# Arithmetic

**Hakim Weatherspoon**
**CS 3410, Spring 2012**
Computer Science
Cornell University

See P&H 2.4 (signed), 2.5, 2.6, C.6, and Appendix C.6

## Goals for today

Binary (Arithmetic) Operations

- One-bit and four-bit adders
- Negative numbers and two’s compliment
- Addition (two’s compliment)
- Subtraction (two’s compliment)
- Performance

# Binary Addition

$$\begin{array}{r} 183 \\ +\ 254 \\ \hline \end{array}$$

Addition works the same way regardless of base

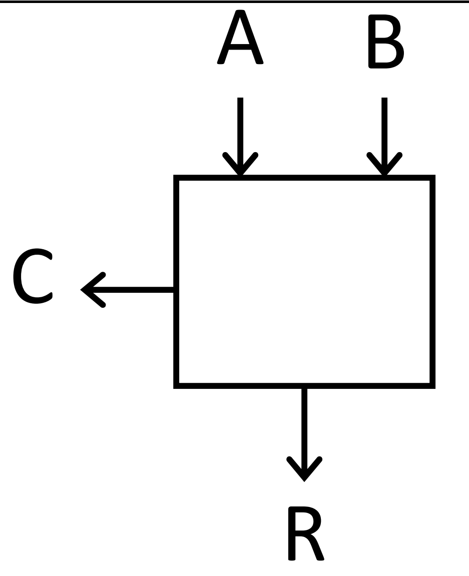
- Add the digits in each position
- Propagate the carry

$$\begin{array}{r} 001110 \\ +\ 011100 \\ \hline \end{array}$$

Unsigned binary addition is pretty easy

- Combine two bits at a time
- Along with a carry

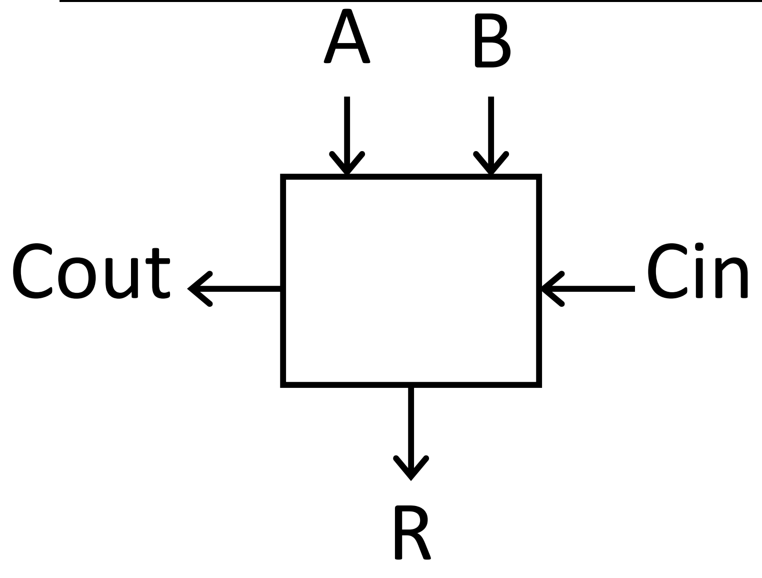
# 1-bit Adder



Half Adder

- Adds two 1-bit numbers
- Computes 1-bit result and 1-bit carry

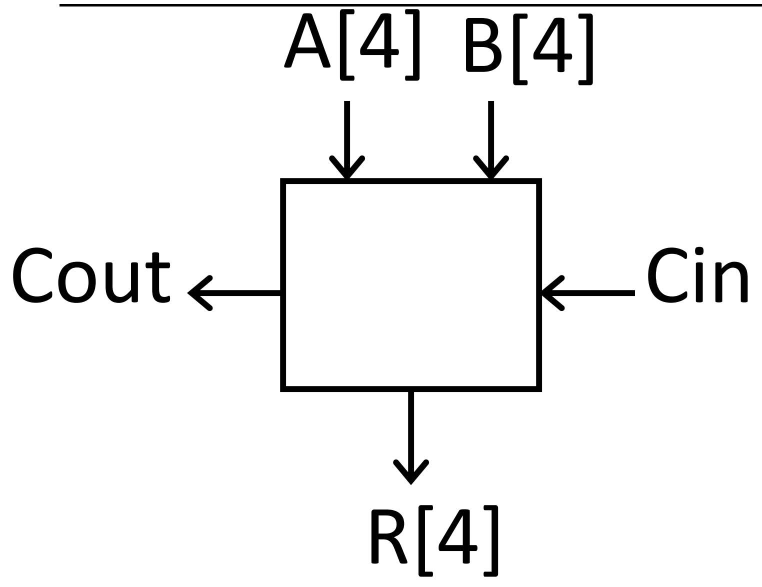
# 1-bit Adder with Carry



Full Adder

- Adds three 1-bit numbers
- Computes 1-bit result and 1-bit carry
- Can be cascaded

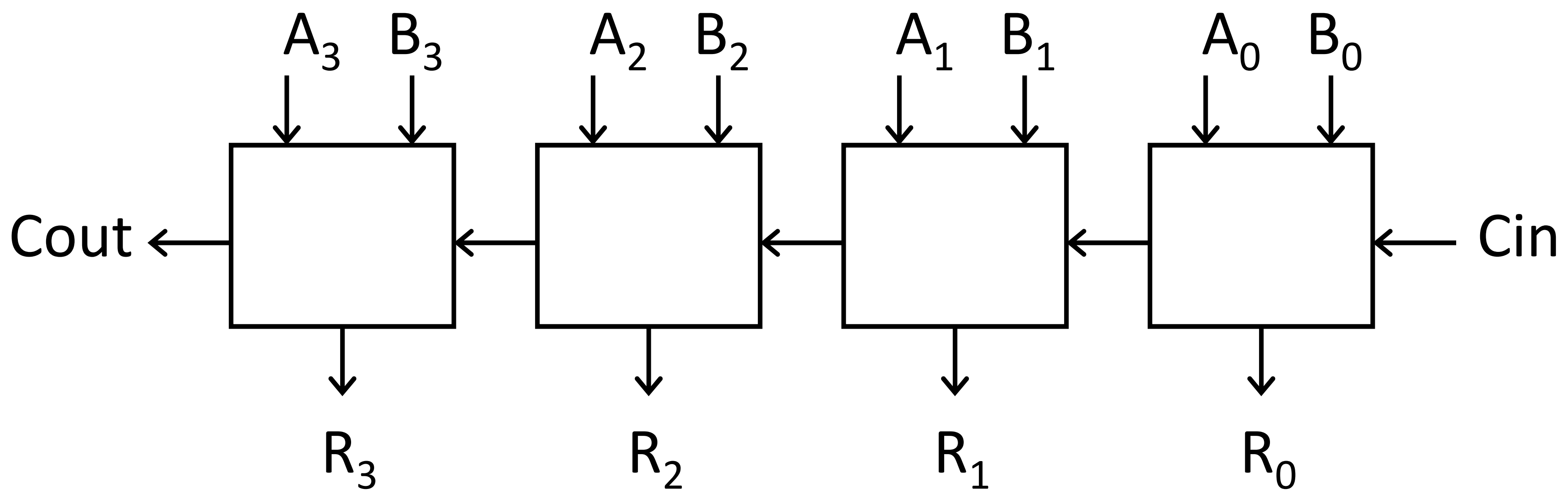
## 4-bit Adder



4-Bit Full Adder

- Adds two 4-bit numbers and carry in
- Computes 4-bit result and carry out
- Can be cascaded

# 4-bit Adder



- Adds two 4-bit numbers, along with carry-in
- Computes 4-bit result and carry out

- Carry-out = overflow indicates result does not fit in 4 bits

## Arithmetic with Negative Numbers

Negative Numbers Complicate Arithmetic

Recall addition with negatives:

## Arithmetic with Negative Numbers

Negative Numbers Complicate Arithmetic

Recall addition with negatives:

- pos + pos → add magnitudes, result positive
- neg + neg → add magnitudes, result negative
- pos + neg → subtract smaller magnitude,
  keep sign of bigger magnitude

## First Attempt: Sign/Magnitude Representation

First Attempt: Sign/Magnitude Representation

- 1 bit for sign (0=positive, 1=negative)
- N-1 bits for magnitude

## Two’s Complement Representation

Better: Two’s Complement Representation

- Leading 1’s for negative numbers
- To negate any number:
  - complement *all* the bits
  - then add 1

# Two’s Complement

Non-negatives (as usual):

+0 = 0000

+1 = 0001

+2 = 0010

+3 = 0011

+4 = 0100

+5 = 0101

+6 = 0110

+7 = 0111

+8 = 1000

Negatives (two’s complement: flip then add 1):

## Two’s Complement

| Non-negatives (as usual): | Negatives (two’s complement: flip then add 1): | |
|---|---|---|
| +0 = 0000 | ~0 = 1111 | -0 = 0000 |
| +1 = 0001 | ~1 = 1110 | -1 = 1111 |
| +2 = 0010 | ~2 = 1101 | -2 = 1110 |
| +3 = 0011 | ~3 = 1100 | -3 = 1101 |
| +4 = 0100 | ~4 = 1011 | -4 = 1100 |
| +5 = 0101 | ~5 = 1010 | -5 = 1011 |
| +6 = 0110 | ~3 = 1001 | -6 = 1010 |
| +7 = 0111 | ~7 = 1000 | -7 = 1001 |
| +8 = 1000 | ~8 = 0111 | -8 = 1000 |

## Two’s Complement Facts

Signed two’s complement

- Negative numbers have leading 1’s
- zero is unique: +0 = - 0
- wraps from largest positive to largest negative

N bits can be used to represent

- unsigned:
  - eg: 8 bits $\Rightarrow$
- signed (two’s complement):
  - ex: 8 bits $\Rightarrow$

