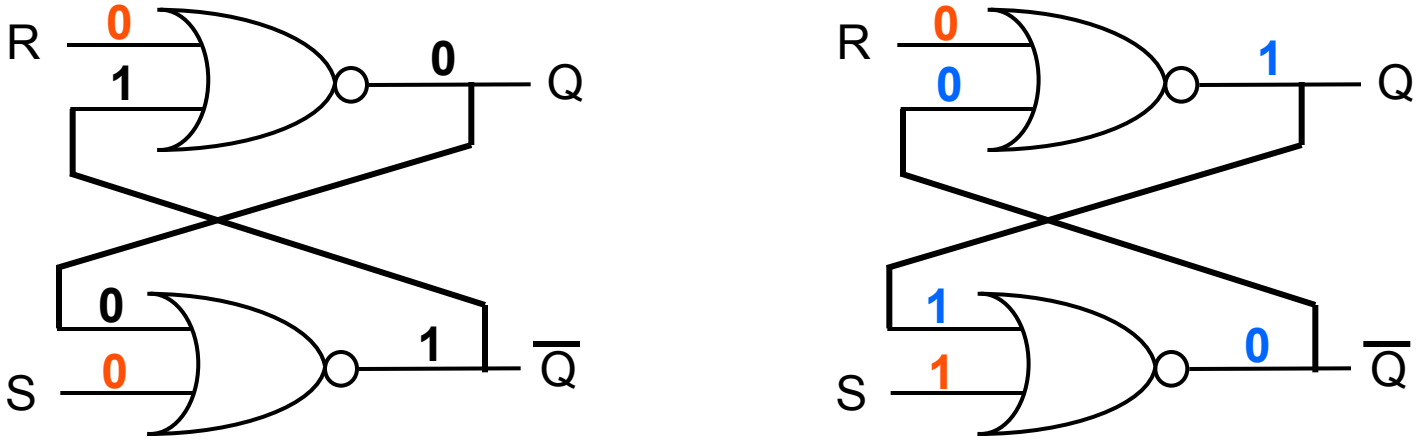


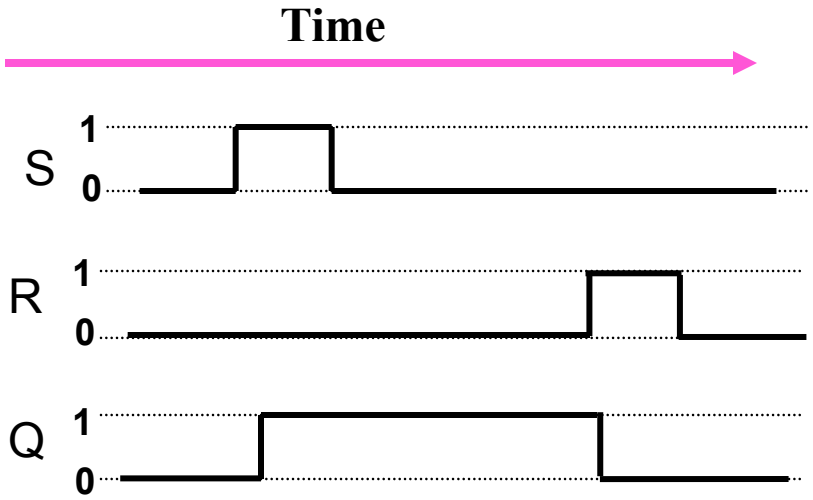
R	S	Q
0	0	Q
0	1	1
1	0	0
1	1	–

Don't set both S & R to 1.
Seriously, don't do it.

© Alvin Lebeck, from Hilton, Sorin

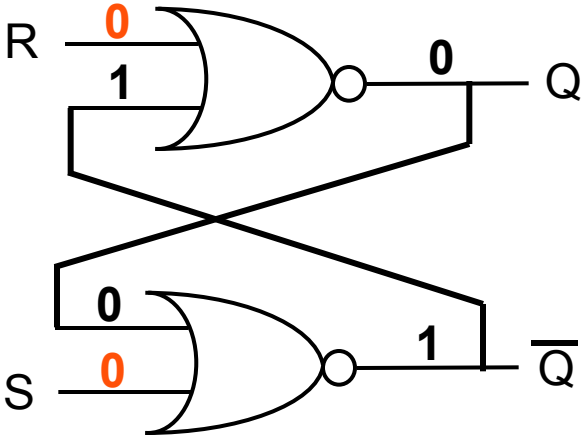
Set-Reset Latch (Continued)

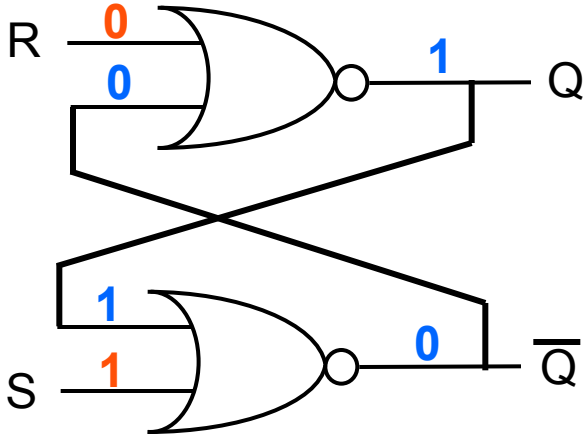


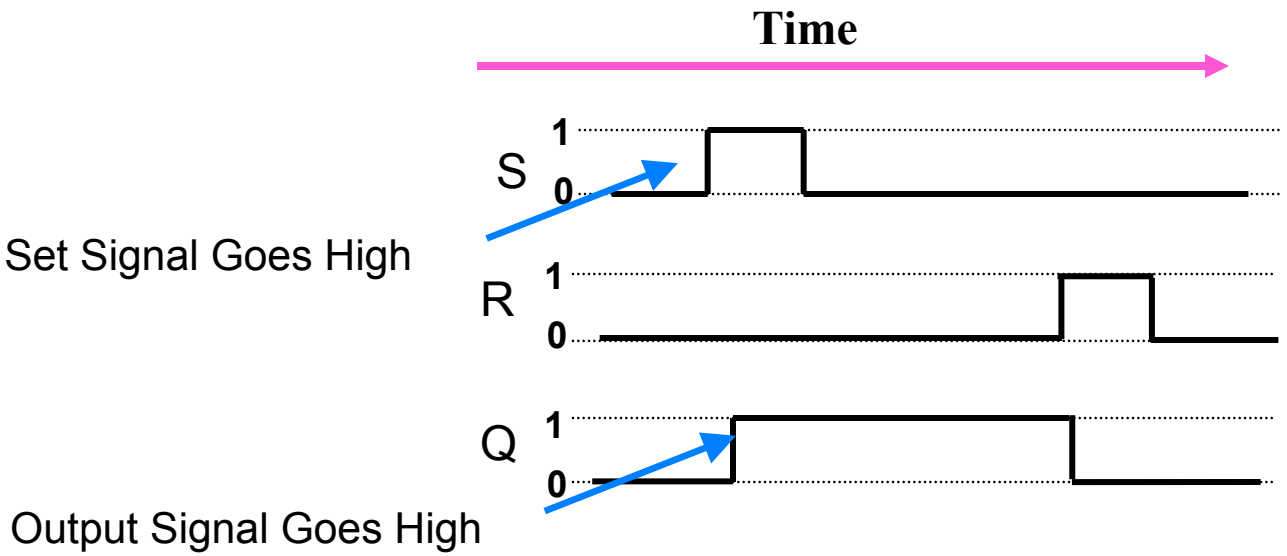


© Alvin Lebeck, from Hilton, Sorin

Set-Reset Latch (Continued)

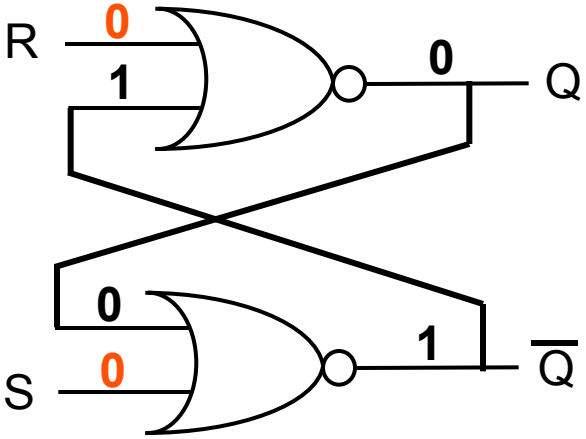


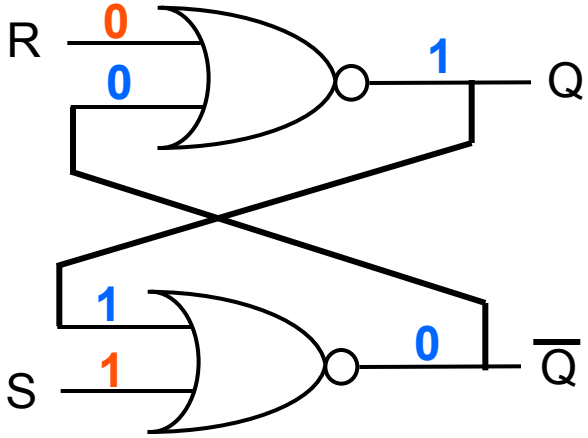


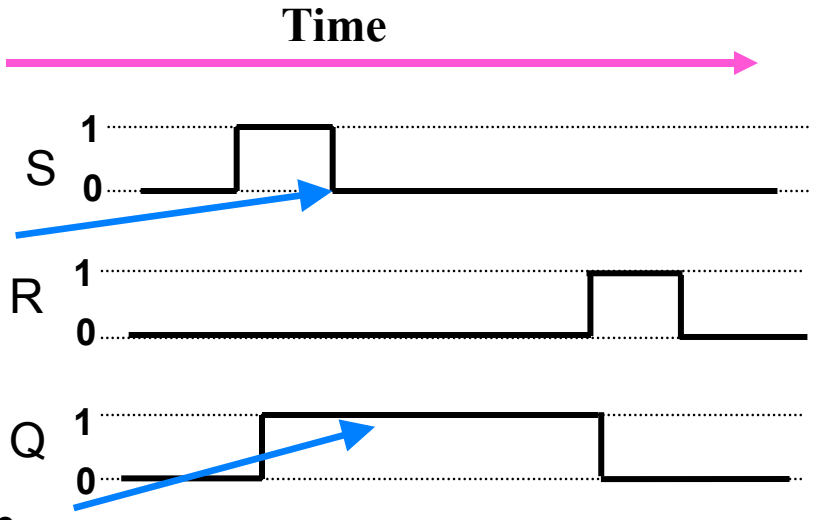


© Alvin Lebeck, from Hilton, Sorin

Set-Reset Latch (Continued)





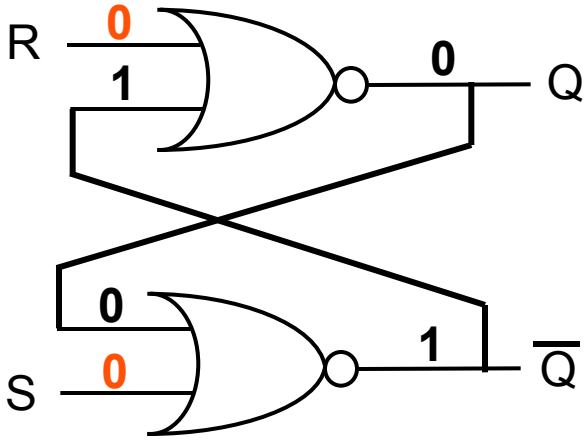


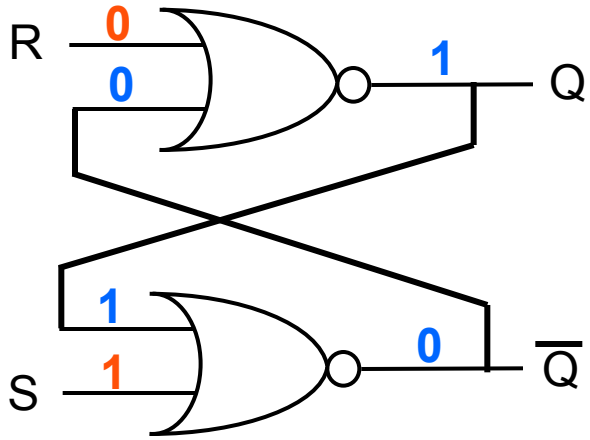
Set Signal Goes Low

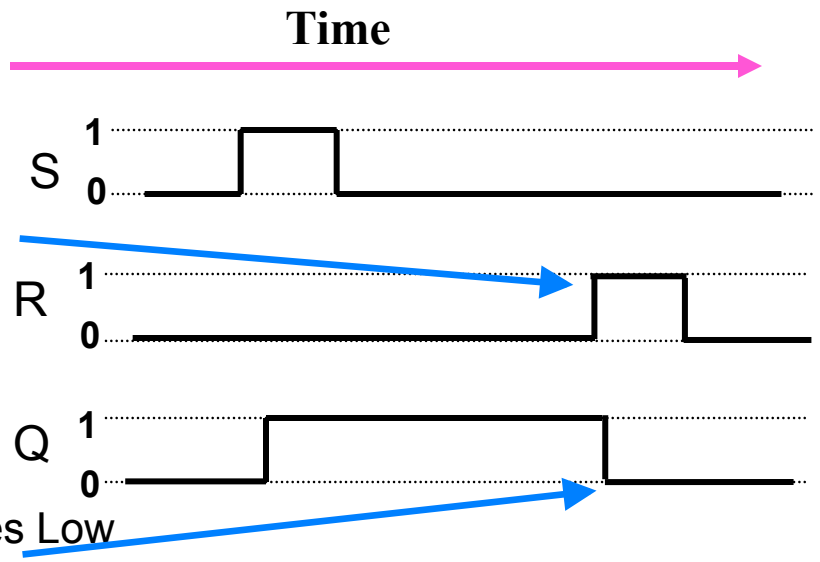
Output Signal Stays High

© Alvin Lebeck, from Hilton, Sorin

Set-Reset Latch (Continued)







Until Reset Signal Goes High

Then Output Signal Goes Low

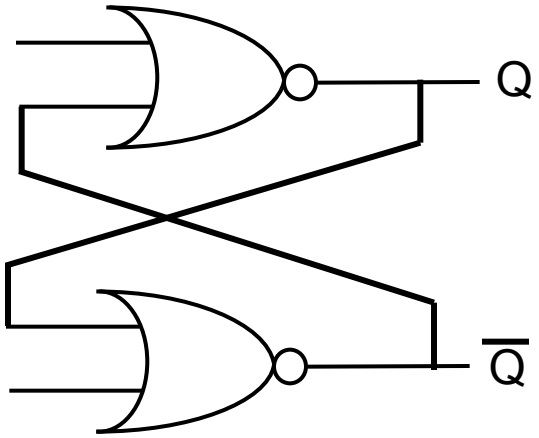
© Alvin Lebeck, from Hilton, Sorin

SR Latch

- Downside: S and R at once = chaos
- Downside: Bad interface

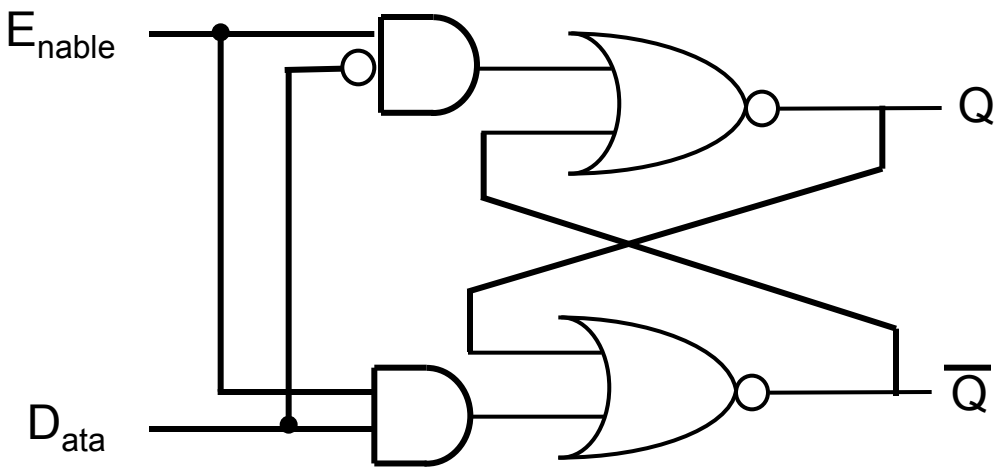
- What is a better solution?

