



# Circuits and Code

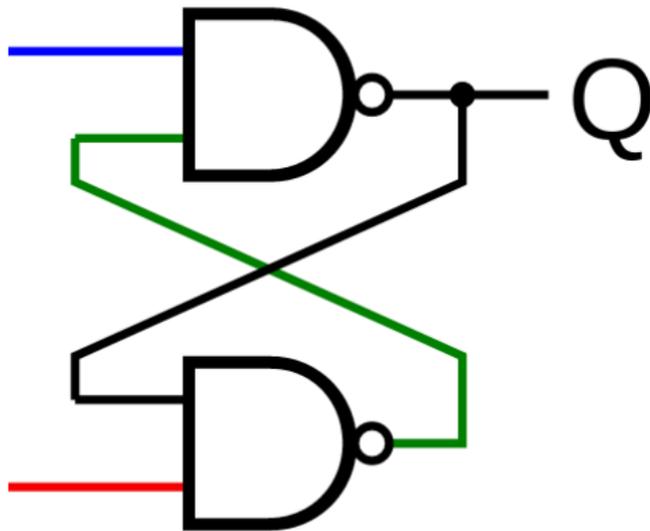
CS 2130: Computer Systems and Organization 1  
February 2, 2026

# Announcements

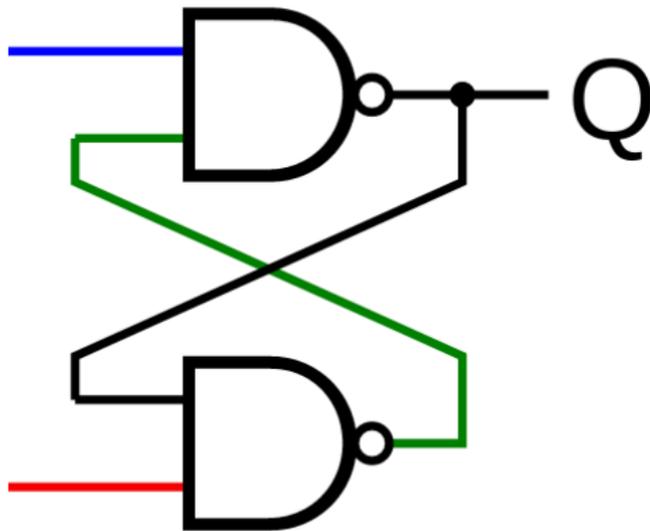
- Homework 1 due tonight
- Homework 2 available today, due next Monday