## Sign Extension & Truncation

Extending to larger size

Truncate to smaller size

## Two’s Complement Addition

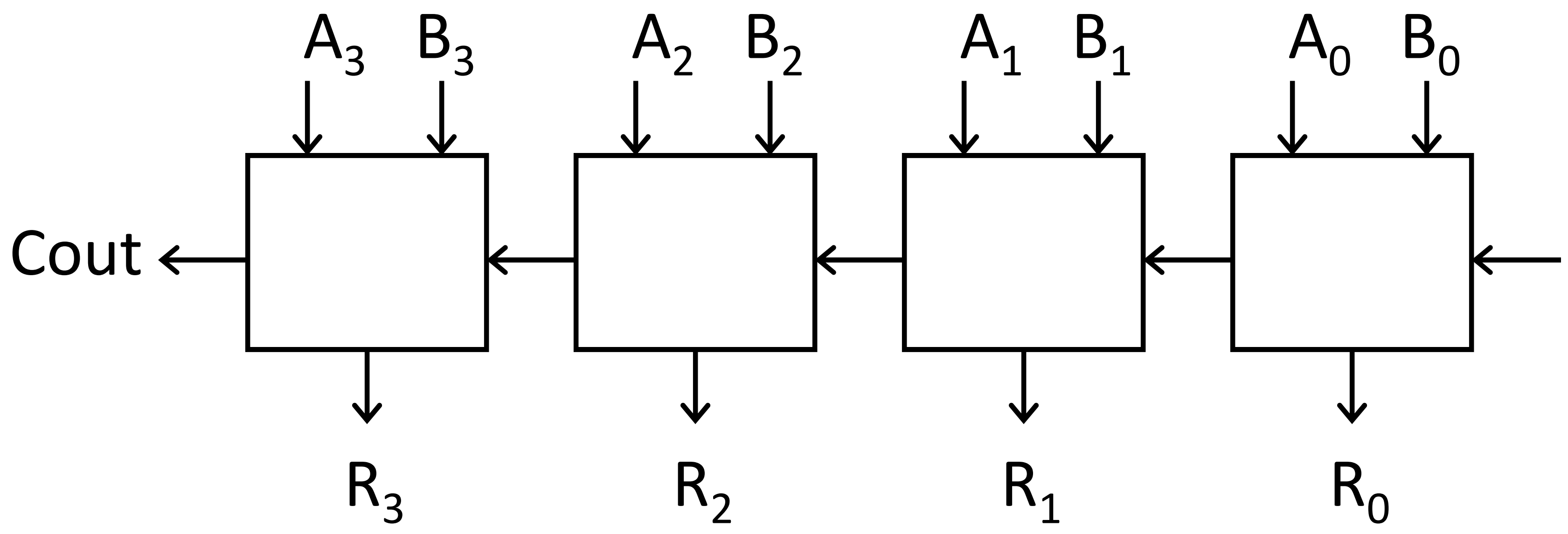
Addition with two’s complement signed numbers

- Perform addition as usual, regardless of sign (it just works)

# Two's Complement Addition

Addition with two's complement signed numbers

- Perform addition as usual, regardless of sign (it just works)



# Overflow

Overflow

- adding a negative and a positive?
- adding two positives?
- adding two negatives?

## Overflow

Overflow

- adding a negative and a positive?

- adding two positives?

- adding two negatives?

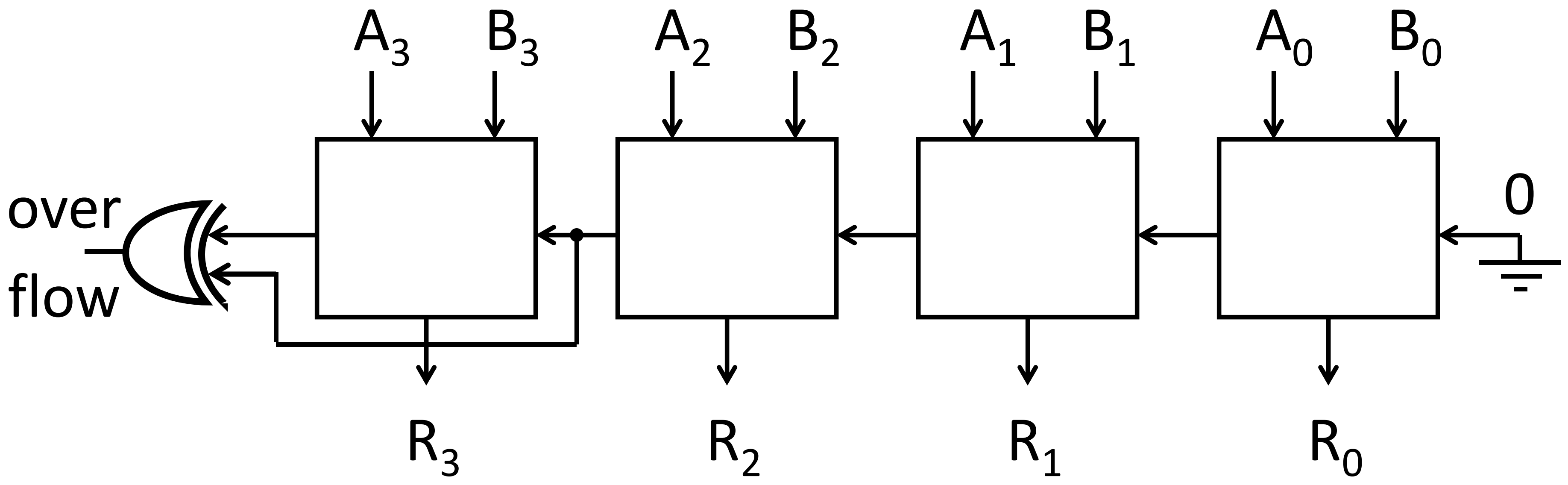
Rule of thumb:

Overflow happened iff

carry into msb != carry out of msb

# Two’s Complement Adder

Two’s Complement Adder with overflow detection



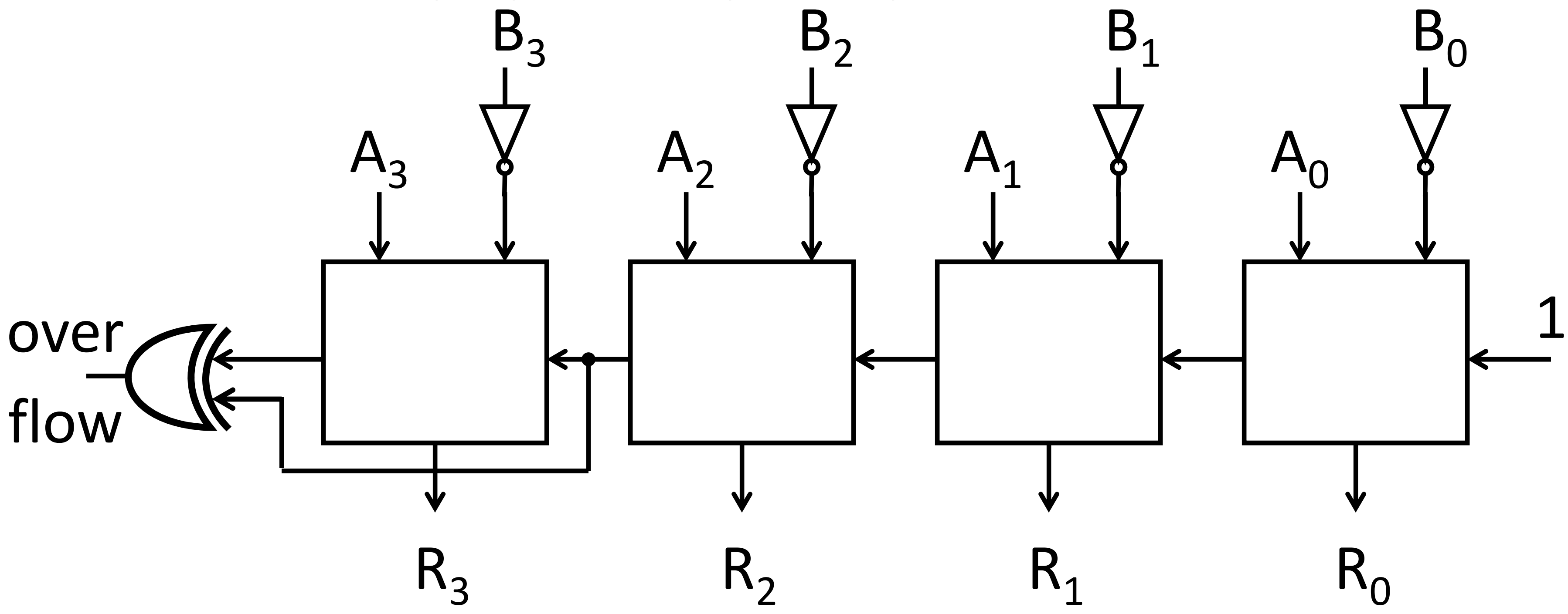
## Binary Subtraction

Two’s Complement Subtraction

## Binary Subtraction

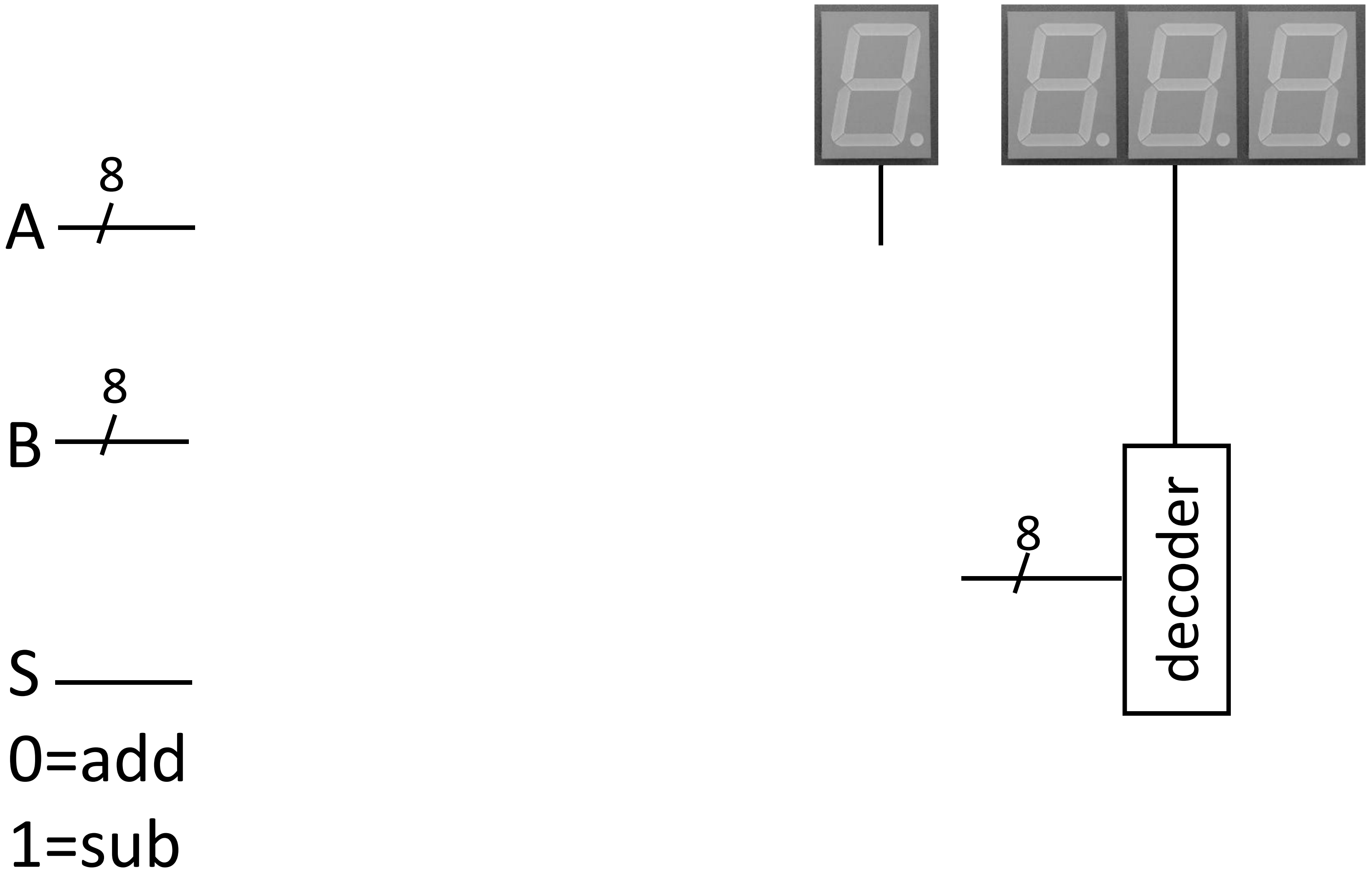
### Two's Complement Subtraction

$A - B = A + (-B) = A + (\overline{B} + 1)$

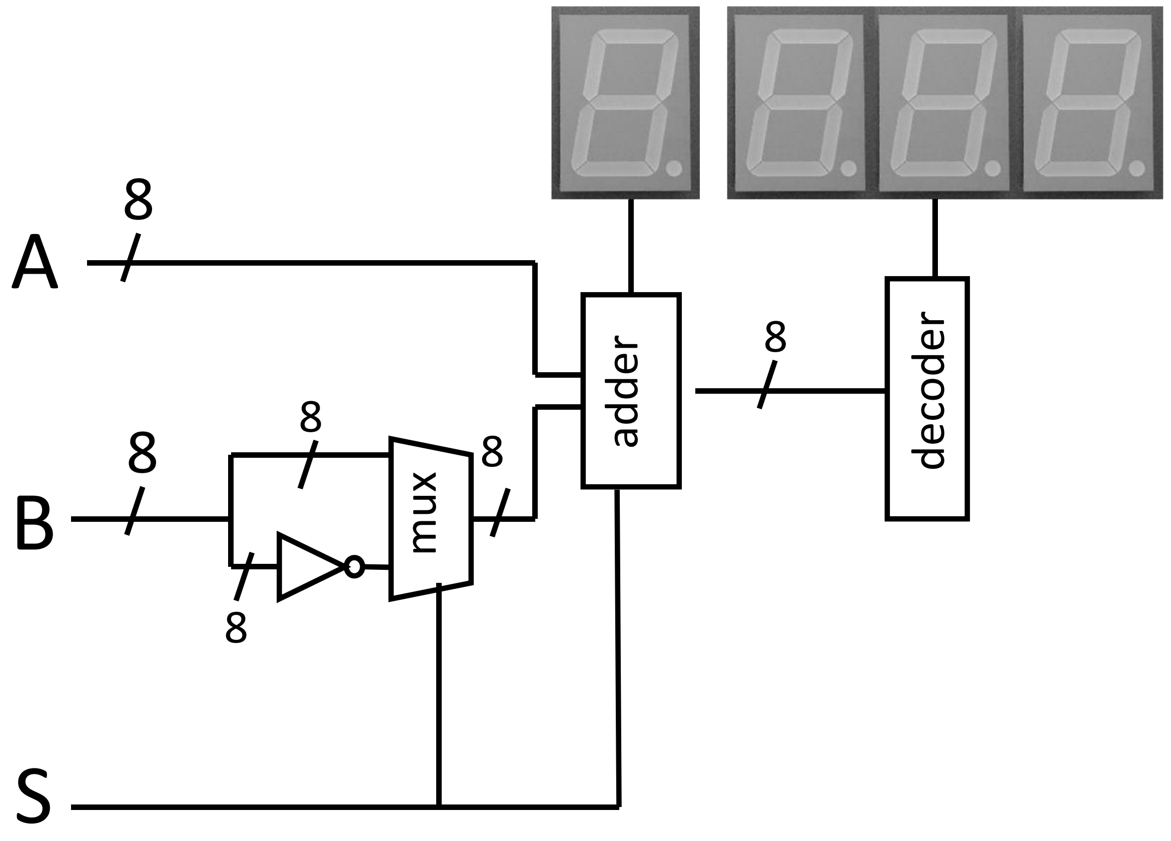


Q: What if (-B) overflows?

# A Calculator



# A Calculator

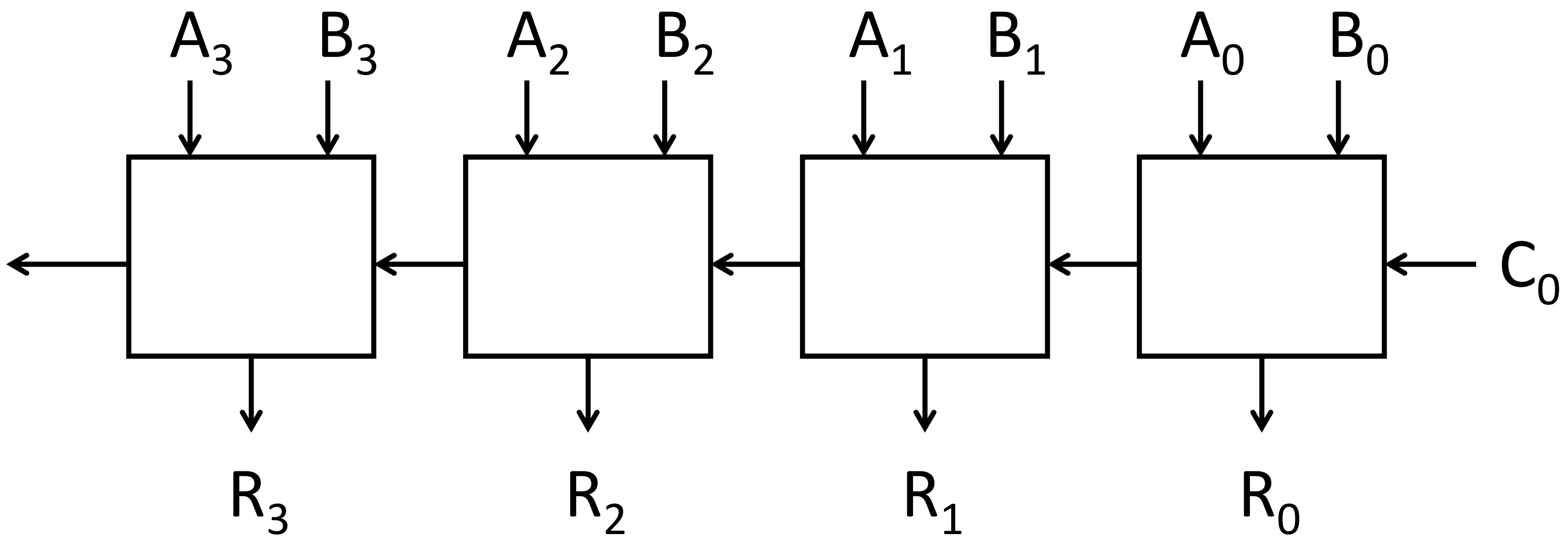


0=add

1=sub

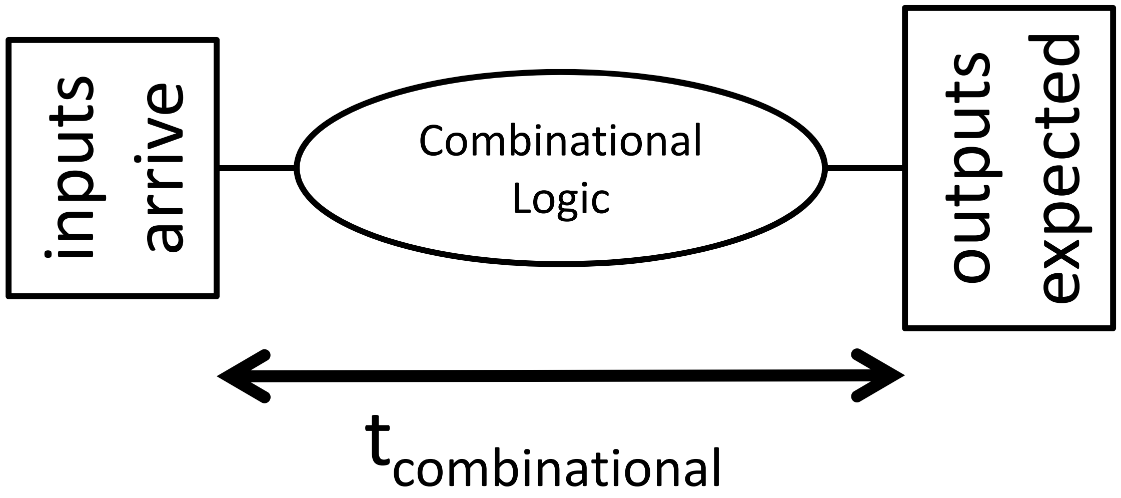
# Efficiency and Generality

- Is this design fast enough?
- Can we generalize to 32 bits? 64? more?