Data Latch (D Latch)



Starting with SR Latch

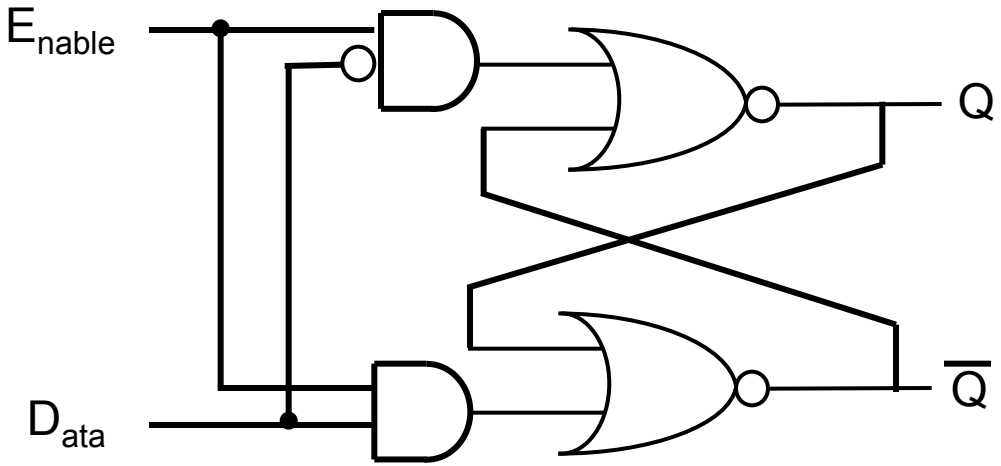
Data Latch (D Latch)



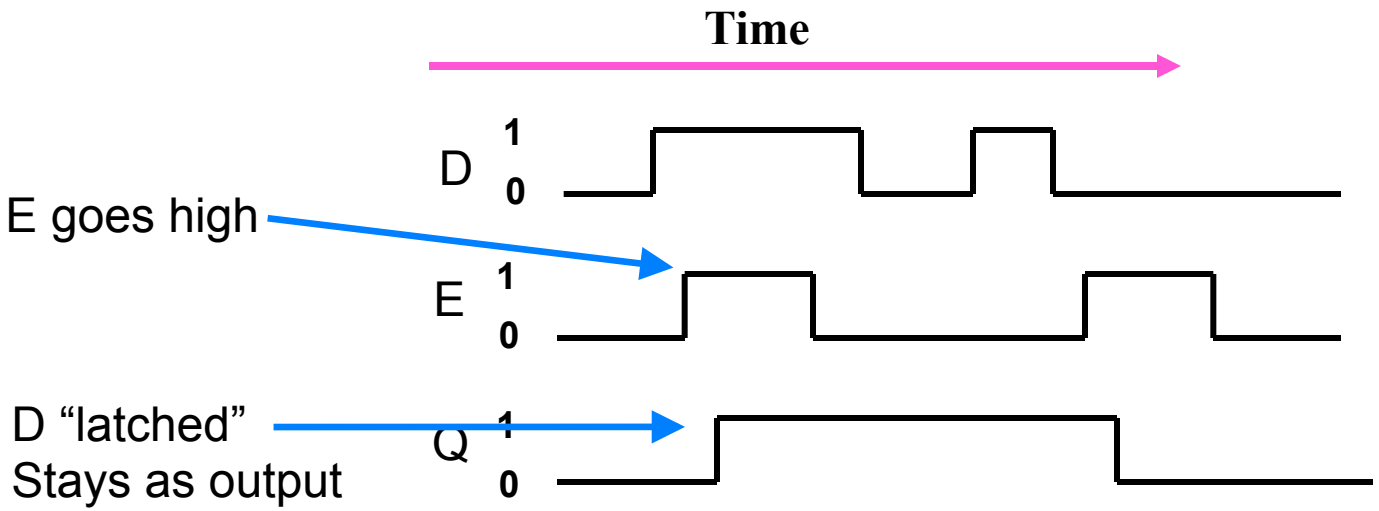
Starting with SR Latch

Change interface to
Data + Enable (D + E)

Data Latch (D Latch)

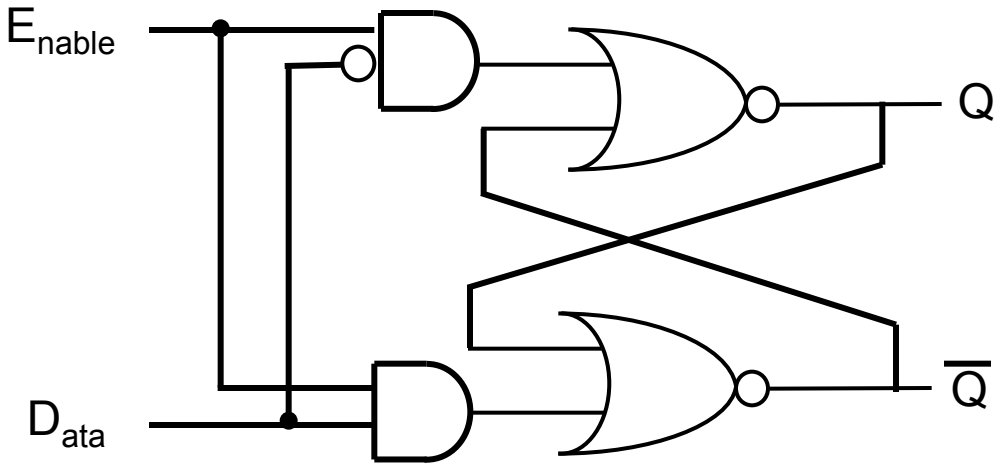


D	E	Q
0	1	0
1	1	1
-	0	Q

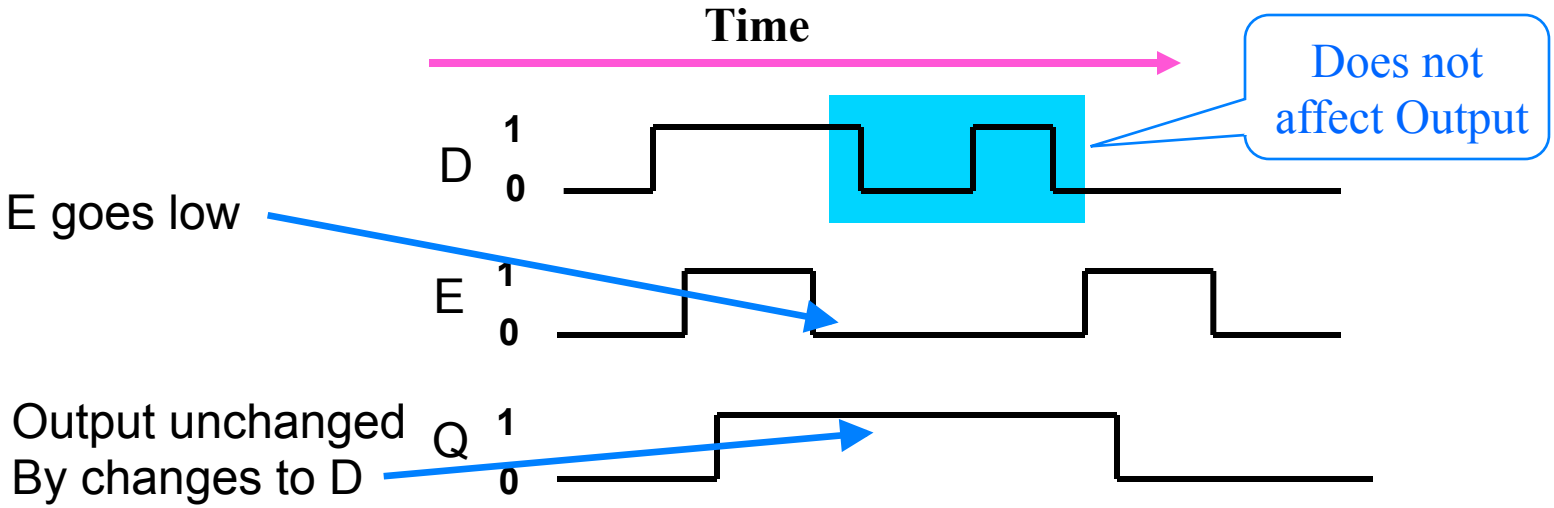


© Alvin Lebeck, from Hilton, Sorin

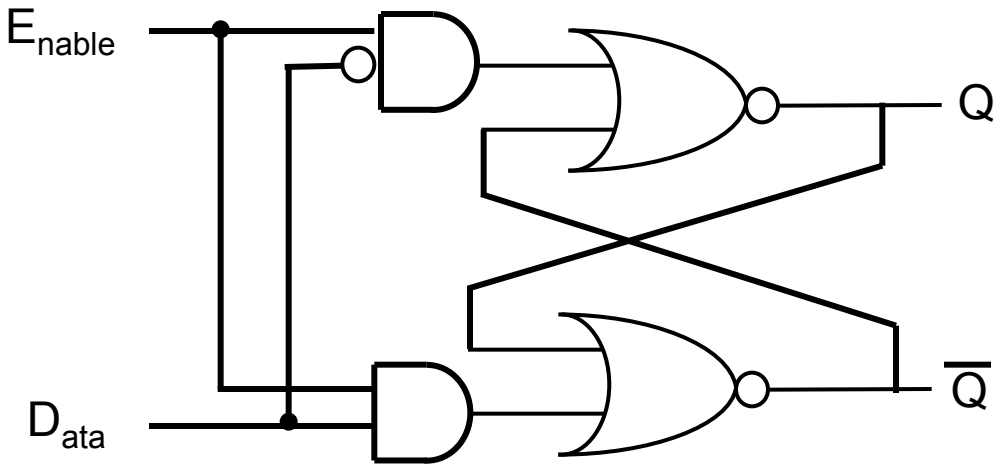
Data Latch (D Latch)



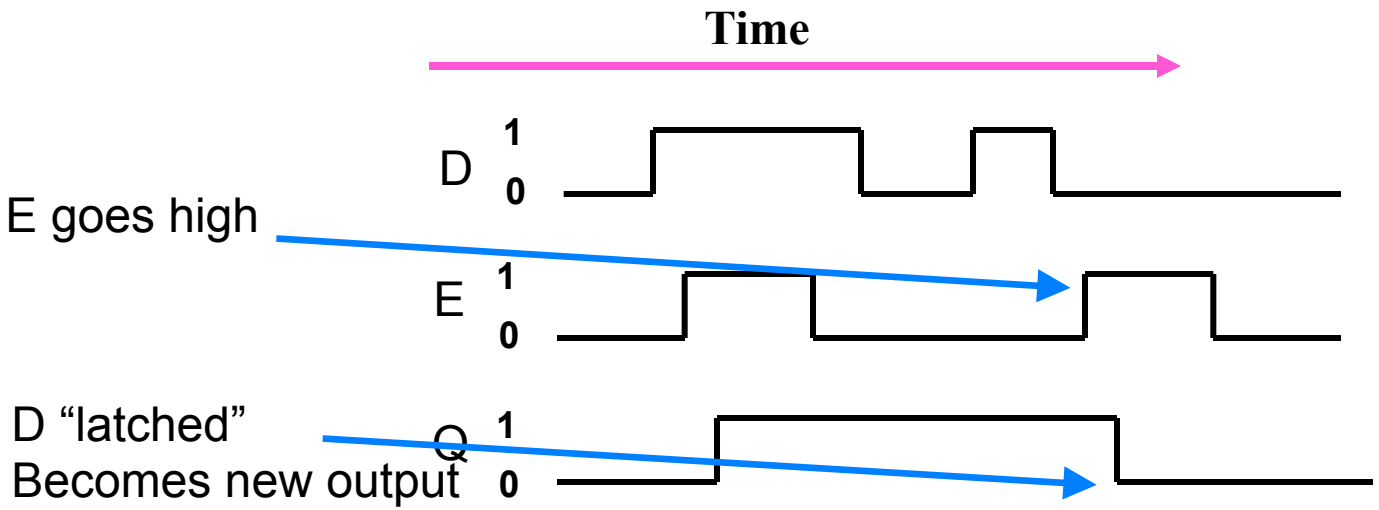
D	E	Q
0	1	0
1	1	1
-	0	Q



Data Latch (D Latch)

