More bits, circuits, adders (From the last class)

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

- Homework 1 due Monday

Adder

Can we use this in parallel to add multi-bit numbers?

What is missing?

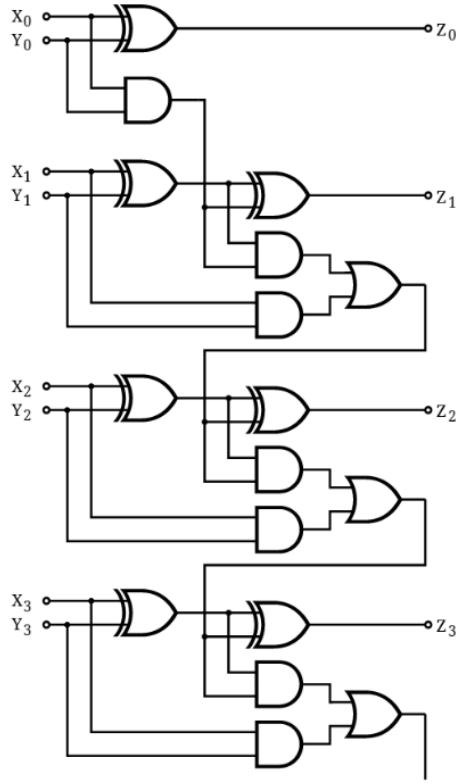
Consider:

```
  11