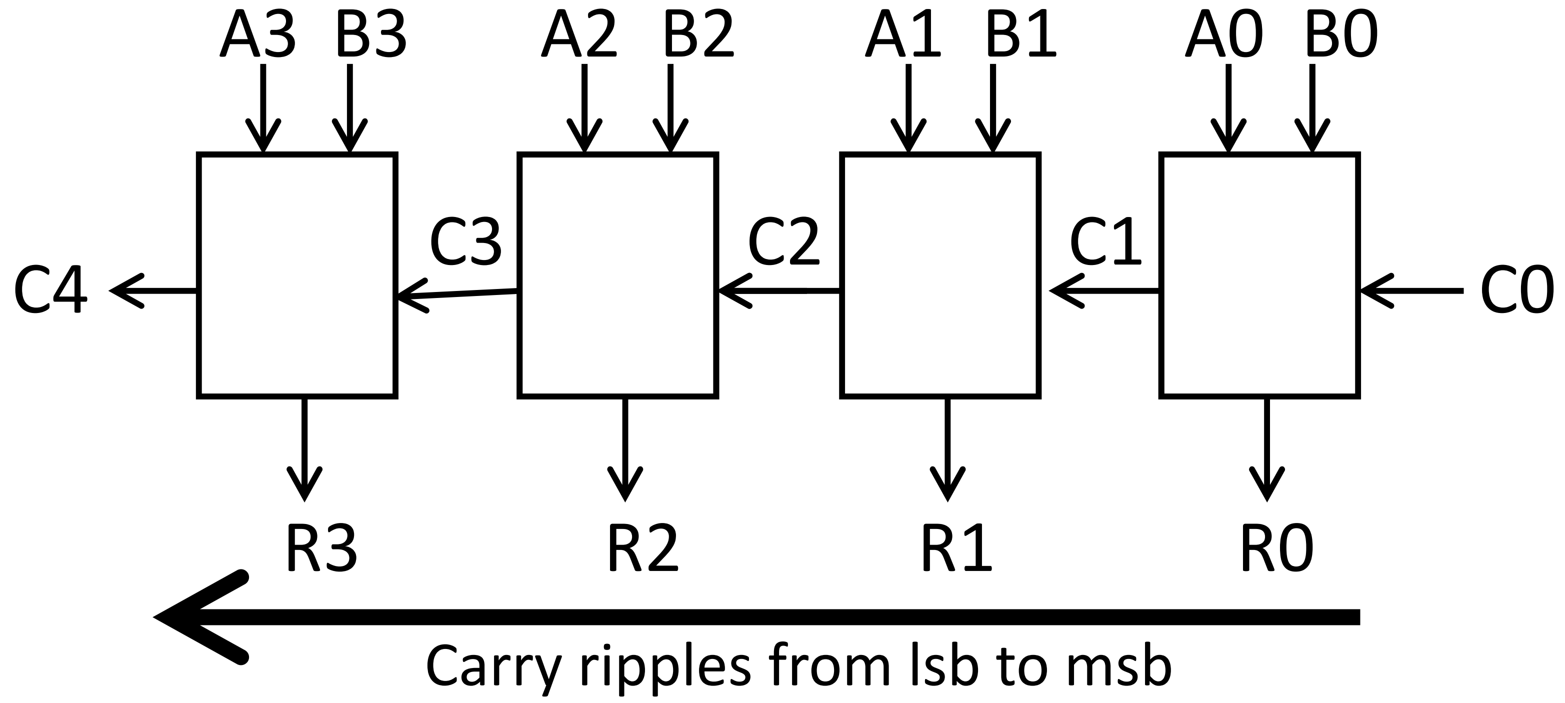


## Performance

Speed of a circuit is affected by the number of gates in series (on the *critical path* or the *deepest level of logic*)



# 4-bit Ripple Carry Adder

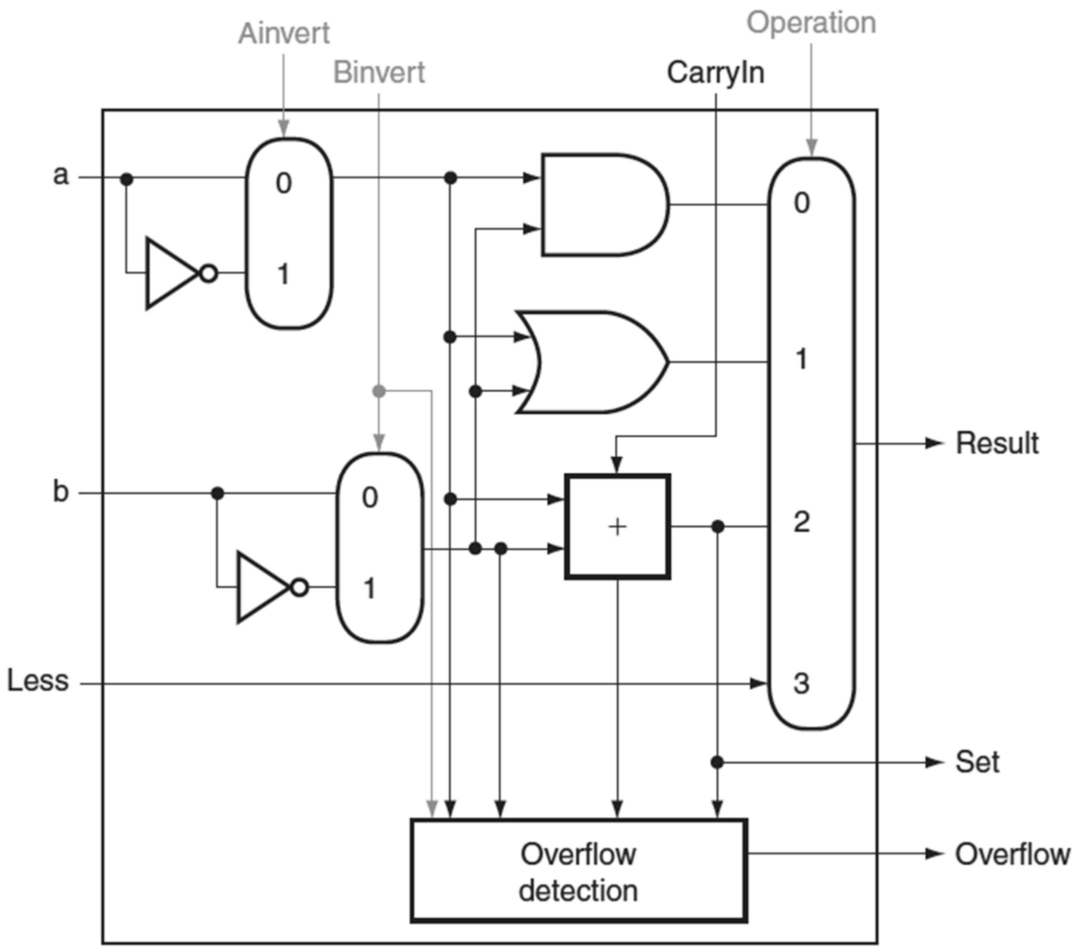


- First full adder, 2 gate delay
- Second full adder, 2 gate delay
- …

# Critical Path

Which operation is the critical path?

- A) ADD/SUB
- B) AND
- C) OR
- D) LT

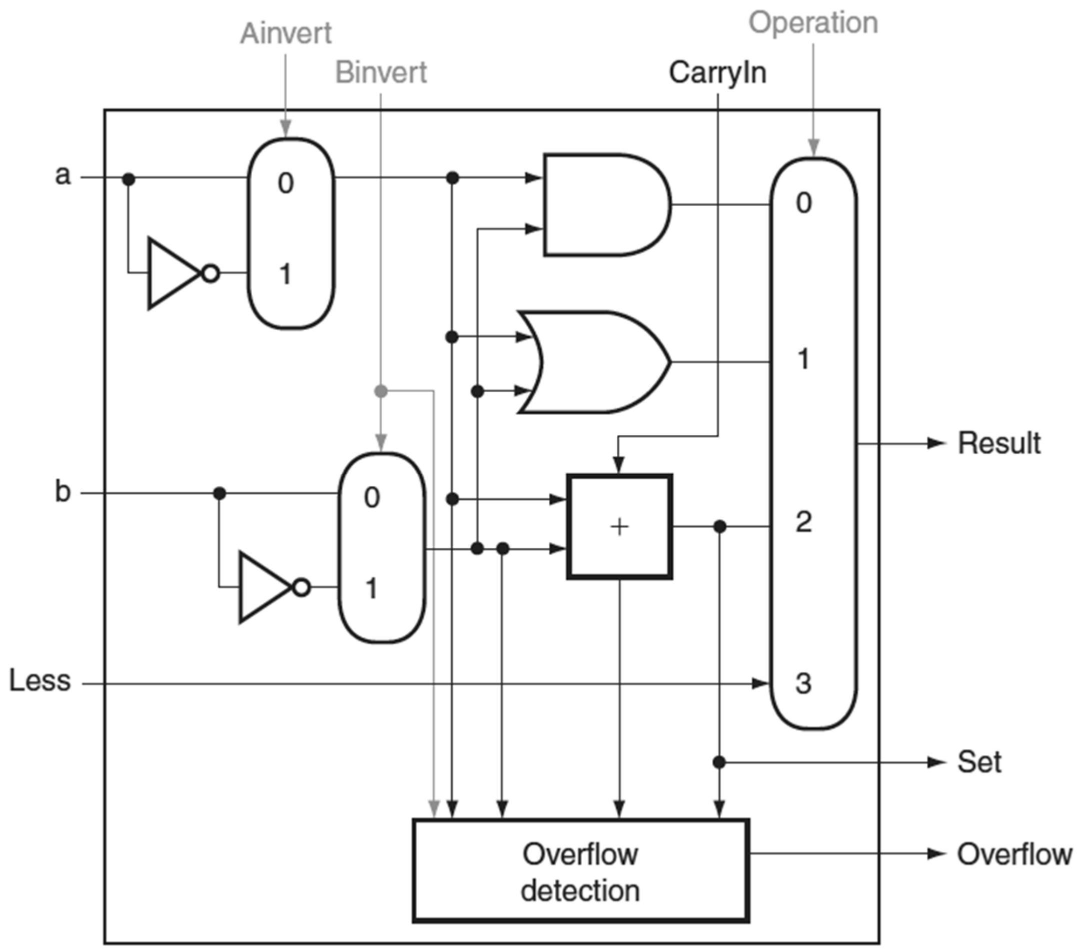


# Critical Path

What is the length of the critical path (in gates)?

(counting inverters)