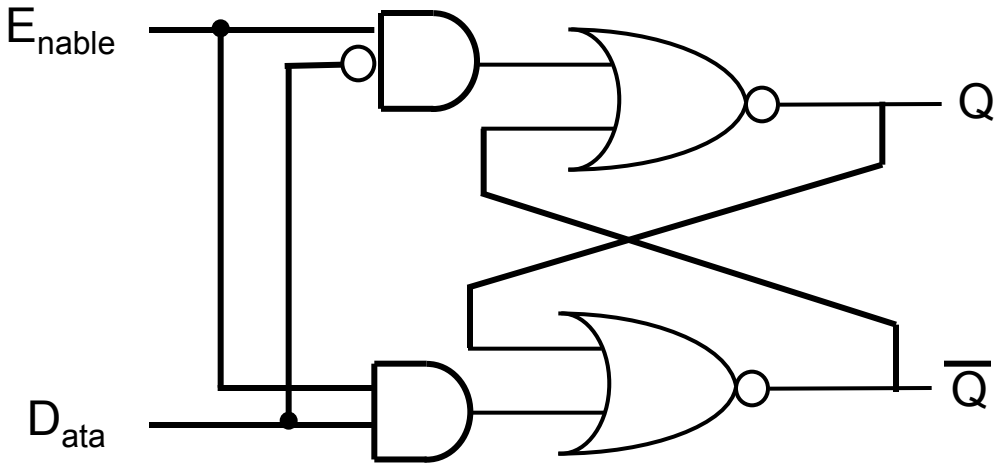


D	E	Q
0	1	0
1	1	1
-	0	Q

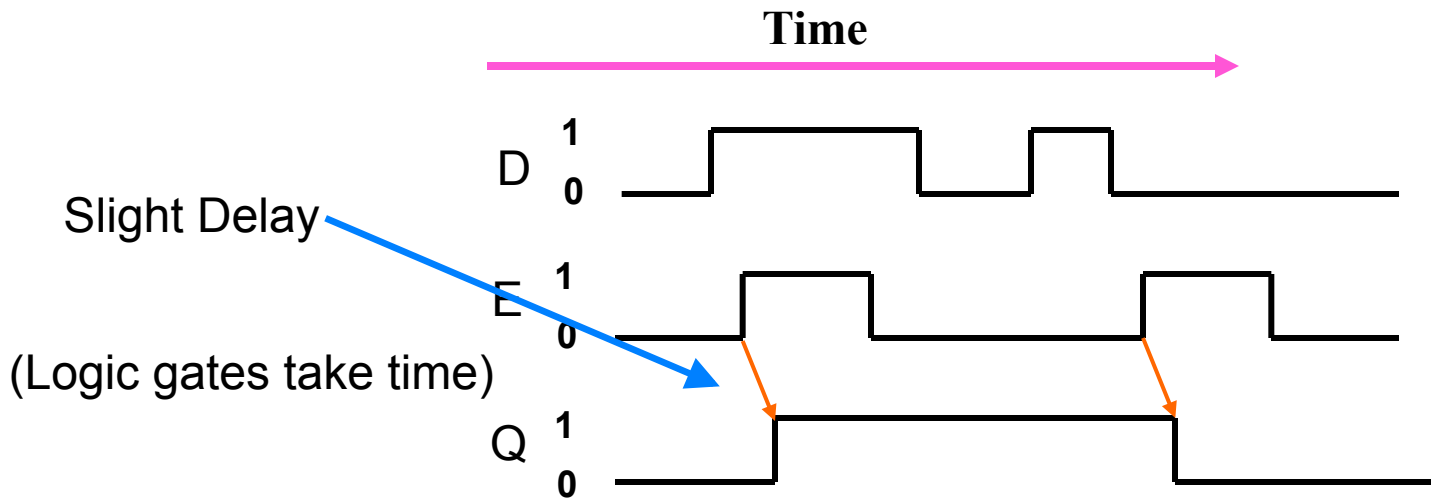


© Alvin Lebeck, from Hilton, Sorin

Data Latch (D Latch)



D	E	Q
0	1	0
1	1	1
-	0	Q



© Alvin Lebeck, from Hilton, Sorin

Logic Takes Time

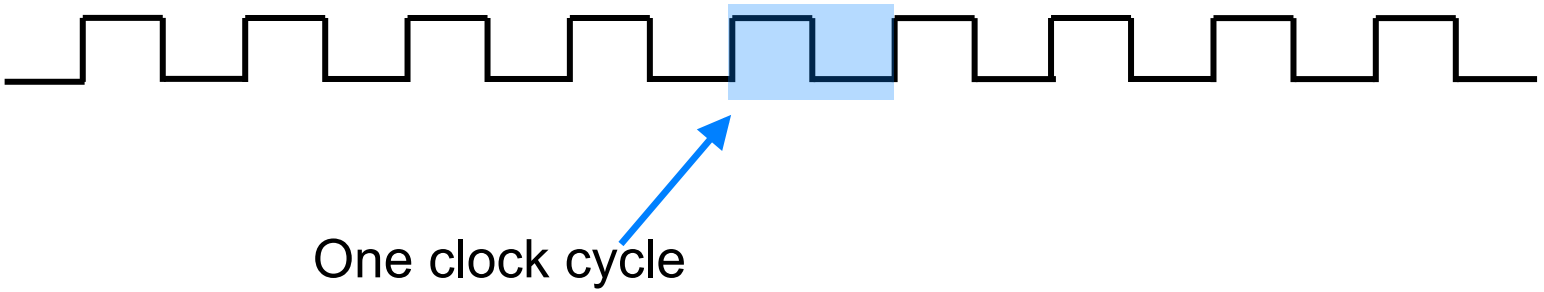
- Logic takes time:
  - Gate delays: delay to switch each gate
  - Wire delays: delay for signal to travel down wire
  - Other factors (not going into them here)

- Need to make sure that signals timing is right
  - Don't want to have races

© Alvin Lebeck, from Hilton, Sorin

Clocks

- **Processors have a clock:**
  - **Alternates 0 1 0 1 (low high low high)**
  - **Latch → logic → latch in one clock cycle**

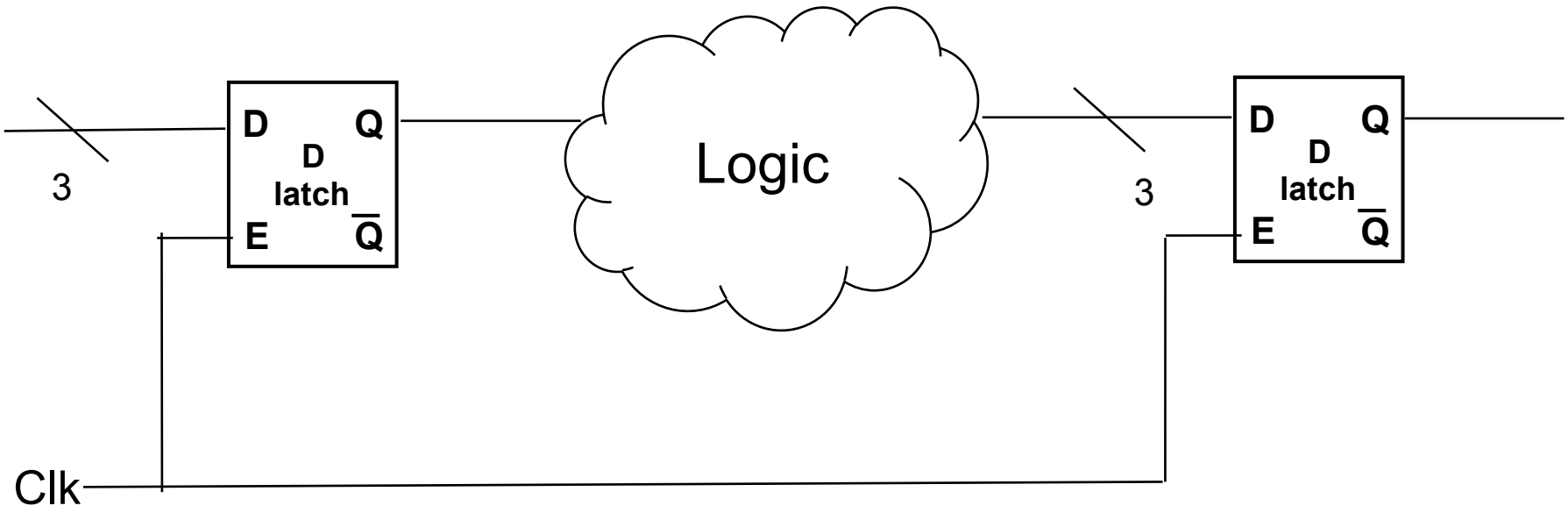


  - **3.4 GHz processor = 3.4 Billion clock cycles/sec**

© Alvin Lebeck, from Hilton, Sorin

Level Triggered Clock

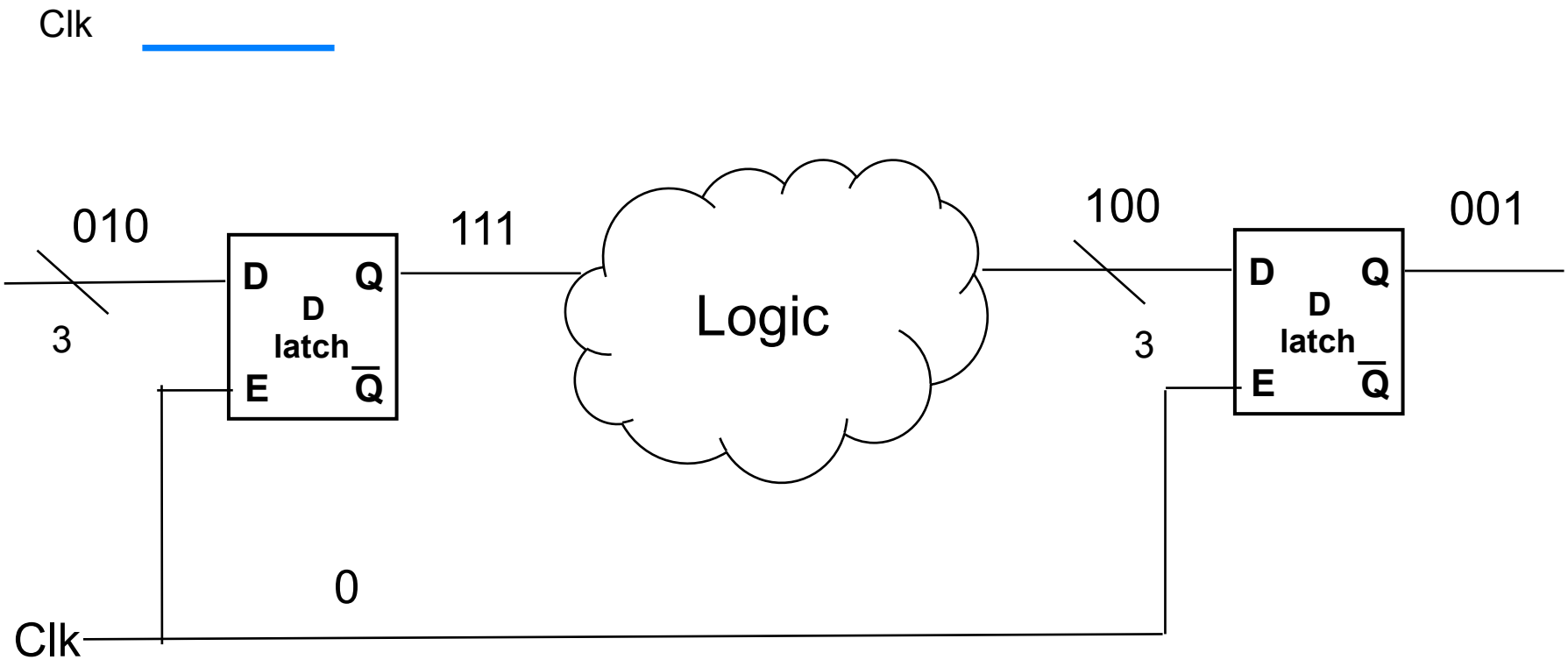
- **First thoughts: Level Triggered**
  - **Latch enabled when clock is high**
  - **Hold value when clock is low**