+01
----
```

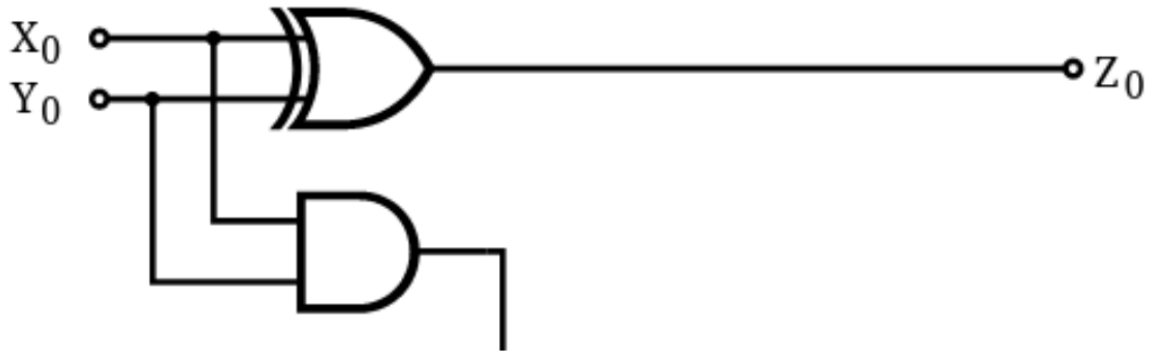
3-input Adder

Add 3 1-bit numbers: a, b, c

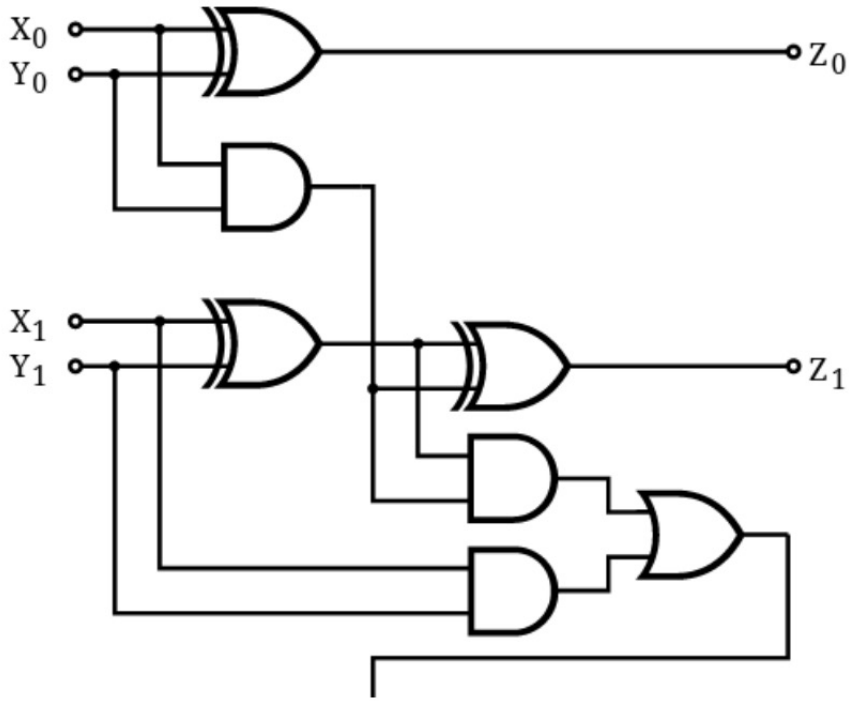
Ripple-Carry Adder



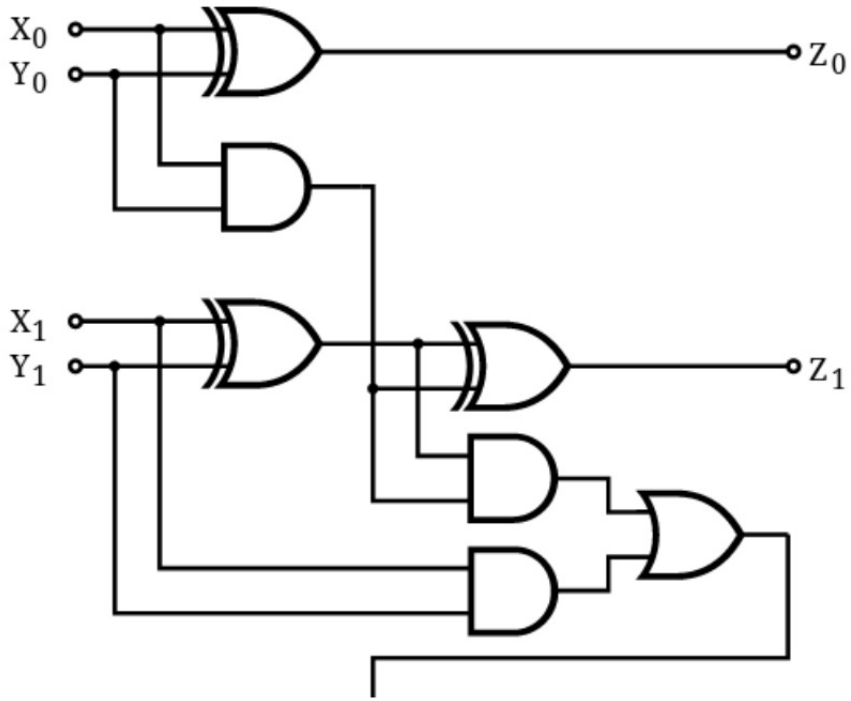
Ripple-Carry Adder: Lowest-order Bit



Ripple-Carry Adder: In General