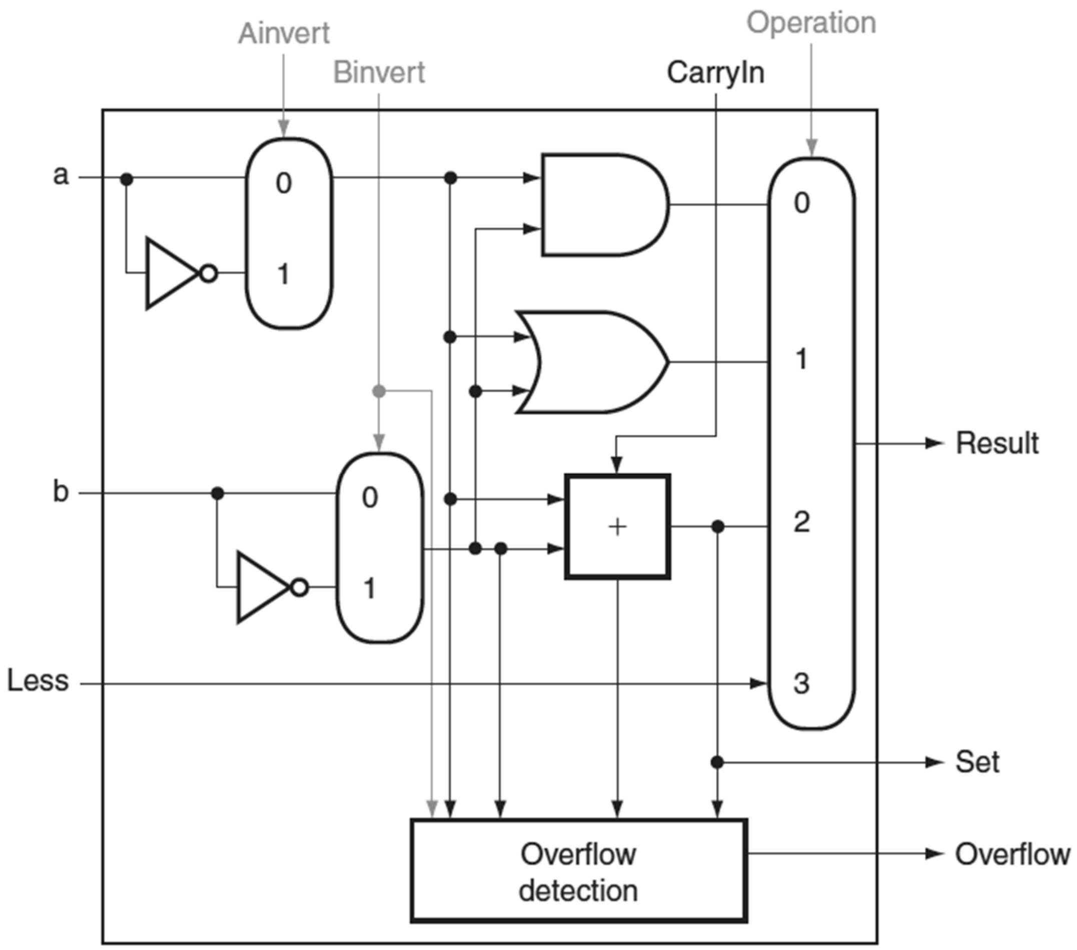
- A) 3
- B) 5
- C) 9
- D) 11



# Critical Path

What is the length of the critical path for a 32-bit ALU (in gates)? (counting inverters)

- A) 11
- B) 32
- C) 64
- D) 71



## Recap

We can now implement any combinational (combinatorial) logic circuit

- Decompose large circuit into manageable blocks
  - Encoders, Decoders, Multiplexors, Adders, ...
- Design each block
  - Binary encoded numbers for compactness
- Can implement circuits using NAND or NOR gates
- Can implement gates using use P- and N-transistors
- And can add and subtract numbers (in two’s compliment)!
- Next, state and finite state machines…

## Administrivia

Make sure you are

Registered for class, can access CMS

Have a Section you can go to

Have project partner in same Lab Section

Lab1 and HW1 are out

Both due in one week, next Monday, start early

Work alone

But, use your resources

- Lab Section, Piazza.com, Office Hours,  Homework Help Session,
- Class notes, book, Sections, CSUGLab

Homework Help Session

Wednesday and Friday from 3:30-5:30pm

Location: 203 Thurston

## Administrivia

Check online syllabus/schedule

- http://www.cs.cornell.edu/Courses/CS3410/2012sp/schedule.html

Slides and Reading for lectures

Office Hours

Homework and Programming Assignments

Prelims (in evenings):

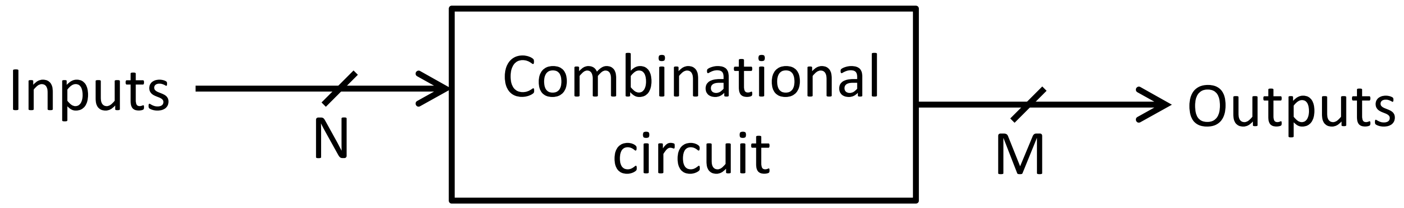
- Tuesday, February 28th
- Thursday, March 29th
- Thursday, April 26th

Schedule is subject to change

## Stateful Components

Until now is combinatorial logic

- Output is computed when inputs are present
- System has no internal state
- Nothing computed in the present can depend on what happened in the past!



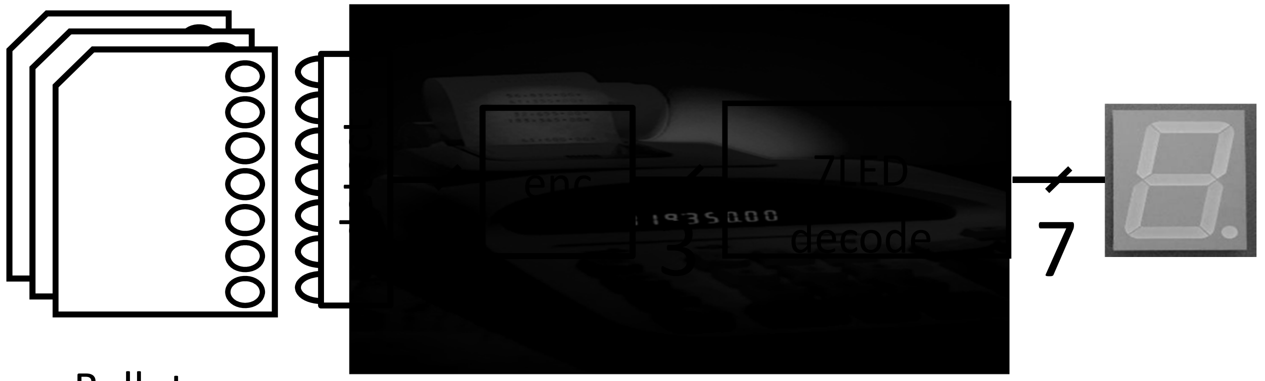
Need a way to record data

Need a way to build stateful circuits

Need a state-holding device

Finite State Machines

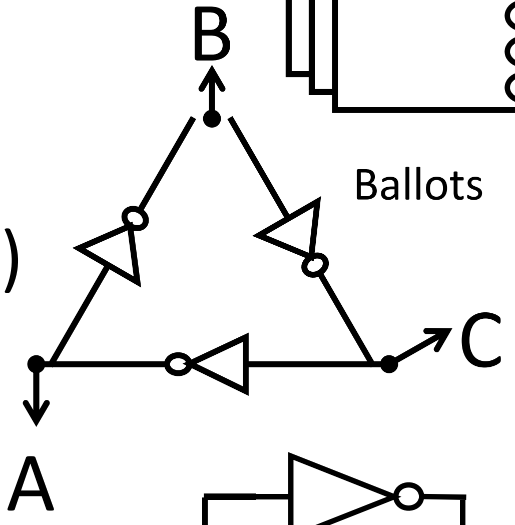
# How can we store ***and*** change values?



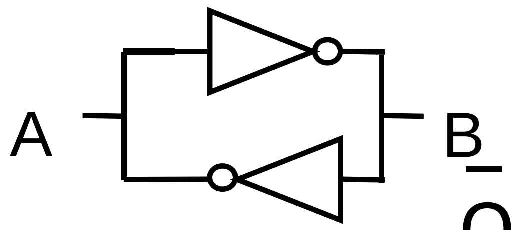
Ballots

How do we create vote counter machine

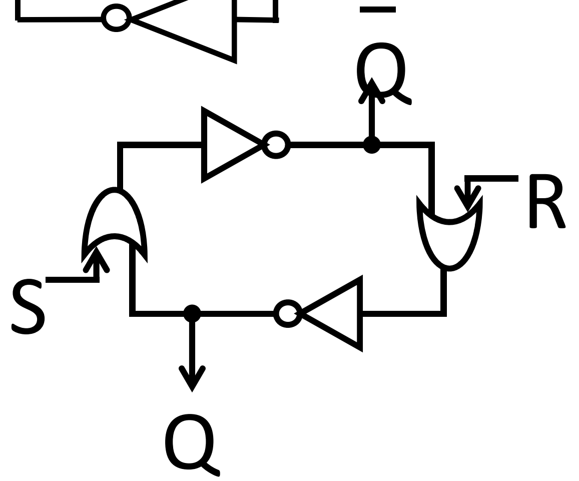
(a)



(b)



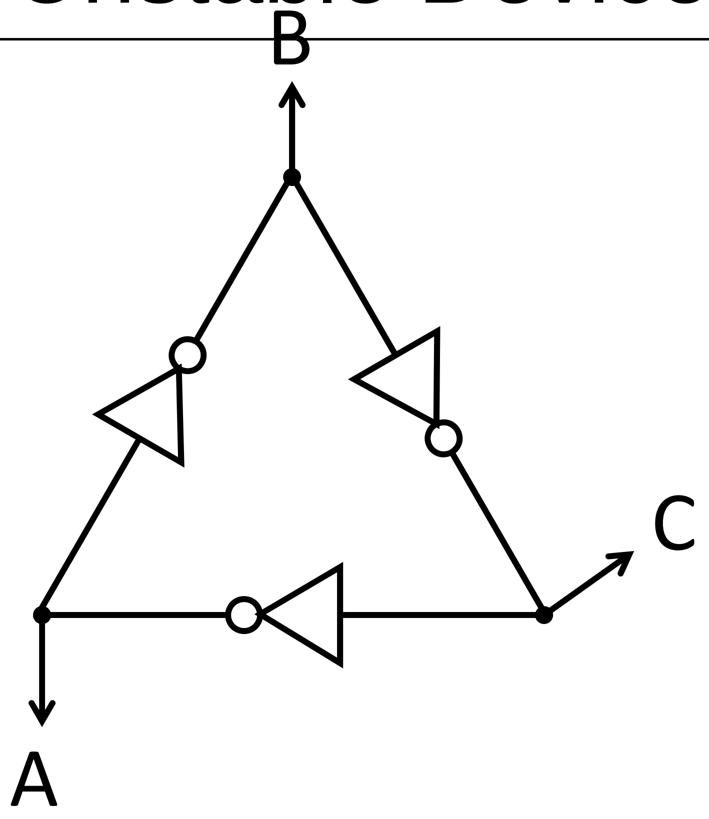
(c)



(d) All the above

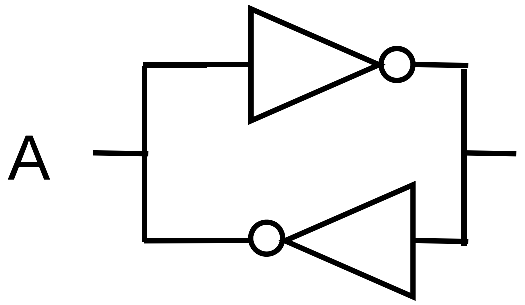
(e) None

## Unstable Devices



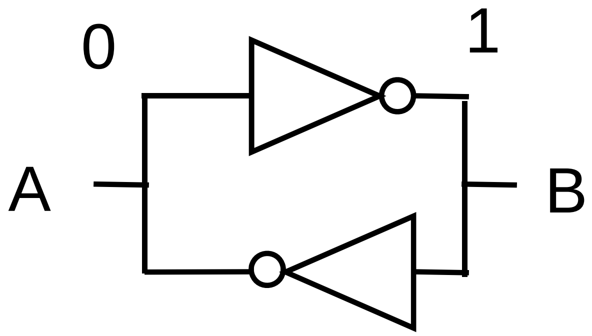
# Bistable Devices

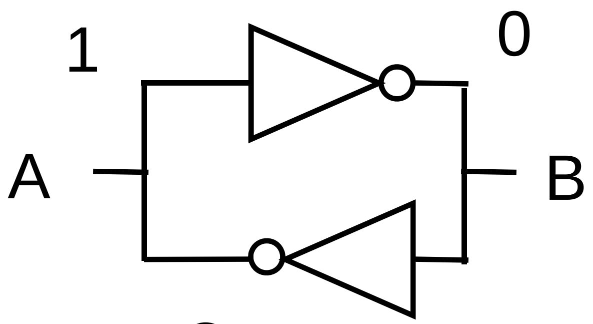
- Stable and unstable equilibria?