© Alvin Lebeck, from Hilton, Sorin

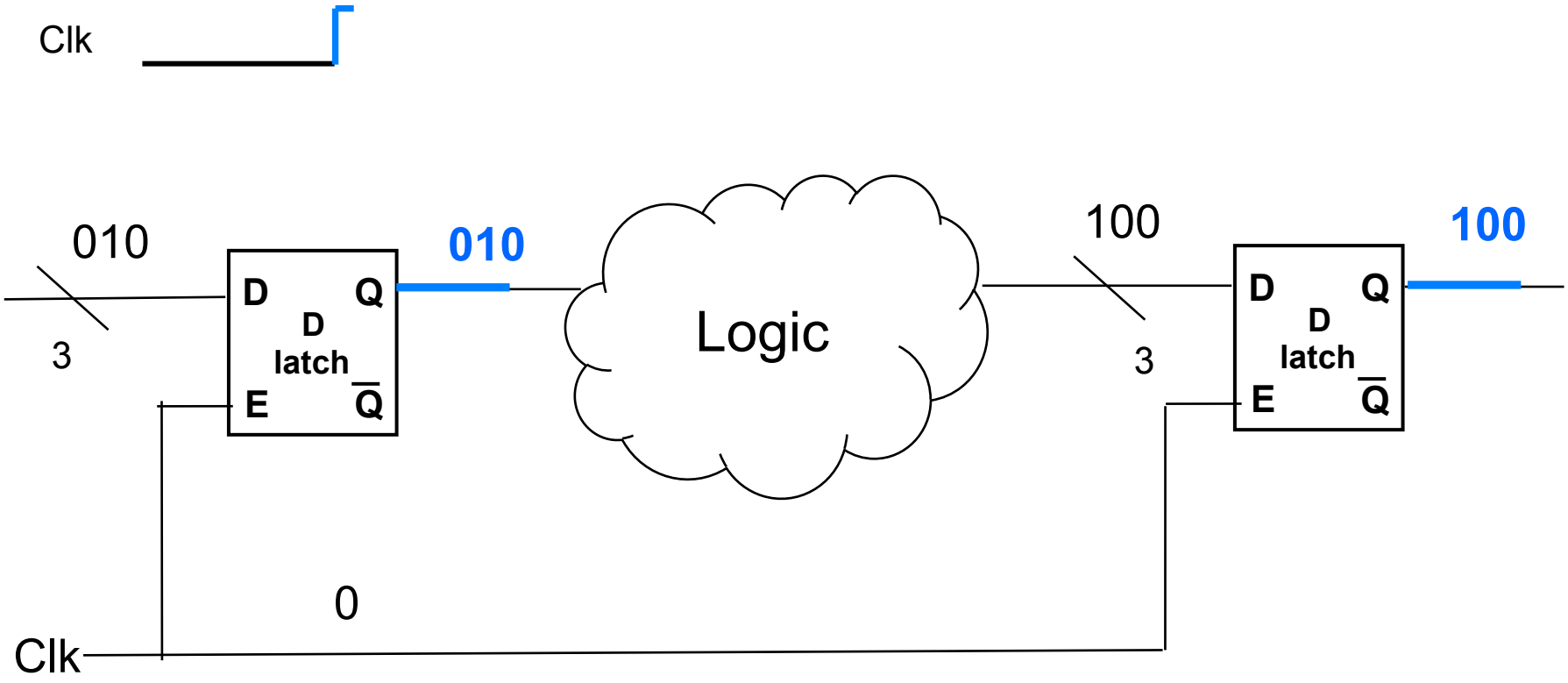
Level Triggered Clock

- **How we'd like this to work**
  - **Clock is low, all values stable**



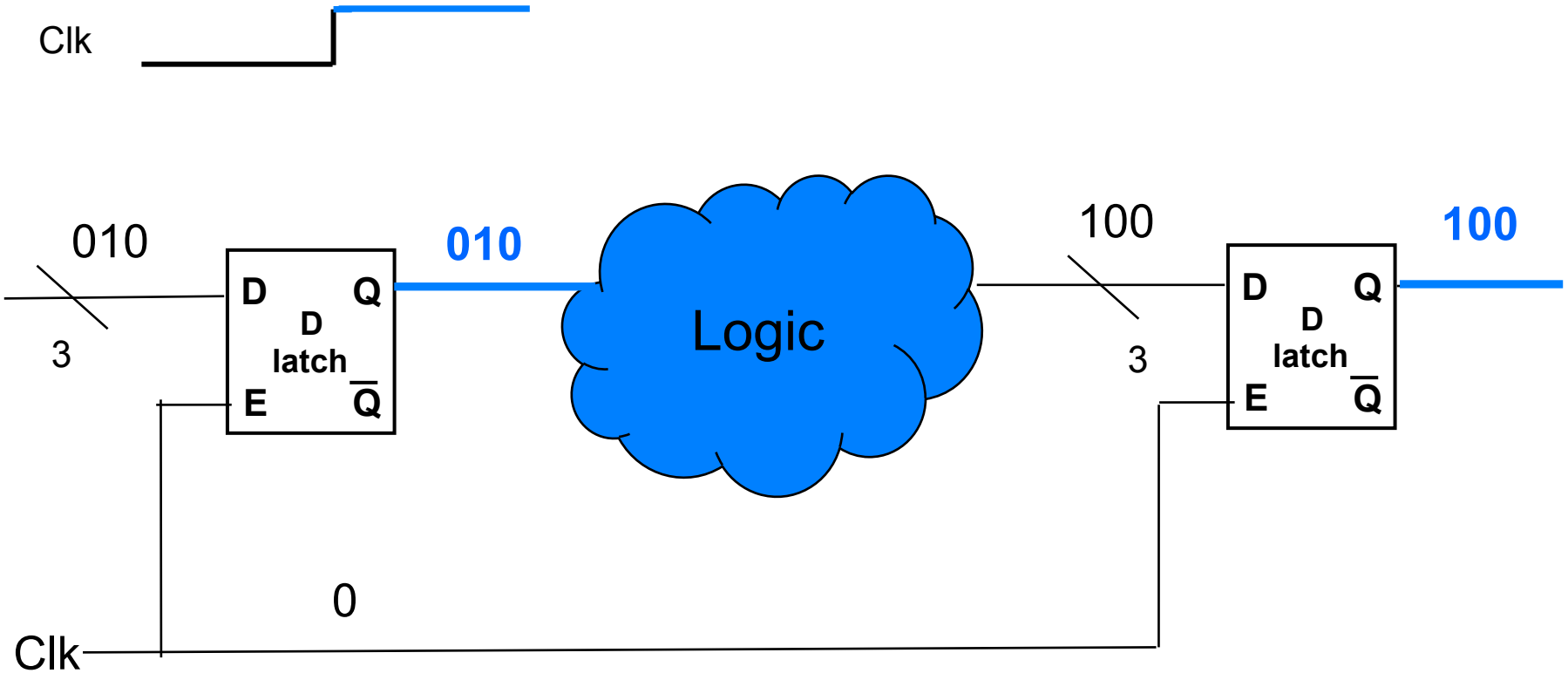
Level Triggered Clock

- **How we'd like this to work**
  - **Clock goes high, latches capture and transmit new value**



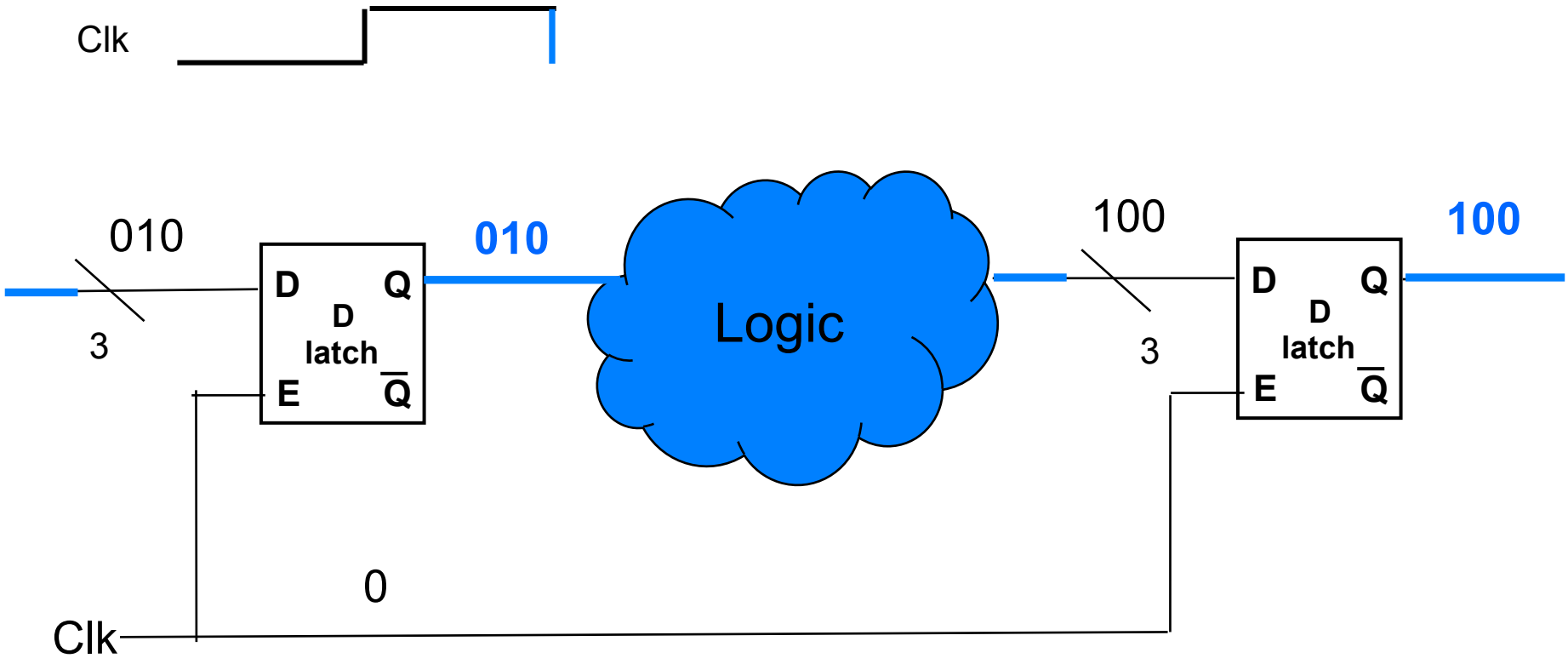
Level Triggered Clock

- **How we'd like this to work**
  - **Signals work their way through logic w/ high clk**



Level Triggered Clock

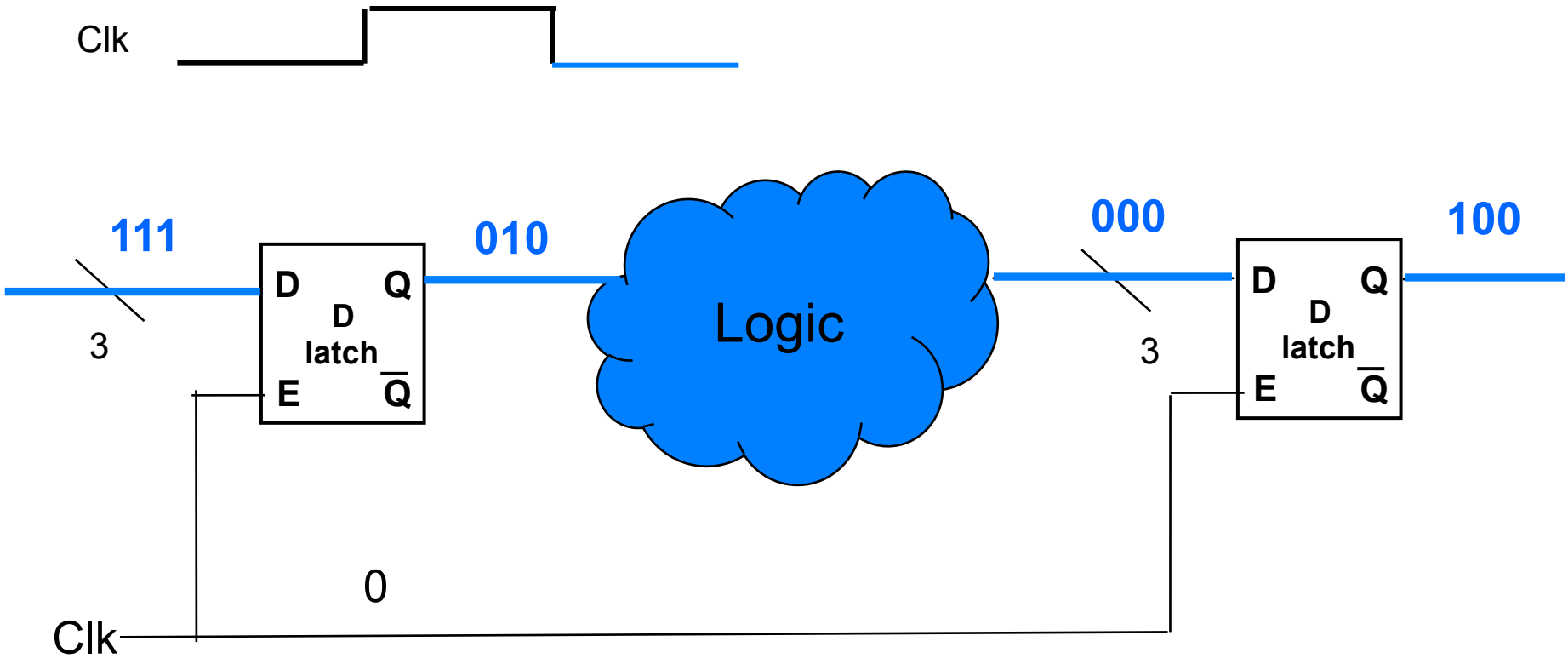
- **How we'd like this to work**
  - **Clock goes low before signals reach next latch**



© Alvin Lebeck, from Hilton, Sorin

Level Triggered Clock

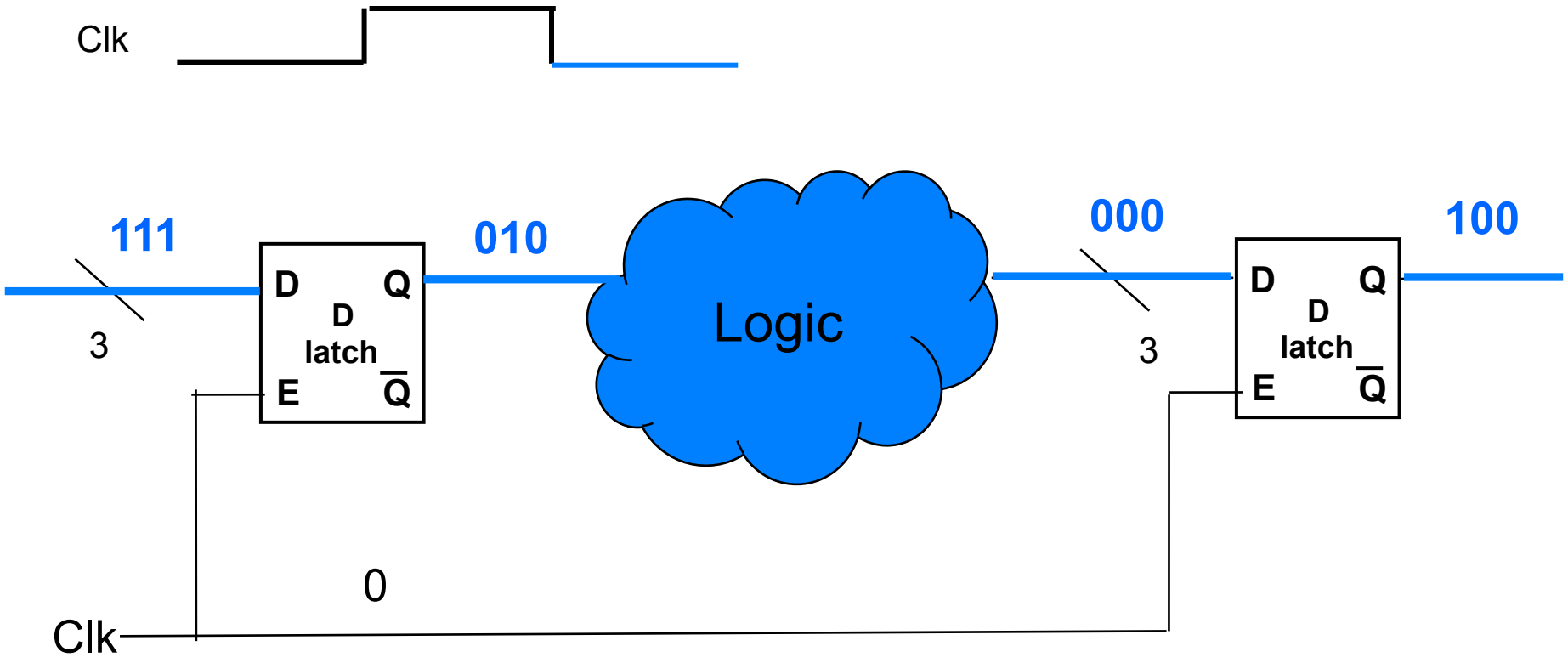
- **How we'd like this to work**
  - **Clock goes low before signals reach next latch**