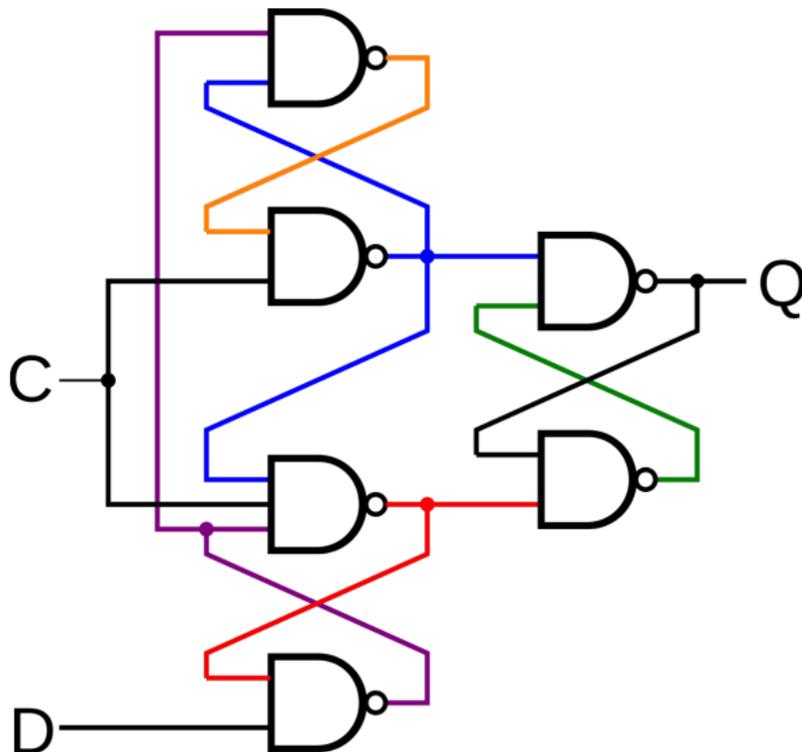
# 1-bit Register Circuit



# 1-bit Register Circuit



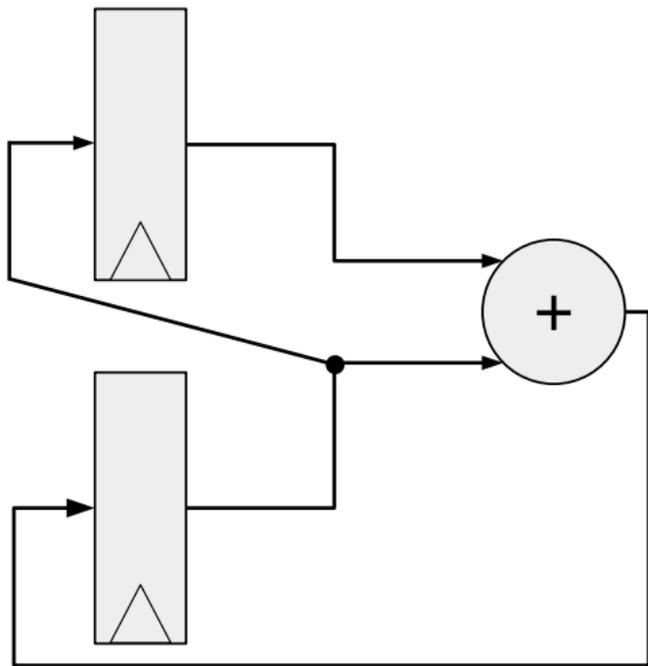
# 1-bit Register Circuit



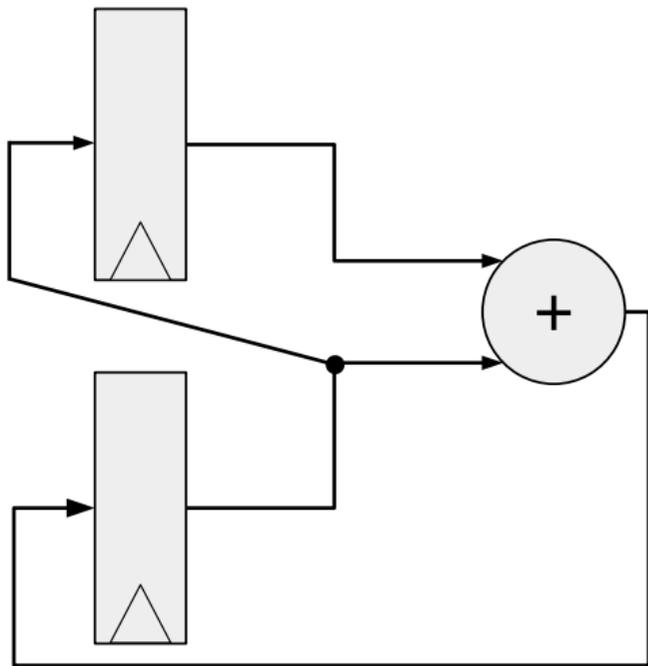
# Building a Counter



# Another Circuit



# Another Circuit



# Common Model in Computers

# Code to Build Circuits from Gates

Write code to build circuits from gates

- Gates we *already* know:  $\&$ ,  $|$ ,  $\wedge$ ,  $\sim$
- 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:  $\&$ ,  $|$ ,  $\wedge$ ,  $\sim$
- 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

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

$$a = \sim y$$

$$b = a + 1$$

$$z = x + y$$

# 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

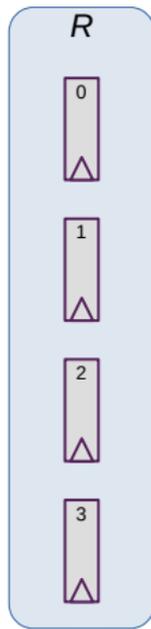
# Indexing

Indexing with square brackets: [ ]

- **Register bank** (or **register file**) - an array of registers
  - Can programmatically pick one based on index
  - I.e., can determine which register while running
- Two important operations:
  - $x = R[i]$  - Read from a register
  - $R[j] = y$  - Write to a register

# Reading

$x = R[i]$  - connect output of registers to  $x$  based on index  $i$

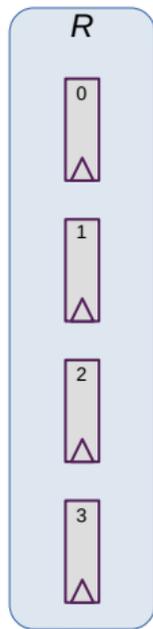


# Aside: 4-input Mux

How do we build a 4-input mux? How many wires should  $i$  be?

# Writing

$R[j] = y$  - connect  $y$  to input of registers based on index  $j$



# Aside: Creating $==0$ gates

How do we build gates that check for  $j == w$ ?

Need one more thing to build computers

# Memory and Storage

## Registers

≈ KiB

- 6 gates each, ≈ 24 transistors
- Efficient, fast
- Expensive!
- Ex: local variables

*These do not persist between power cycles*

# Memory and Storage

## Memory

≈ GiB

- Two main types: SRAM, DRAM
- DRAM: 1 transistor, 1 capacitor per bit
- DRAM is cheaper, simpler to build
- Ex: data structures, local variables

*These do not persist between power cycles*

# Memory and Storage

## Disk

≈ GiB-TiB

- Two main types: flash (solid state), magnetic disk
- Magnetic drive
  - Platter with physical arm above and below
  - Cheap to build
  - Very slow! Physically move arm while disk spins
- Ex: files

*Data on disk does persist between power cycles*

Putting it all together  
Next time!