A Simple Device

- In stable state, $\bar{A} = B$



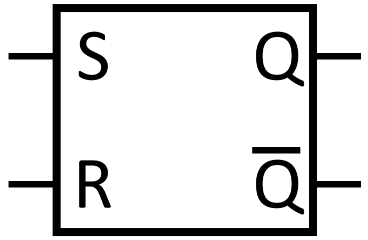


- How do we change the state?

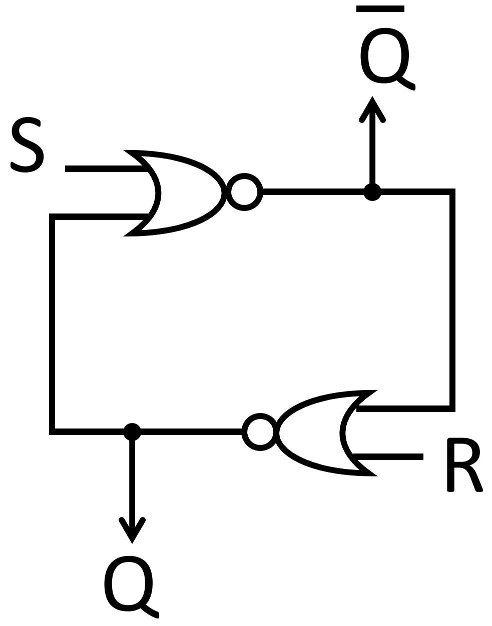
# SR Latch

Set-Reset (SR) Latch

Stores a value Q and its complement $\overline{Q}$



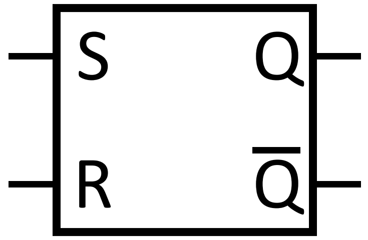
| S | R | Q | $\overline{Q}$ |
|---|---|---|---|
| 0 | 0 |  |  |
| 0 | 1 |  |  |
| 1 | 0 |  |  |
| 1 | 1 |  |  |



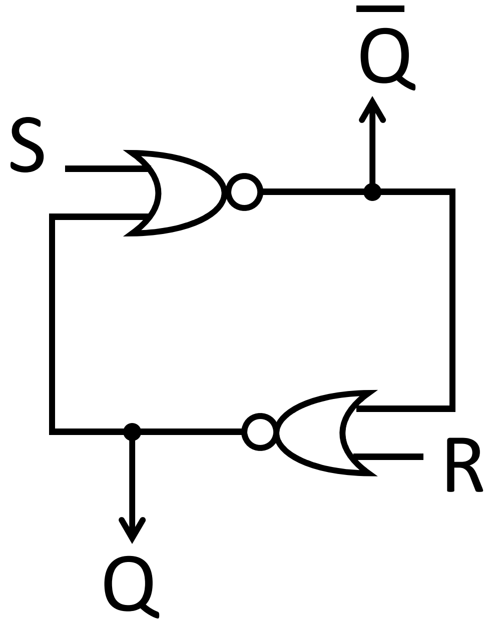
## SR Latch

Set-Reset (SR) Latch

- Stores a value Q and its complement $\overline{Q}$

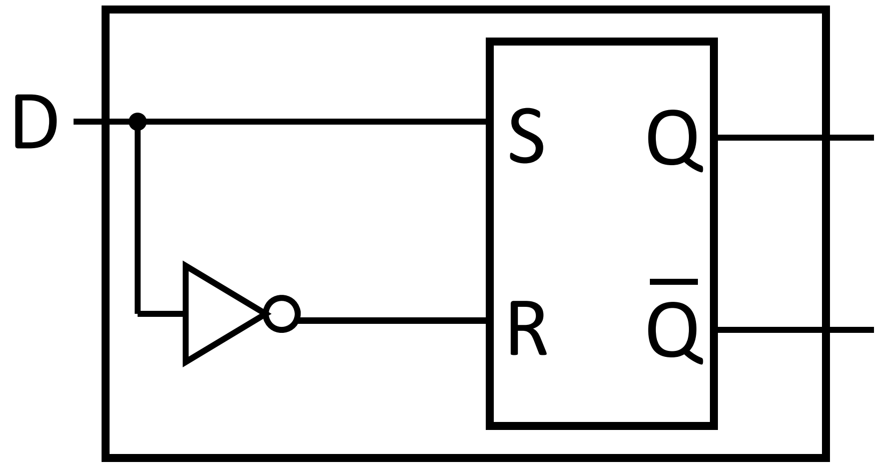


| S | R | Q | $\overline{Q}$ |
|---|---|---|---|
| 0 | 0 | | |
| 0 | 1 | | |
| 1 | 0 | | |
| 1 | 1 | | |



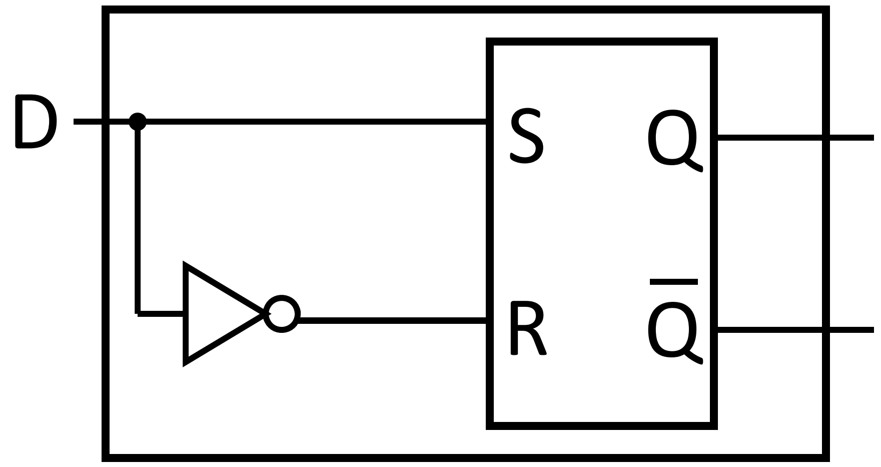
## Unclocked D Latch

Data (D) Latch



| D | Q | $\overline{Q}$ |
|---|---|---|
| 0 | | |
| 1 | | |

# Unclocked D Latch



Data (D) Latch

| D | Q | $\overline{Q}$ |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

Data Latch

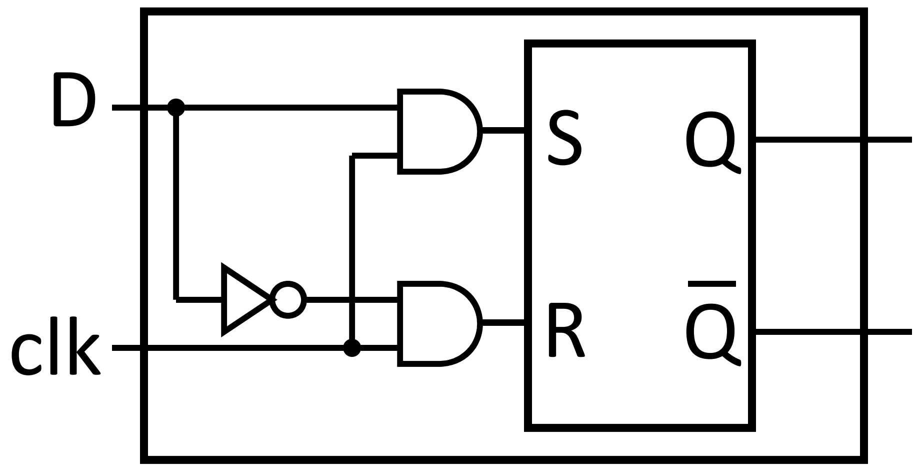
- Easier to use than an SR latch
- No possibility of entering an undefined state

When D changes, Q changes

  - … immediately (after a delay of 2 Ors and 2 NOTs)

Need to control when the output changes

# D Latch with Clock



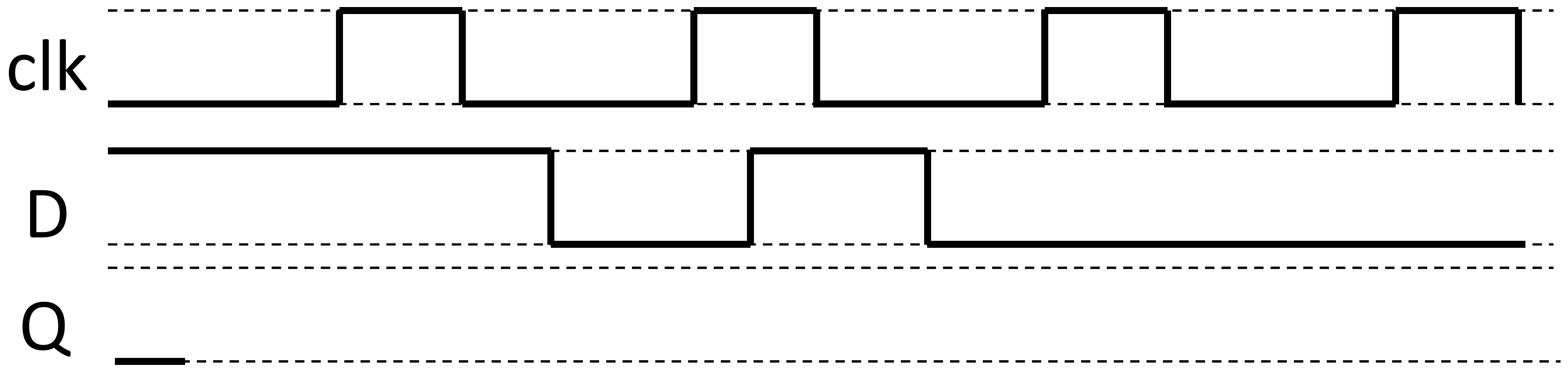
Level Sensitive D Latch

Clock high:

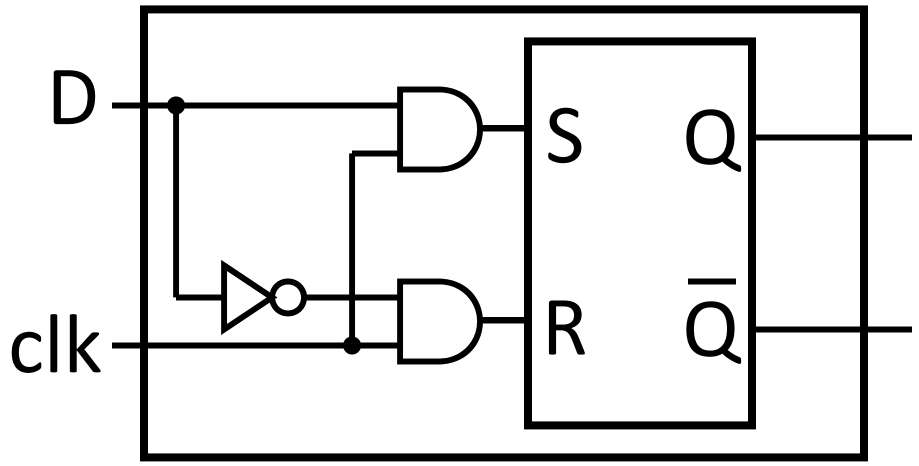
set/reset (according to D)

Clock low:

keep state (ignore D)



# D Latch with Clock

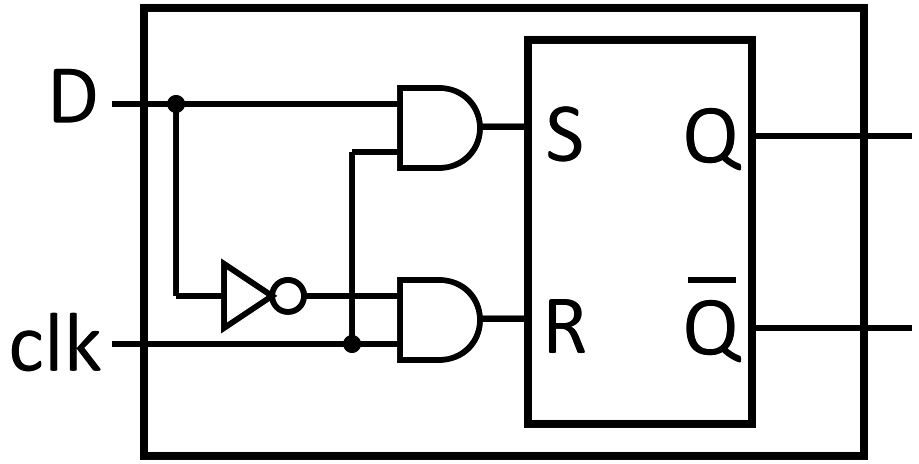


| D | Q | $\overline{Q}$ |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

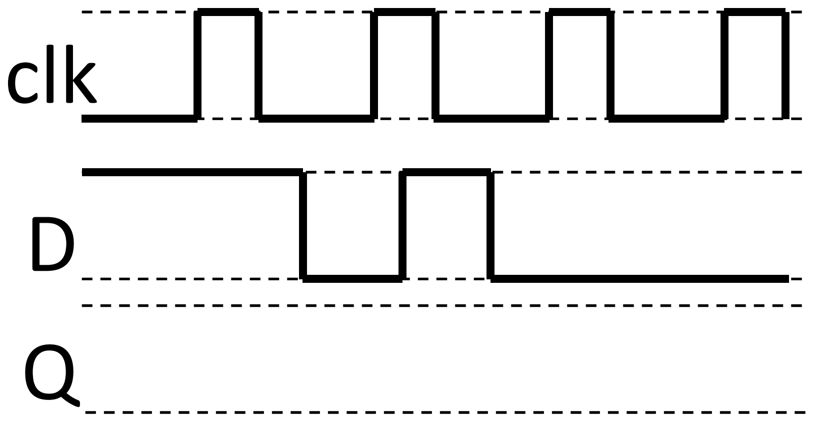
| S | R | Q | $\overline{Q}$ |
|---|---|---|---|
| 0 | 0 | Q | $\overline{Q}$ |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | forbidden | |

| clk | D | Q | $\overline{Q}$ |
|---|---|---|---|
| 0 | 0 | Q | $\overline{Q}$ |
| 0 | 1 | Q | $\overline{Q}$ |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

# D Latch with Clock



| D | Q | $\overline{Q}$ |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

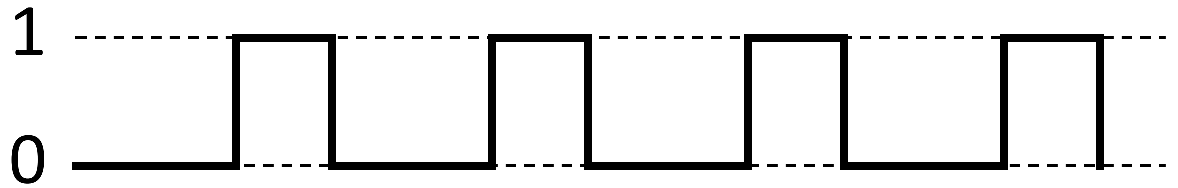


| clk | D | Q | $\overline{Q}$ |
|---|---|---|---|
| 0 | 0 | Q | $\overline{Q}$ |
| 0 | 1 | Q | $\overline{Q}$ |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

# Clocks

Clock helps coordinate state changes

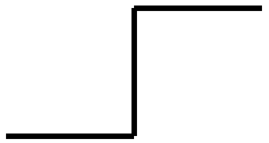
- Usually generated by an oscillating crystal
- Fixed period; frequency = 1/period



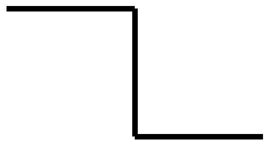
# Edge-triggering

- Can design circuits to change on the rising or falling edge

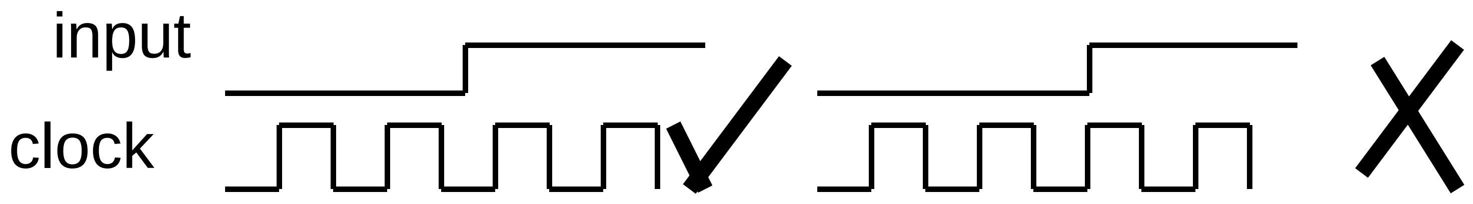
- Trigger on rising edge = positive edge-triggered



- Trigger on falling edge = negative edge-triggered



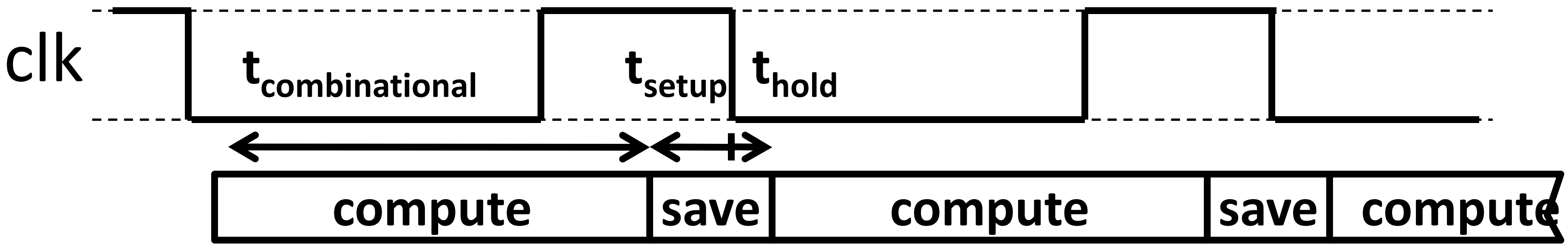
- Inputs must be stable just before the triggering edge