© Alvin Lebeck, from Hilton, Sorin

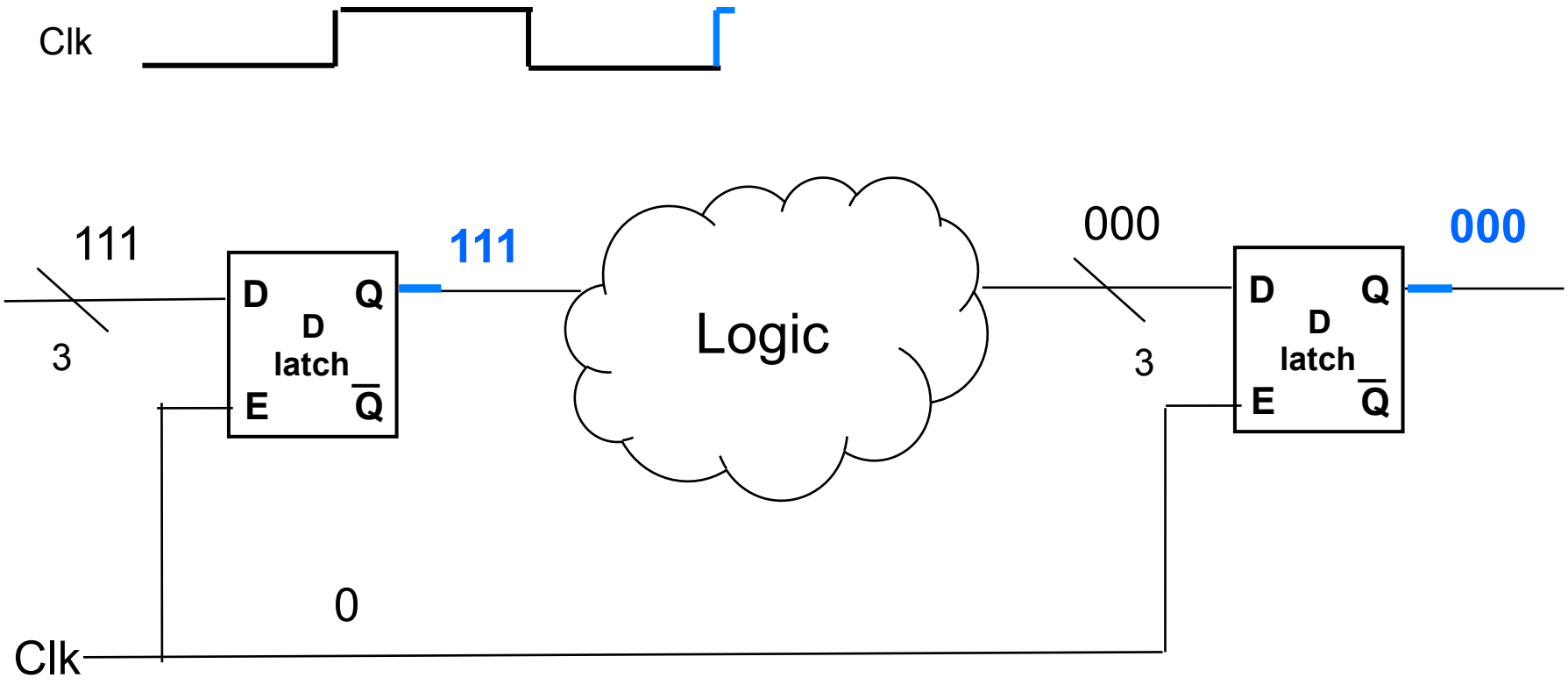
Level Triggered Clock

- **How we'd like this to work**
  - **Everything stable before clk goes high**



Level Triggered Clock

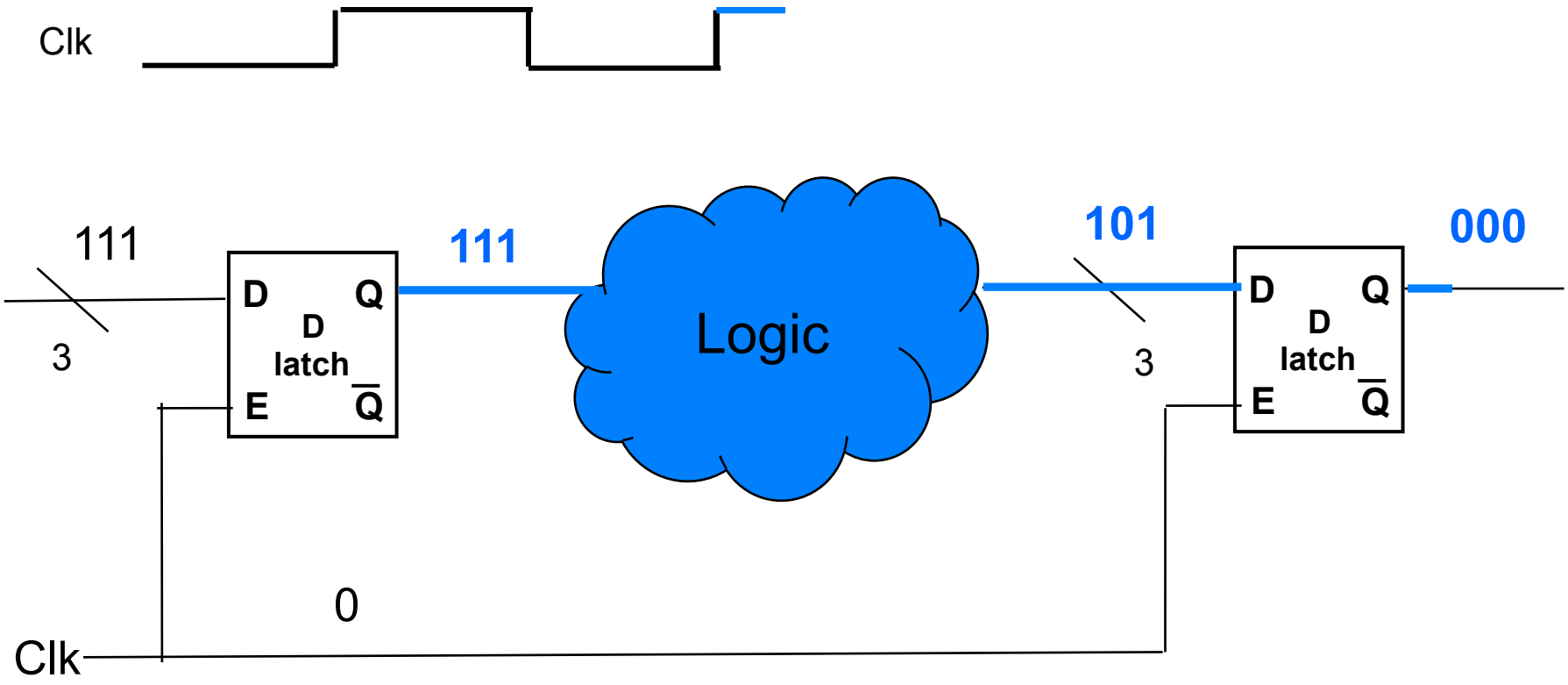
- **How we'd like this to work**
  - **Clk goes high again, repeat**



© Alvin Lebeck, from Hilton, Sorin

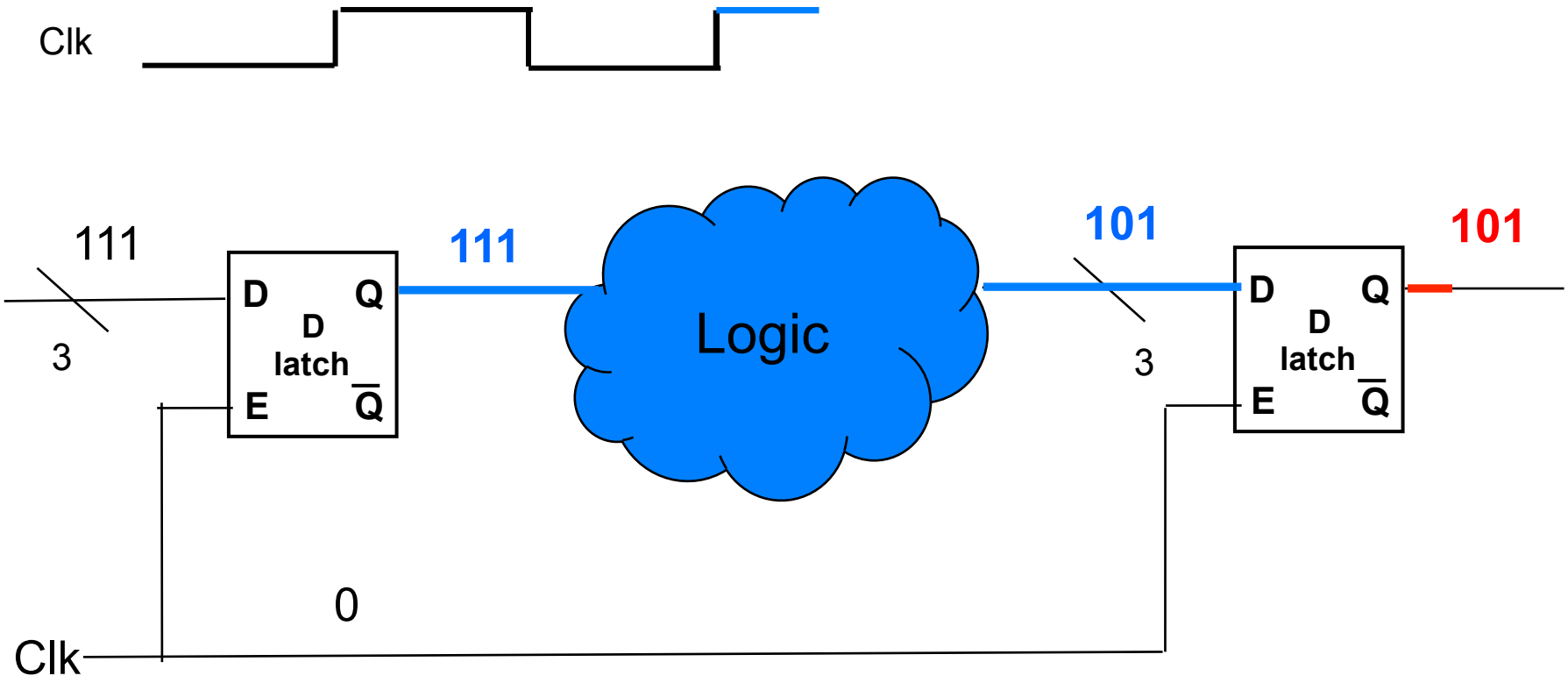
Level Triggered Clock

- **Problem: What if signal reaches latch too early?**
  - **i.e., while clk is still high**



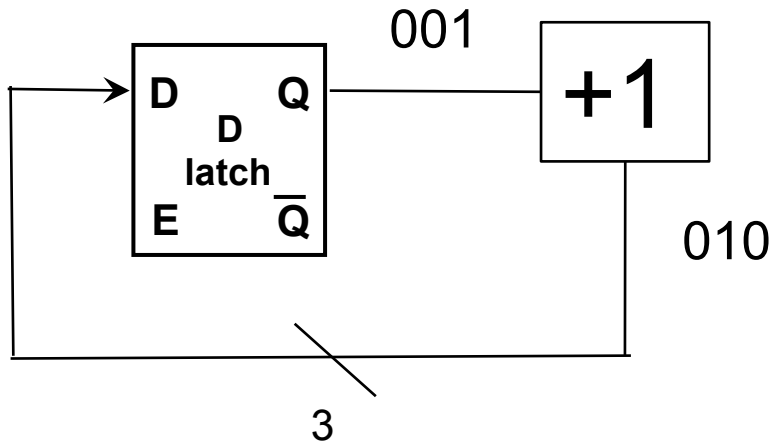
Level Triggered Clock

- **Problem: What if signal reaches latch too early?**
  - **Signal goes right through latch, into next stage..**



That would be bad…

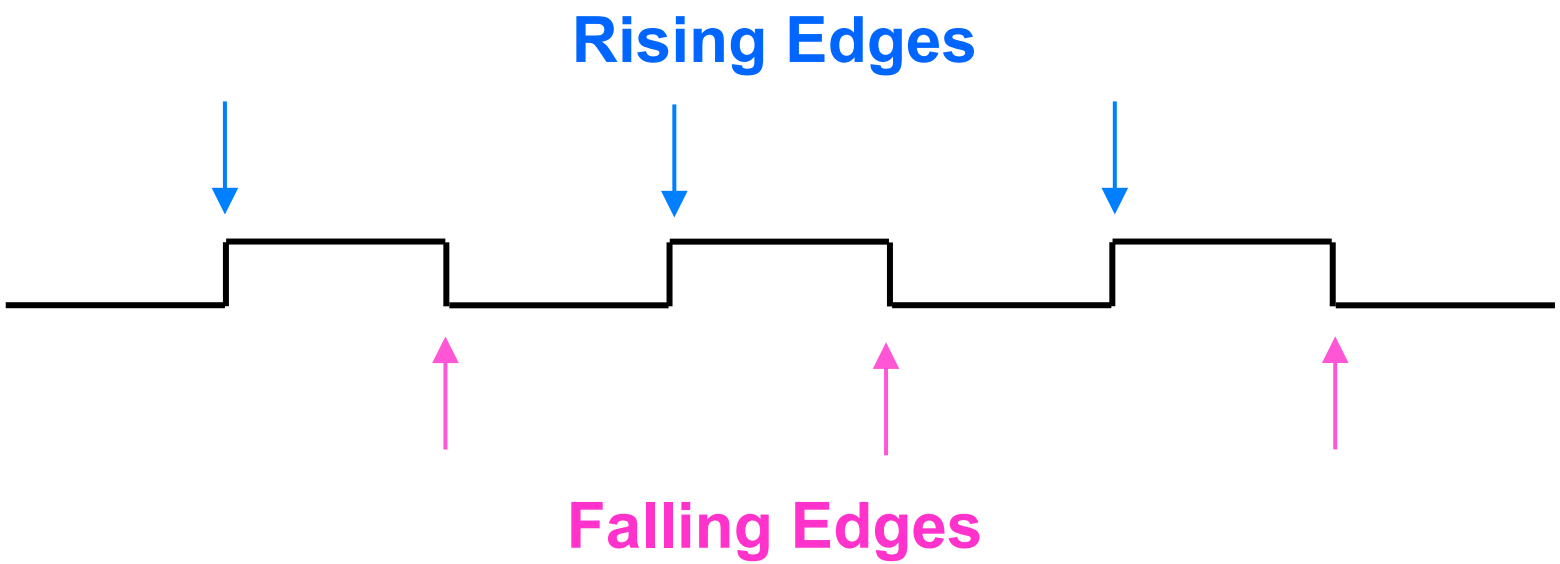
- **Getting into a stage too early is bad**
  - **Something else is going on there → corrupted**
  - **Also may be a loop with one latch**
- **Consider incrementing counter (or PC)**
  - **Too fast -- increment twice? Not good.**