Ripple-Carry Adder: In General



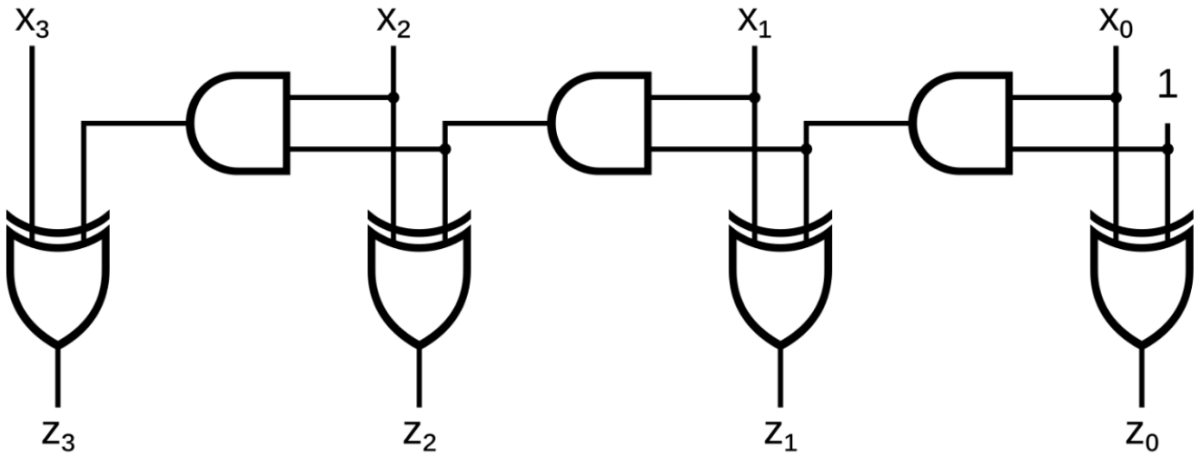
Clocks, Registers

CS 2130: Computer Systems and Organization 1

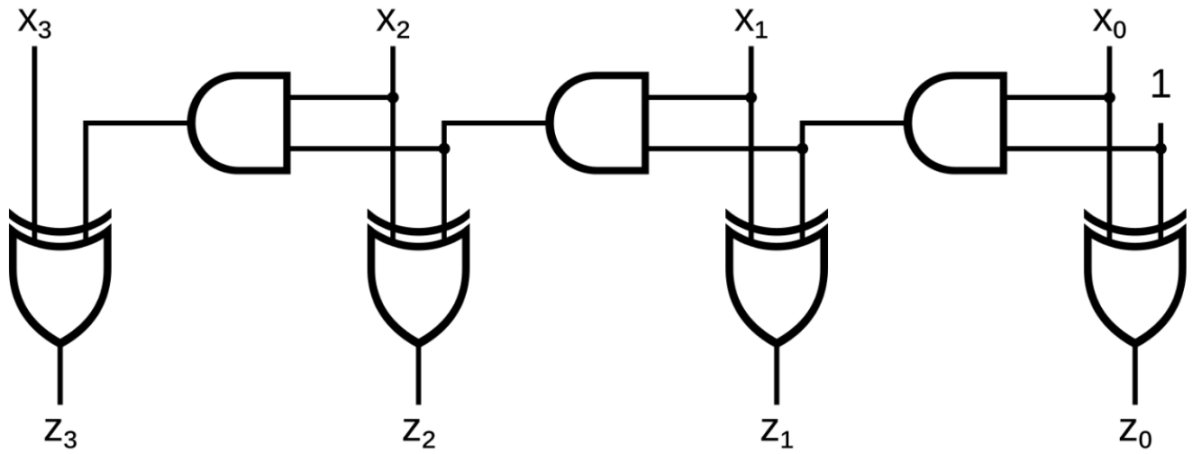
Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

What does this circuit do?