# Clock Methodology

## Clock Methodology

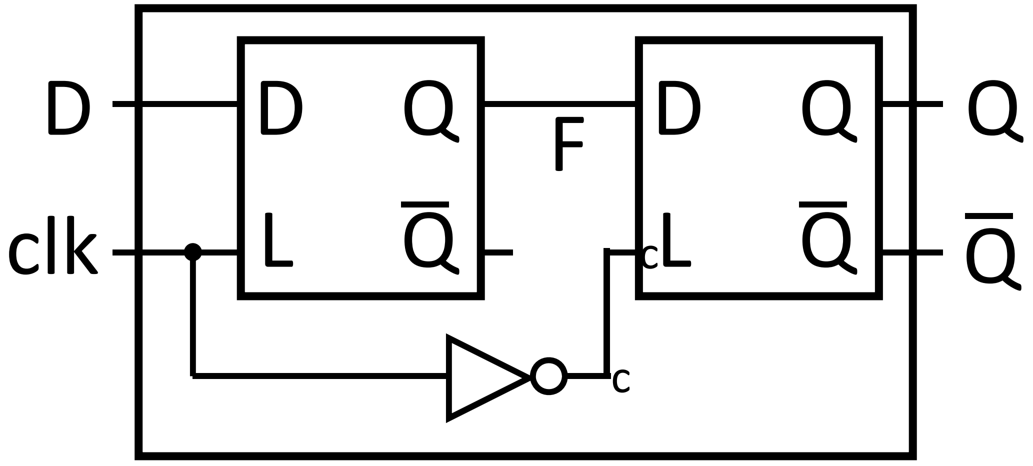
- Negative edge, synchronous



  - – Signals must be stable near falling clock edge

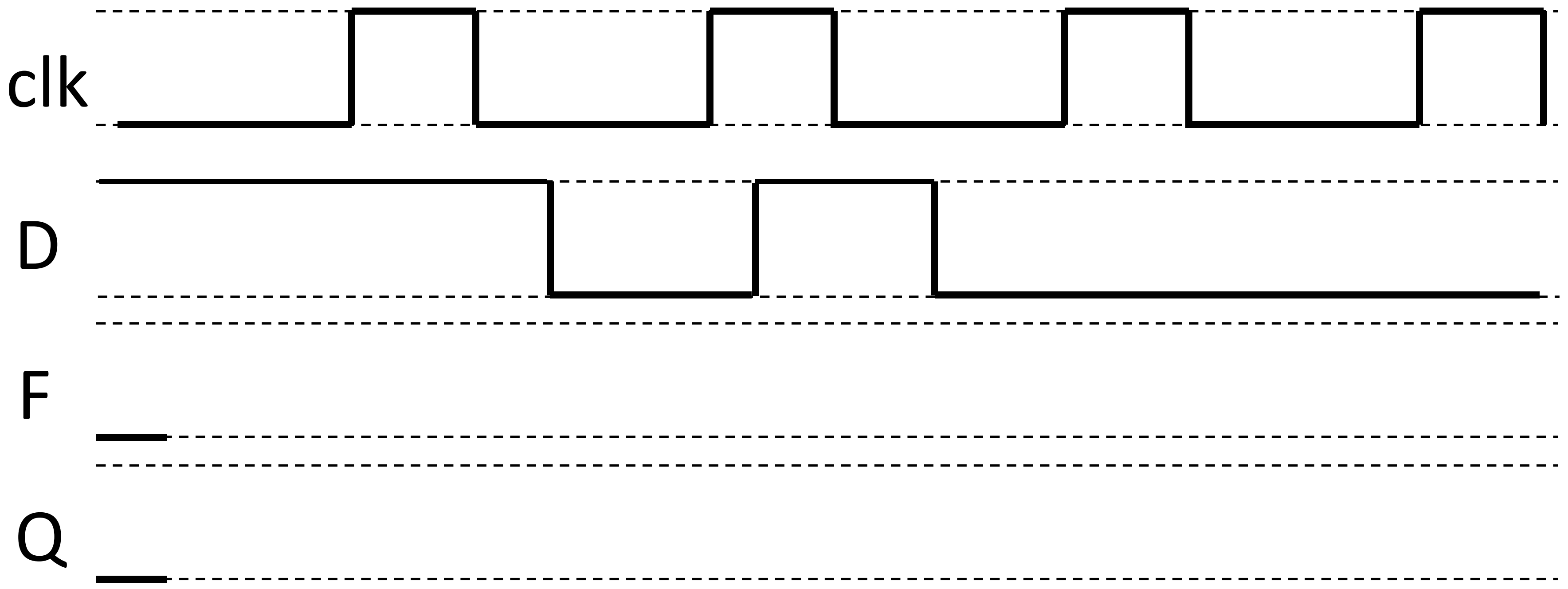
- Positive edge synchronous
- Asynchronous,  multiple clocks, . . .

# Edge-Triggered D Flip-Flop



D Flip-Flop

- Edge-Triggered
- Data is captured when clock is high
- Outputs change only on falling edges



# Clock Disciplines

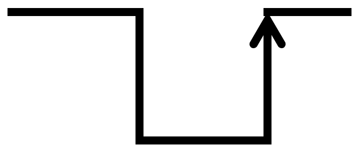
Level sensitive

- State changes when clock is high (or low)

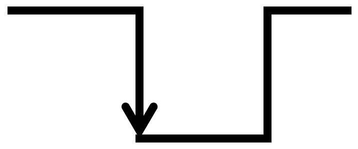
Edge triggered

- State changes at clock edge

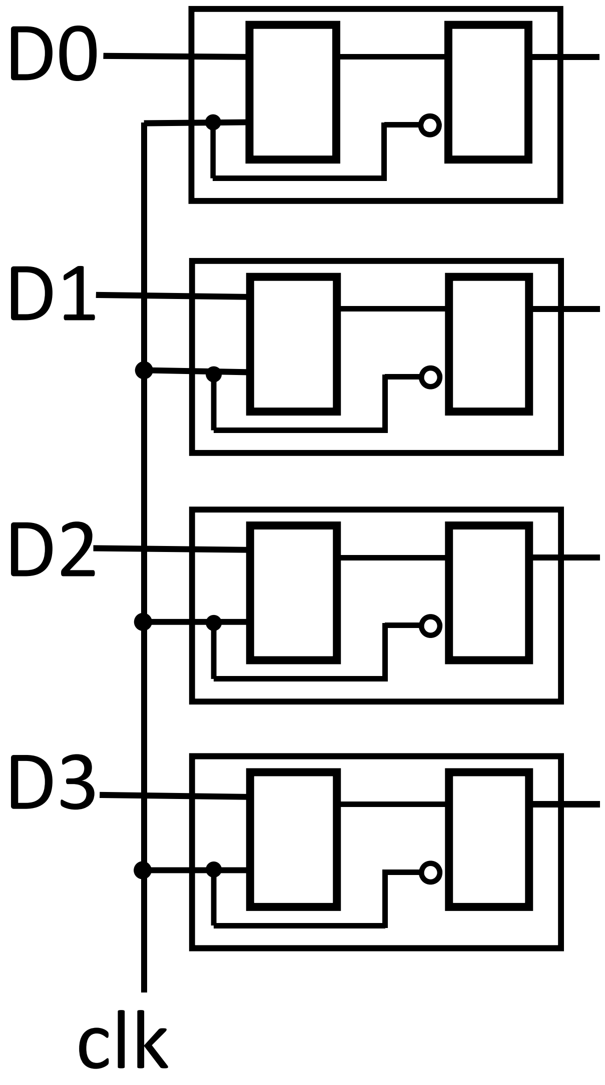
positive edge-triggered



negative edge-triggered

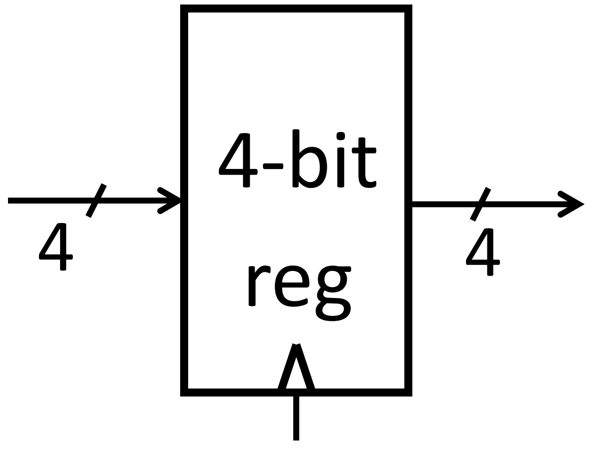


# Registers

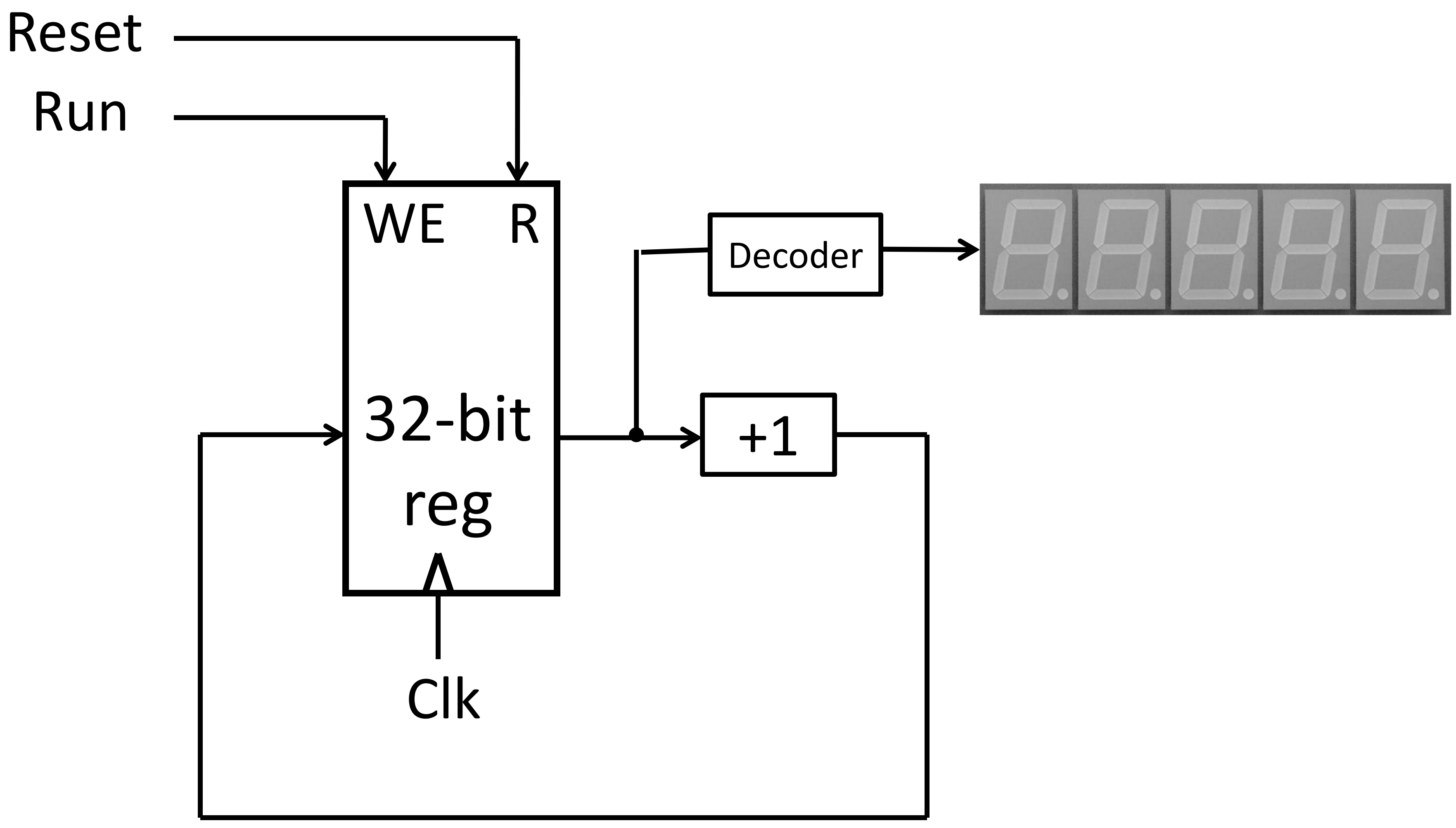


Register

- D flip-flops in parallel
- shared clock
- extra clocked inputs: write_enable, reset, …



# An Example: What will this circuit do?


Reset
Run
WE
R
32-bit reg
Clk
Decoder
+1