© Alvin Lebeck, from Hilton, Sorin

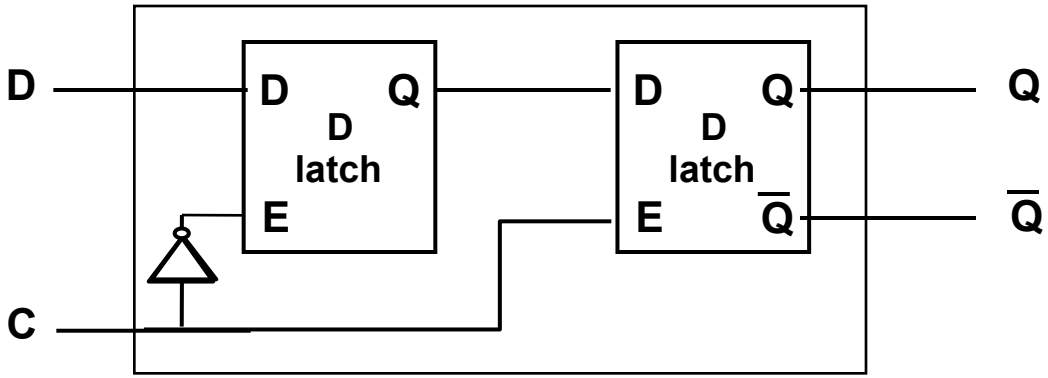
Edge Triggered

- **Instead of level triggered**
  - **Latch a new value at a clock level (high or low)**
- **We use edge triggered**
  - **Latch a value at an clock edge (rising or falling)**



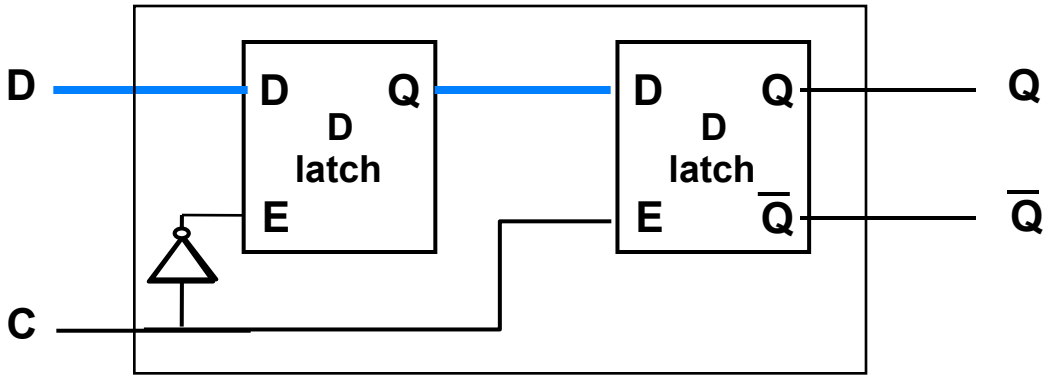
© Alvin Lebeck, from Hilton, Sorin

D Flip-Flop



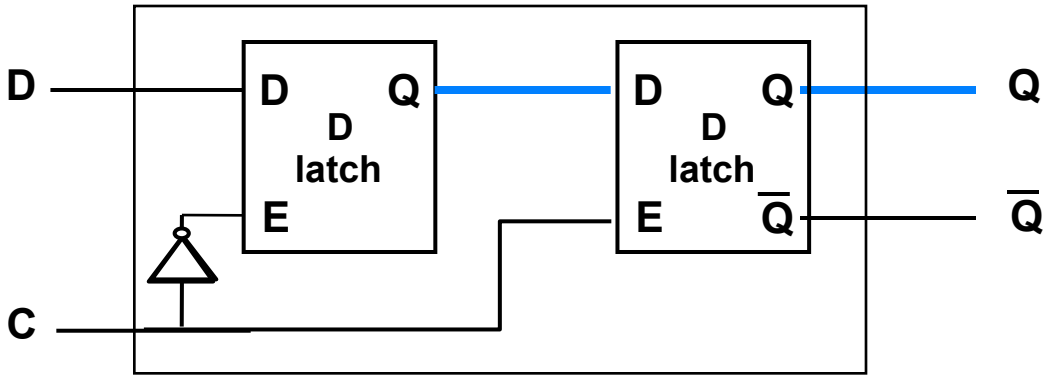
- **Rising edge triggered D Flip-flop**
  - **Two D Latches w/ Opposite clking of enables**

D Flip-Flop



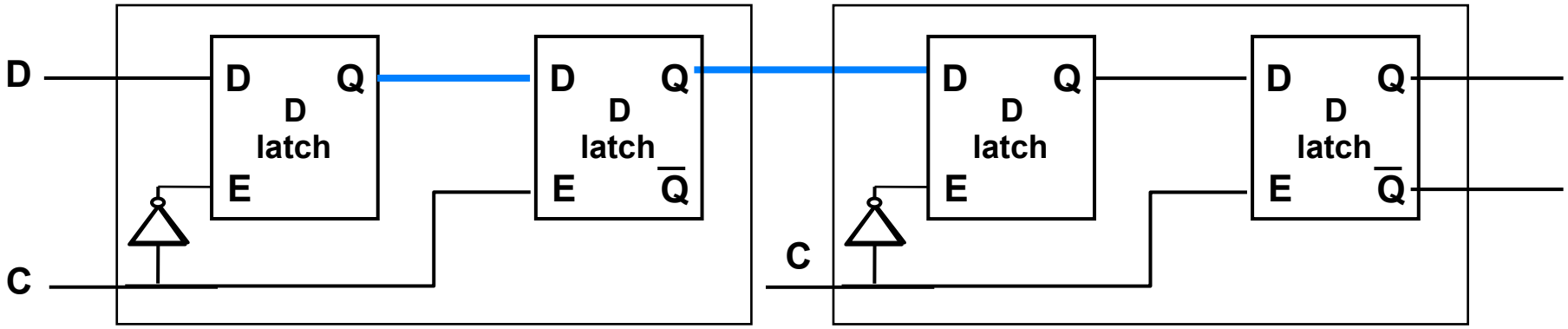
- **Rising edge triggered D Flip-flop**
  - Two D Latches w/ opposite clking of enables
  - On Low Clk, first latch enabled (propagates value)
    - Second not enabled, maintains value

D Flip-Flop



- **Rising edge triggered D Flip-flop**
  - Two D Latches w/ opposite clking of enables
  - On Low Clk, first latch enabled (propagates value)
    - Second not enabled, maintains value
  - On High Clk, second latch enabled
    - First latch not enabled, maintains value

D Flip-Flop



- **No possibility of "races"**
  - Even if I put 2 DFFs back-to-back…
  - By the time signal gets through 2nd latch of 1st DFF, 1st latch of 2nd DFF is disabled

- **Still must ensure signals reach DFF before clk rises**
  - Important concern in logic design is "making timing"

© Alvin Lebeck, from Hilton, Sorin

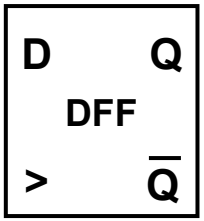
D Flip-flops (continued…)

- **Could also do falling edge triggered**
  - Switch which latch has NOT on clk

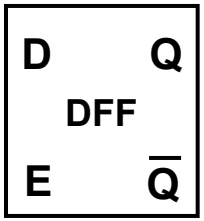
- **D Flip-flop is ubiquitous**
  - Typically people just say “latch” and mean DFF
  - Which edge is used does not matter
    - As long as same edge is used consistently
    - We will use rising edge

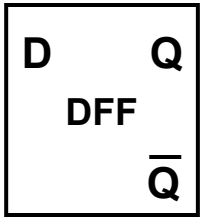
D flip flops

- **Generally do not draw clk input**
  - Have one global clk, assume it goes there
  - Often see > as symbol meaning clk



- **Maybe have explicit enable**
  - Might not want to write every cycle
  - If no enable signal shown, implies always enabled





- **Get output and NOT(output) for "free"**

© Alvin Lebeck, from Hilton, Sorin

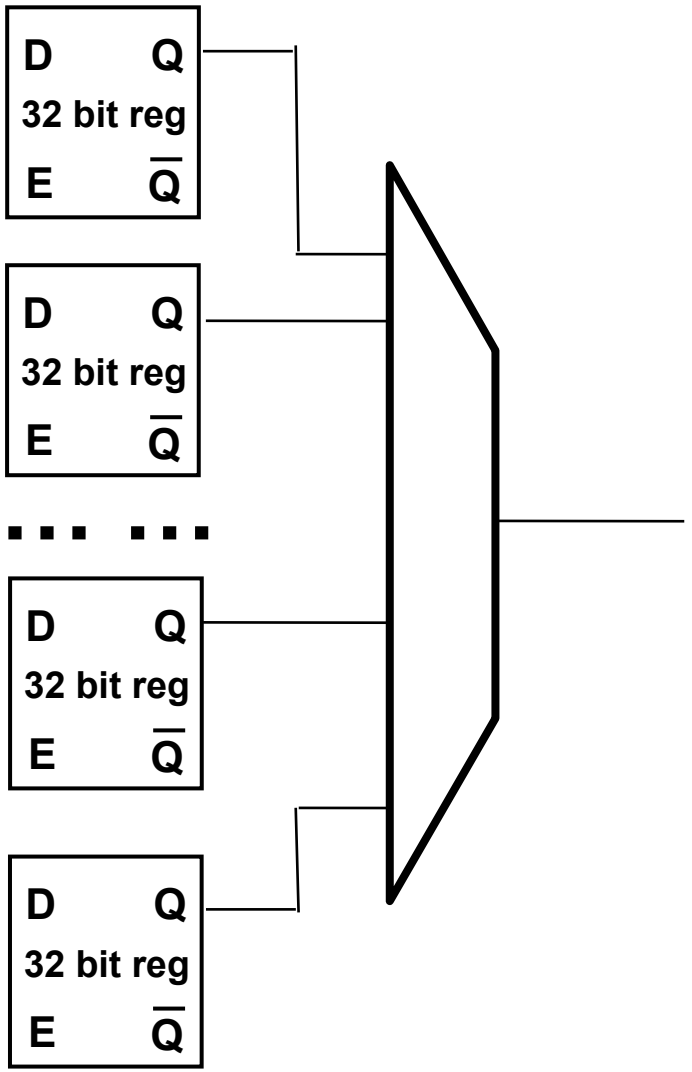
Register File

- Can store one value…what about manyvalues ?
- E.g., Register File (the physical storage for the regs)
  - MIPS, 32 32-bit integer registers

- How do we build a Register File using D Flip-Flops?
- What other components do we need?

Register File

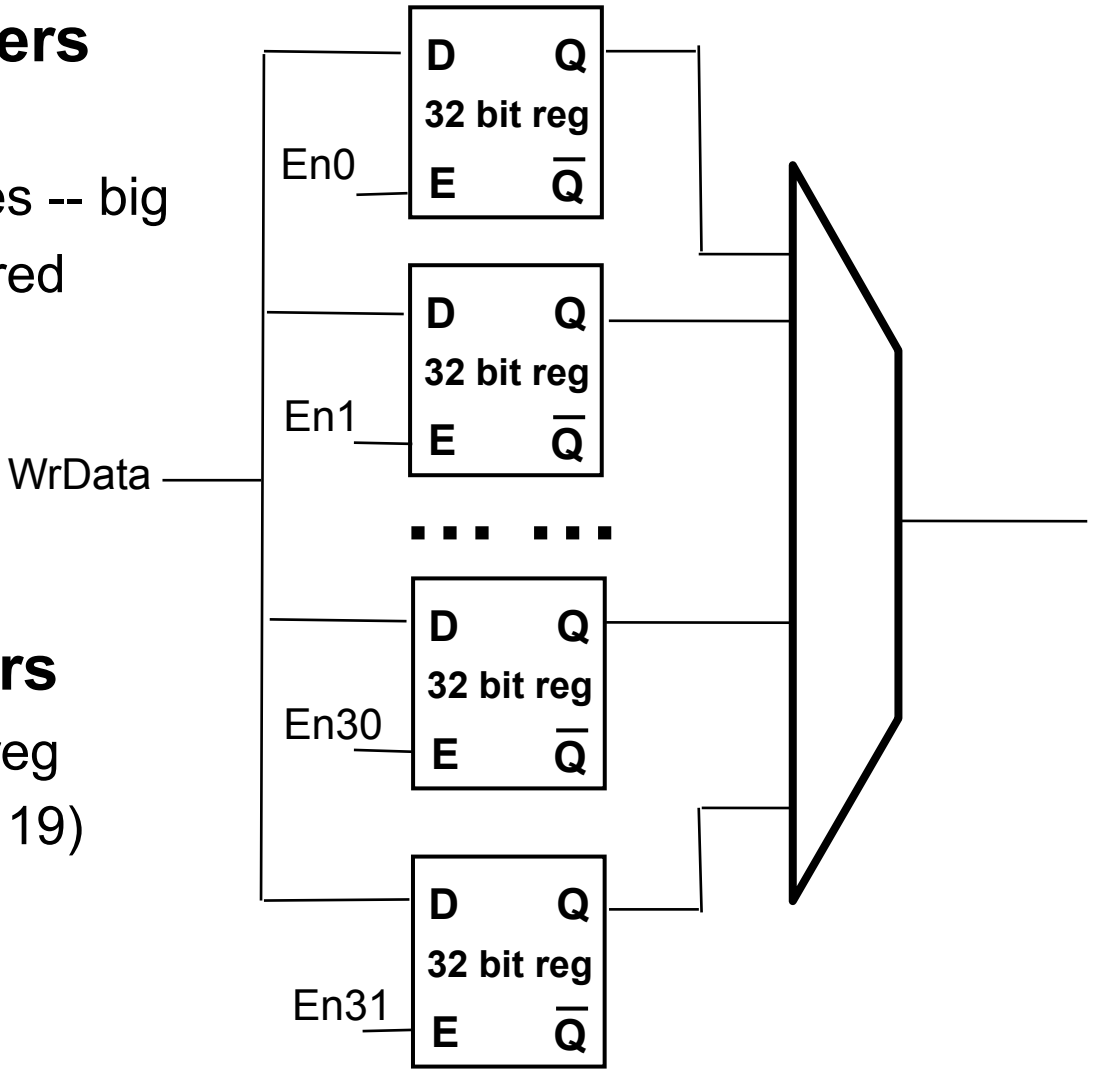
- **Reading the registers**
  - 32 input mux -- slow
  - Need 32 32-1 MUXes -- big
  - Other regs not pictured



© Alvin Lebeck, from Hilton, Sorin

Register File

- **Reading the registers**
  - 32 input mux -- slow
  - Need 32 32-1 MUXes -- big
  - Other regs not pictured