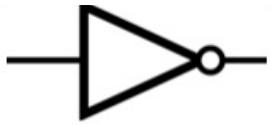


Increment Circuit



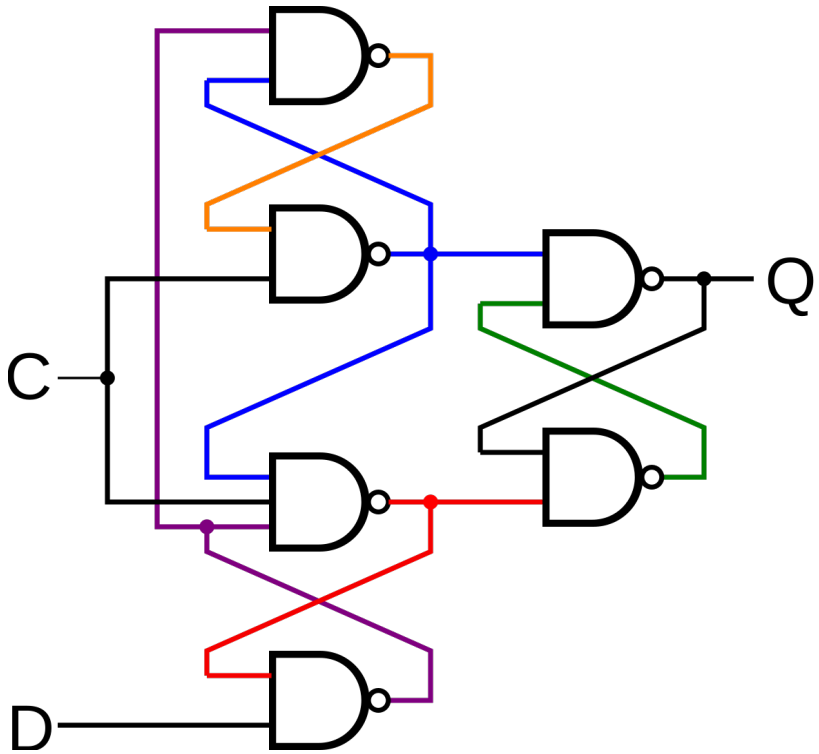
Gate Delay

What happens when I change my input?