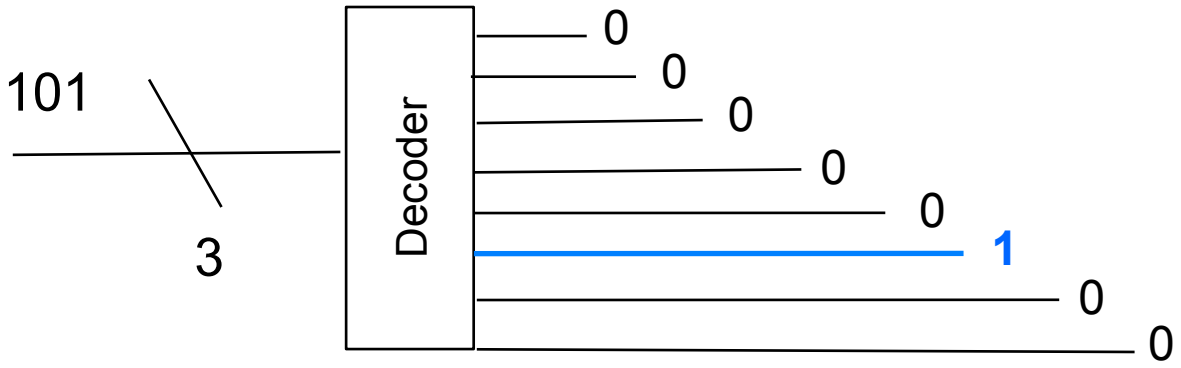


- **Writing the registers**
  - Need to pick which reg
  - Have reg num (e.g., 19)
  - Make En19=1
    - En0, En1,… = 0

© Alvin Lebeck, from Hilton, Sorin

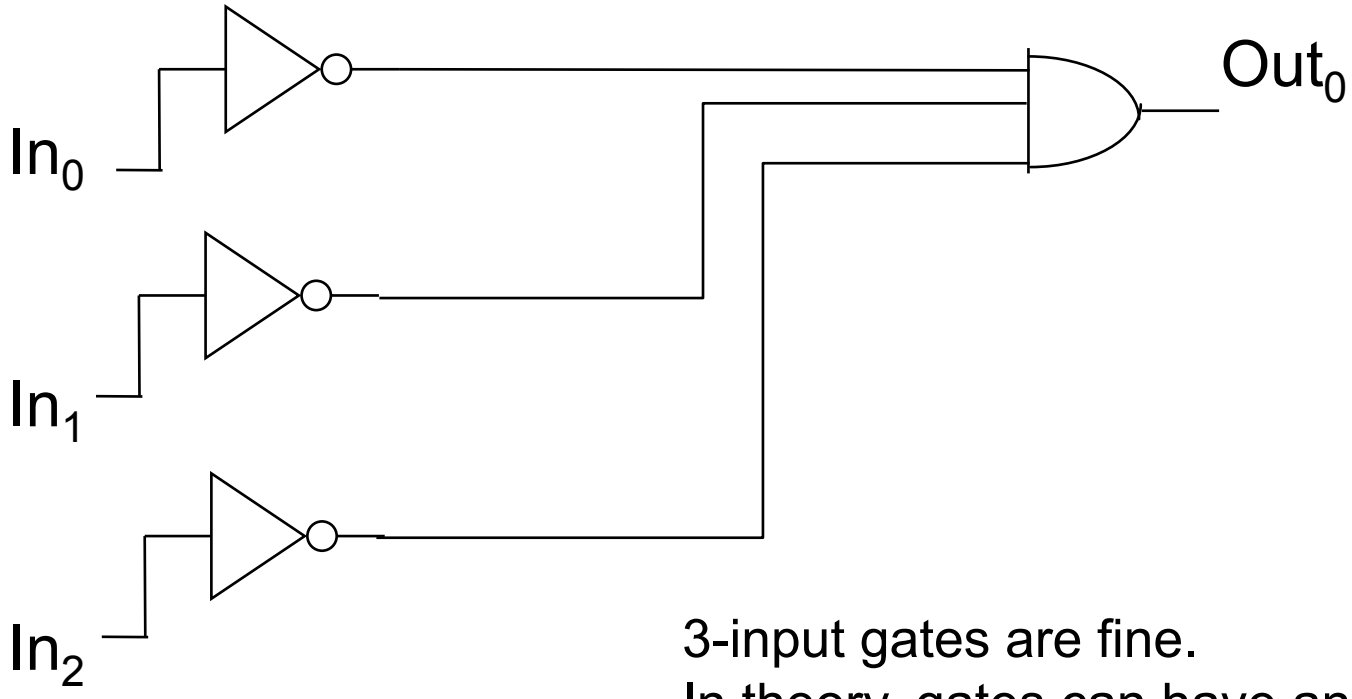
Decoders

- **First task: convert binary number to "one hot"**
  - N bits in
  - 2^N bits out
  - 2^N-1 bits are 0, 1 bit (matching the input) is 1



Decoder Logic

- **Decoder basically AND gates for each output:**
  - Out_0 only True (one) if input 000

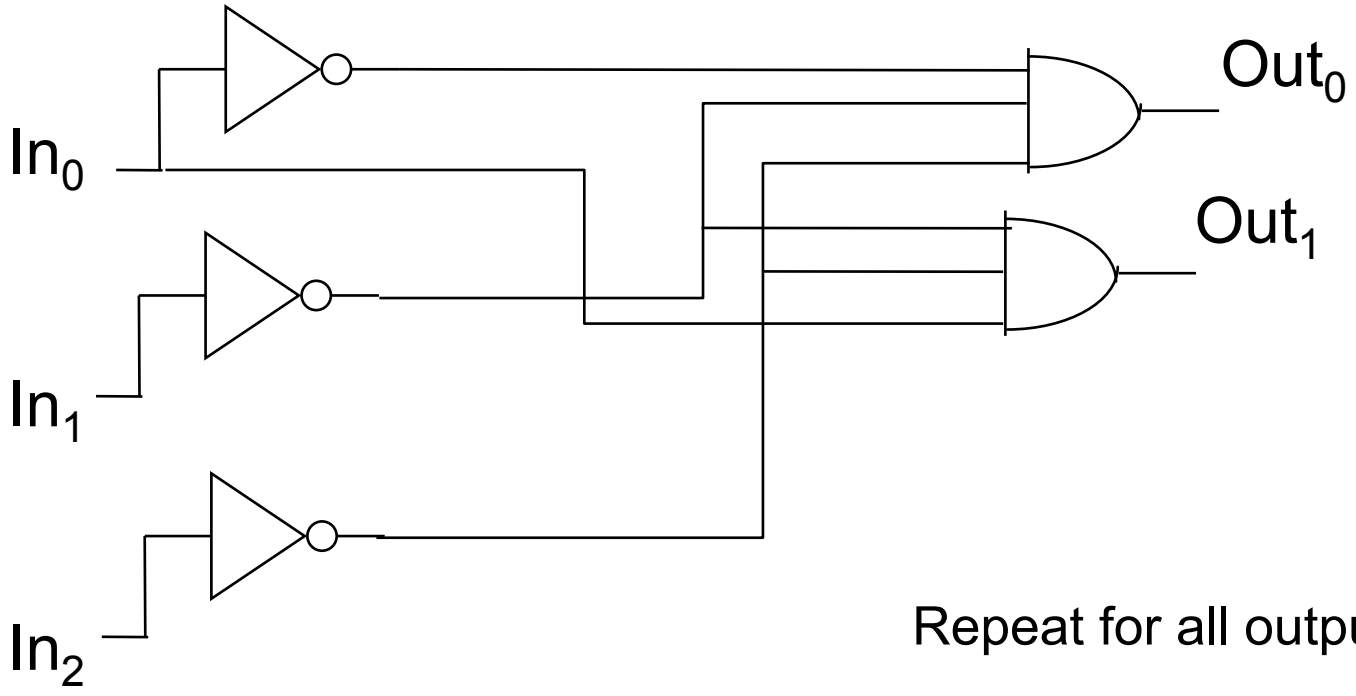


3-input gates are fine.
In theory, gates can have any # of inputs
In practice >4 converted to multiple gates

© Alvin Lebeck, from Hilton, Sorin

Decoder Logic

- **Decoder basically AND gates for each output:**
  - Out_1 only True (one) if input 001



Repeat for all outputs

AND together correct sets of bits

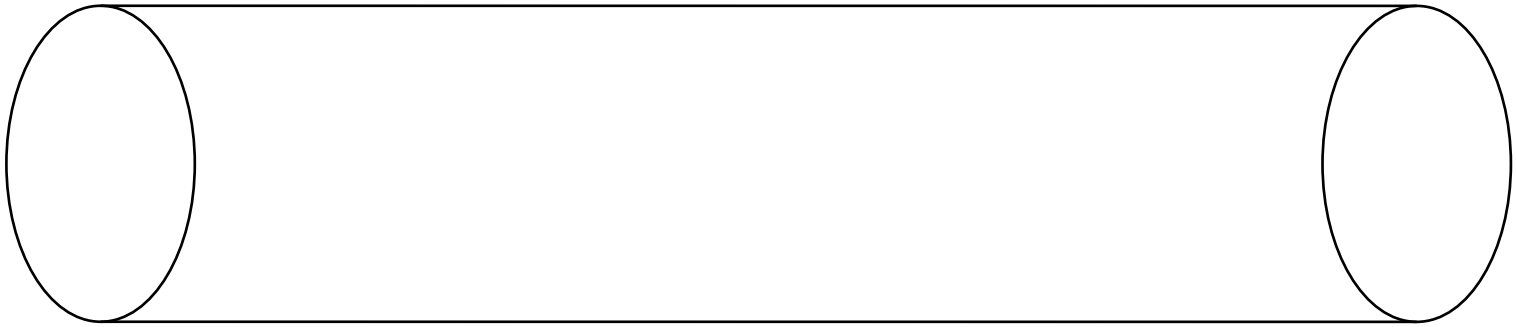
Register File

- **Decoder supports register addressing:**
  - Use decoder to convert register number into control signal
  - Send write data to all registers
  - Use one hot encoding to enable destination register

- **Need to fix register read speed**
  - 32 input mux is not realistic
  - For tractability, expand our world from {1,0} to {1, 0, Z}

Water Analogy

- **To understand Z, let's make an analogy**
  - Think of a wire as a pipe
    - Has water = 1
    - Has water = 0
  - This wire is 0 (it has no water)



Water Analogy

- **To understand Z, let's make an analogy**
  - Think of a wire as a pipe
    - Has water = 1
    - Has water = 0
  - This wire is 1 (its full of water)

Water Analogy

- **To understand Z, let's make an analogy**
  - Think of a wire as a pipe
    - Has water = 1
    - Has water = 0
  - Suppose a gate drives a 0 onto this wire
    - Drain the water

© Alvin Lebeck, from Hilton, Sorin

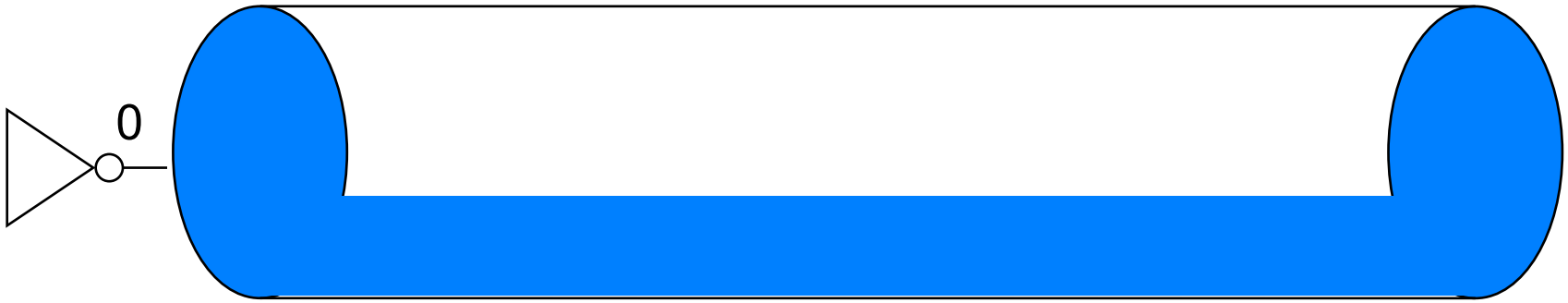
Water Analogy

- **To understand Z, let's make an analogy**
  - Think of a wire as a pipe
    - Has water = 1
    - Has water = 0
  - Suppose a gate drives a 0 onto this wire
    - Drain the water

© Alvin Lebeck, from Hilton, Sorin

Water Analogy

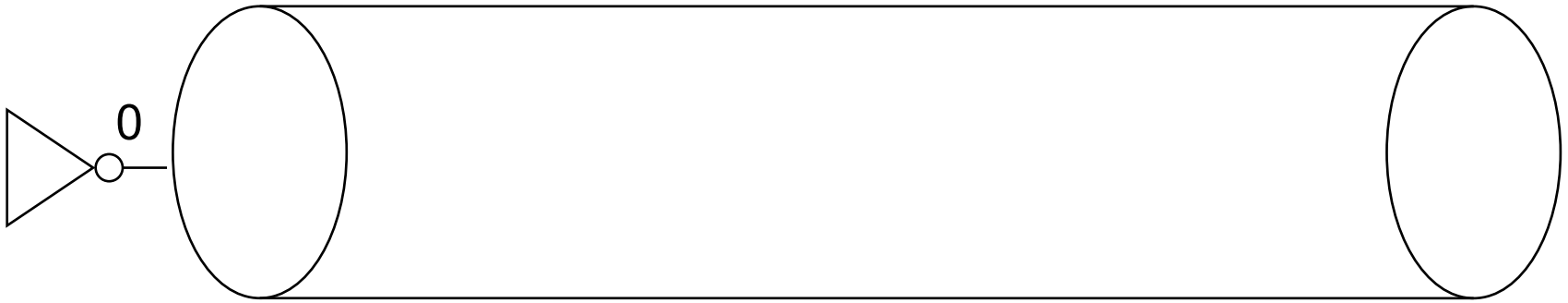
- **To understand Z, let's make an analogy**
  - Think of a wire as a pipe
    - Has water = 1
    - Has water = 0
  - Suppose a gate drives a 0 onto this wire
    - Drain the water



© Alvin Lebeck, from Hilton, Sorin

Water Analogy

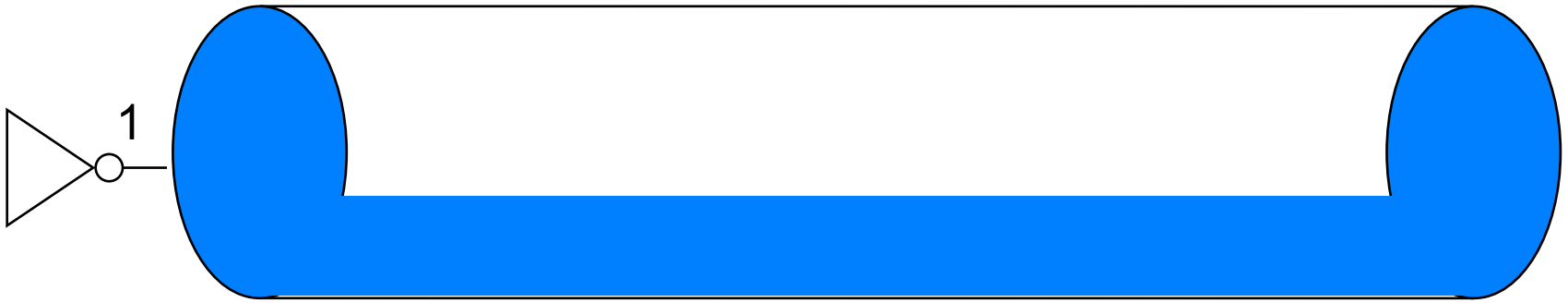
- **To understand Z, let's make an analogy**
  - Think of a wire as a pipe
    - Has water = 1
    - Has water = 0
  - Suppose a gate drives a 0 onto this wire
    - Drain the water



© Alvin Lebeck, from Hilton, Sorin

Water Analogy

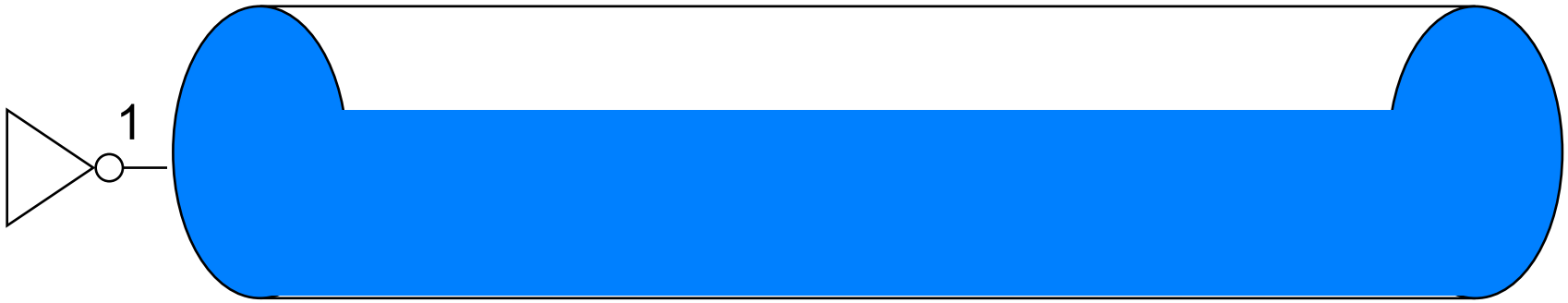
- **To understand Z, let's make an analogy**
  - Think of a wire as a pipe
    - Has water = 1
    - Has water = 0
  - Suppose the gate now drives a 1
    - Pump the water



© Alvin Lebeck, from Hilton, Sorin

Water Analogy

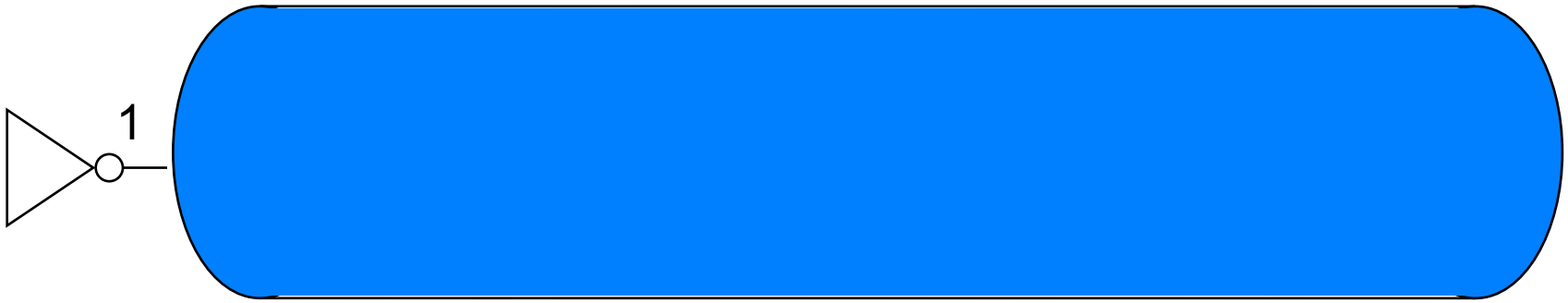
- **To understand Z, let's make an analogy**
  - Think of a wire as a pipe
    - Has water = 1
    - Has water = 0
  - Suppose the gate now drives a 1
    - Pump the water



© Alvin Lebeck, from Hilton, Sorin

Water Analogy

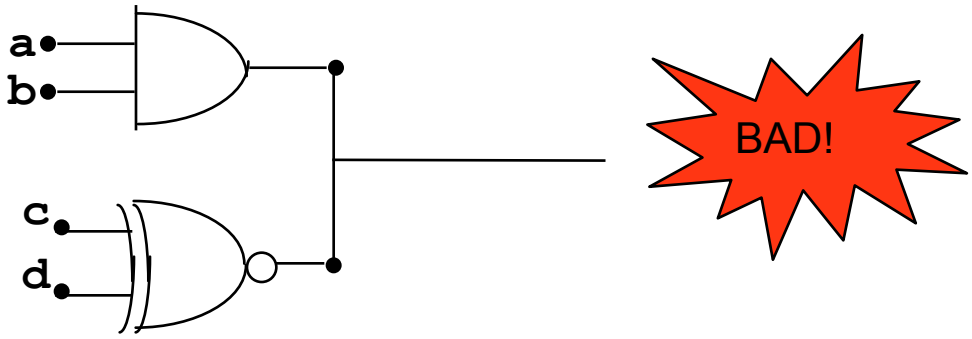
- **To understand Z, let's make an analogy**
  - Think of a wire as a pipe
    - Has water = 1
    - Has water = 0
  - Suppose the gate now drives a 1
    - Pump the water



© Alvin Lebeck, from Hilton, Sorin

Remember this rule?

- **Do not connect two outputs to the same wire**

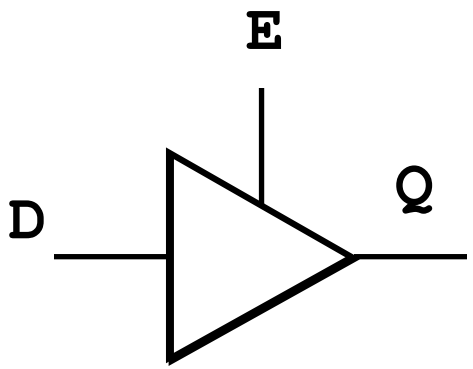


- **One gate drives 1. The other drives 0.**
  - One pumps water in. The other drains water out
  - Except it's not water, it's electric charge
  - "Short circuit" → lots of current → lots of heat

A third option: Z

- There is a third possibility: Z ("high impedance")
  - Neither pumping or draining charge
  - Prevents charge from flowing through

- Gate that gives us Z : Tri-state

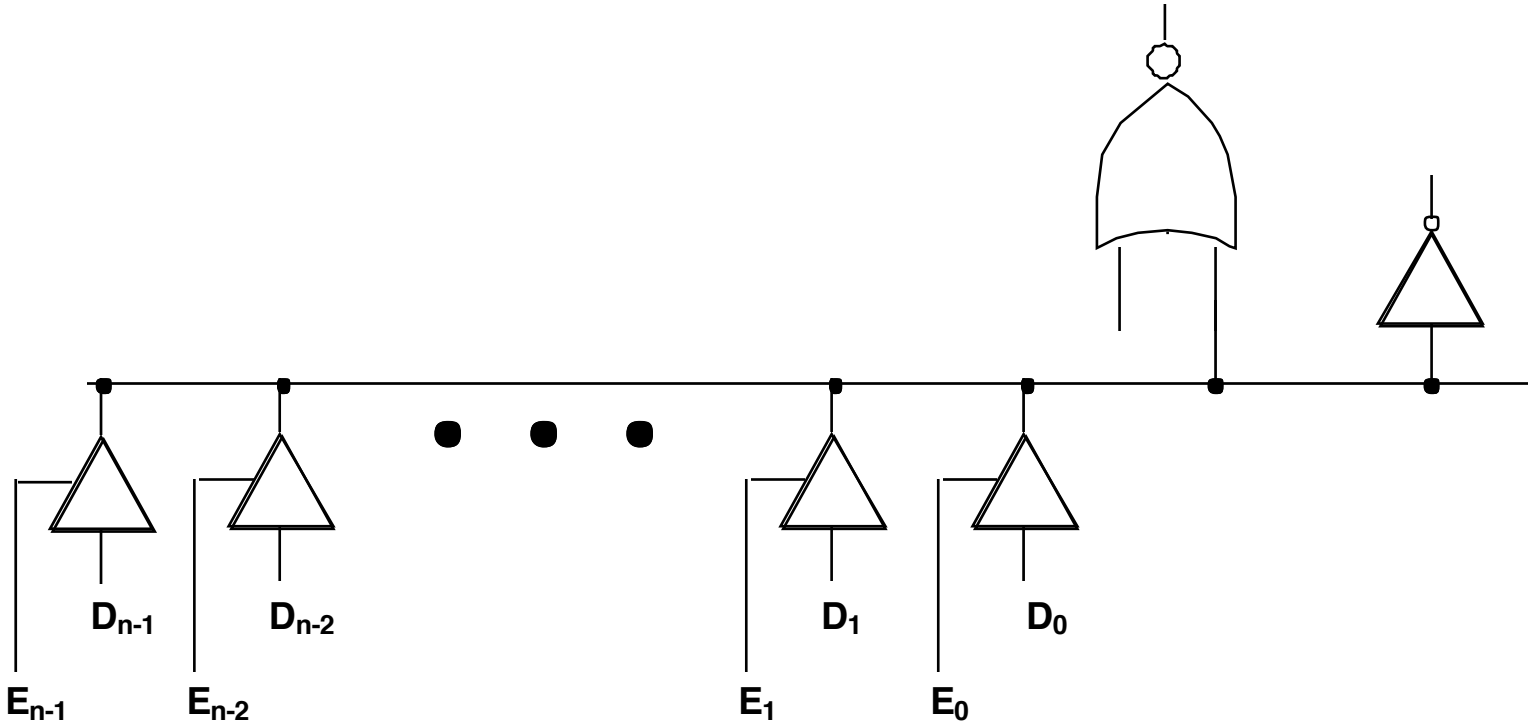


D	E	Q
0	1	0
1	1	1
-	0	Z

© Alvin Lebeck, from Hilton, Sorin

Tri-State Buffers

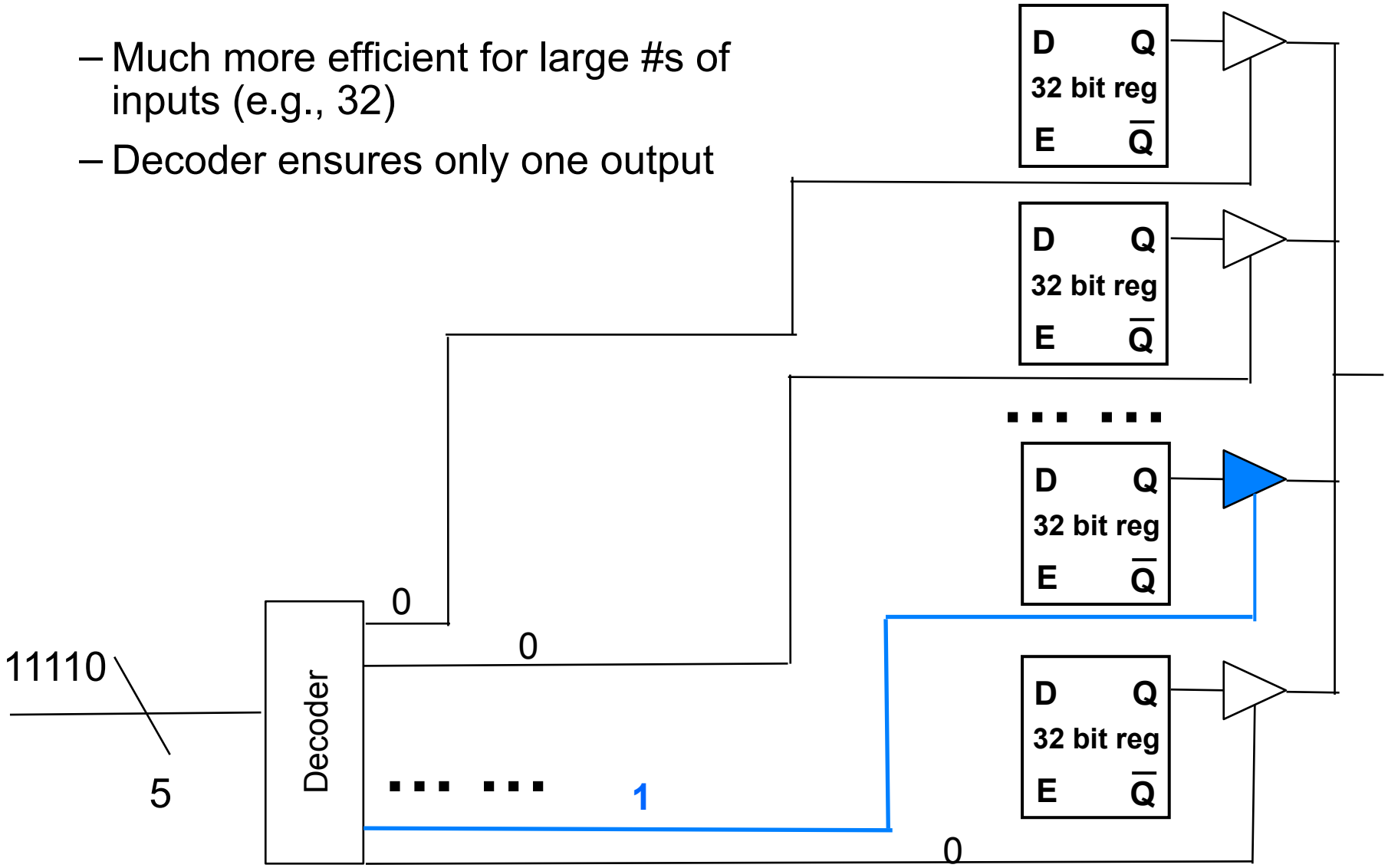
It's ok to connect multiple outputs together under one circumstance -- all but one must be outputting Z at any time



© Alvin Lebeck, from Hilton, Sorin

Mux with Tri-State Buffers

- Much more efficient for large #s of inputs (e.g., 32)
- Decoder ensures only one output



© Alvin Lebeck, from Hilton, Sorin

Ports

- **Read Ports**
  - Ability to do one read per clock cycle
  - May want more -- read two source registers per instruction
    - Maybe even more if we do many instrs at once (later…)
  - Could add more: need to replicate port
    - Another decoder
    - Another set of tri-states
    - Another output bus (wire connecting the tri-states)
- **Write Ports**
  - Ability to do one write/cycle
  - Could add more: need to multiplex write values

Minor Detail

- **This is not how a register file is implemented in today's processors**
  - (Though it is how other things are implemented)
  - Actually done with SRAM
  - We'll see that later this semester…