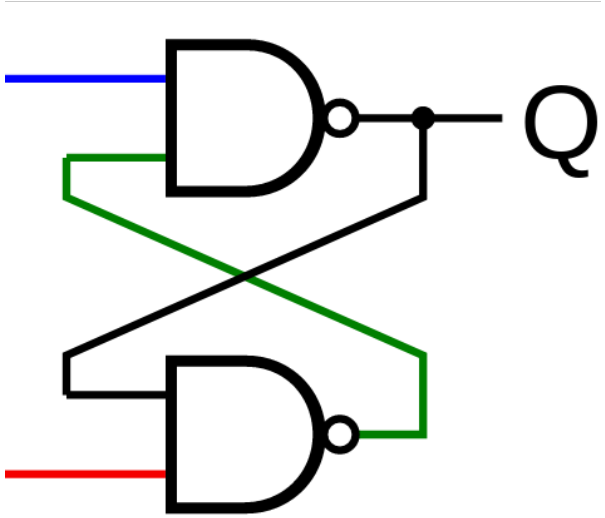


Building a Counter

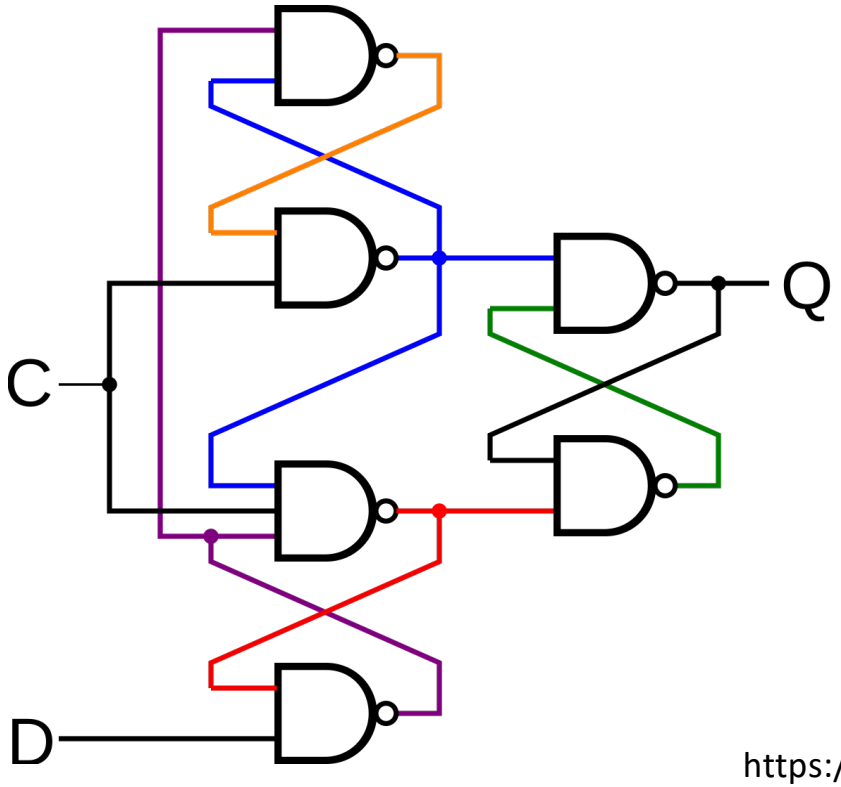
1-bit Register Circuit



1-bit Register Circuit



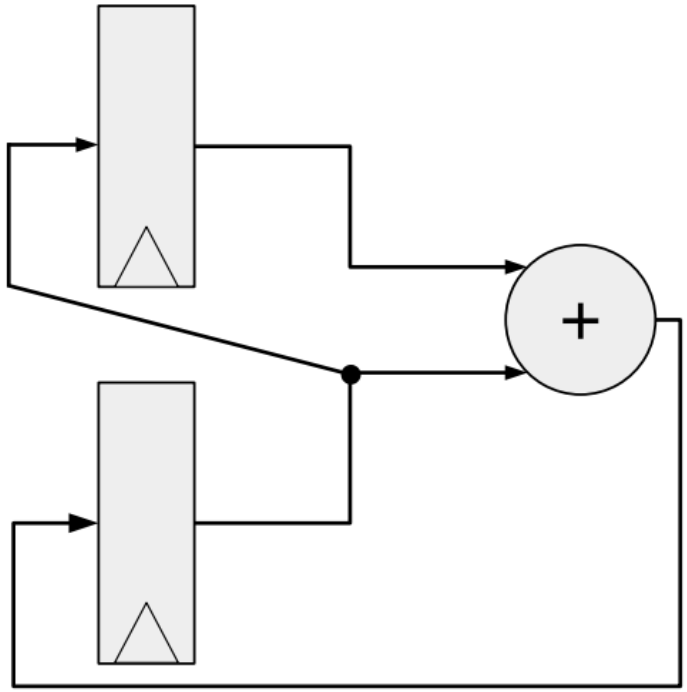
1-bit Register Circuit



https://www.falstad.com/circuit/e-edgedff.html

Building a Counter

Another Counter



Common Model in Computers

Code to Build Circuits from Gates

Write code to build circuits from gates

- Gates we already know: &, |, ^, ~
- Operations we can build from gates: +, -
- Others we can build:

Code to Build Circuits from Gates

Write code to build circuits from gates

- Gates we already know: &, |, ^, ~
- Operations we can build from gates: +, -
- Others we can build:
- Ternary operator: ? :

Equals

Equals: =

- Attach with a wire (i.e., connect things)
- Ex: z = x * y

Equals

Equals: =

- Attach with a wire (i.e., connect things)
- Ex: z = x * y
- What about the following?

  x = 1

  x = 0

Equals

Equals: =

- Attach with a wire (i.e., connect things)
- Ex: z = x * y
- What about the following?

  x = 1

  x = 0
- **Single assignment:** each variable can only be assigned a value once

Subtraction

$$z = x + \sim y + 1$$

$$a = \sim y$$
$$b = a + 1$$
$$z = x + b$$

Comparisons

Each of our comparisons in code are straightforward to build:

- == - xor then nor bits of output

Comparisons

Each of our comparisons in code are straightforward to build:

- == - xor then nor bits of output
- != - same as == without not of output

Comparisons

Each of our comparisons in code are straightforward to build:

- == - xor then nor bits of output
- != - same as == without not of output
- < - consider x < 0

Comparisons

Each of our comparisons in code are straightforward to build:

- == - xor then nor bits of output
- != - same as == without not of output
- < - consider x < 0
- >, <=, => are similar

Any Questions?