Building to a Computer

CS 2130: Computer Systems and Organization 1
February 4, 2026

Announcements

- Homework 2 due Monday

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

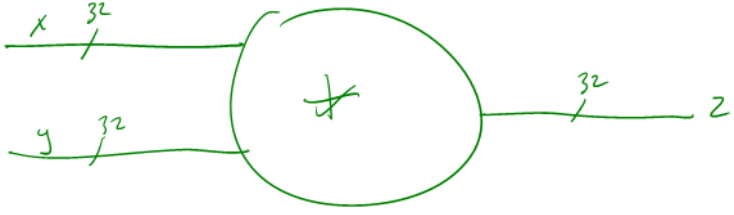
- Gates we *already* know: `&`, `|`, `^`, `~`
- Operations we can build from gates: `+`, `-`
- Others we can build: `*` `&&`, `<<`, `+`
- Ternary operator: `?` `:`

```
z = (a==b) ? x : y;
```

Equals

Equals: =

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

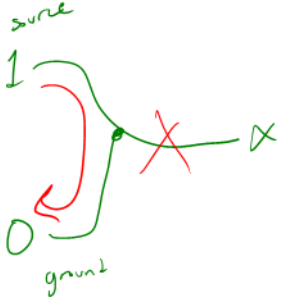


Equals

Equals: `=`

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

```
x = 1
x = 0
```



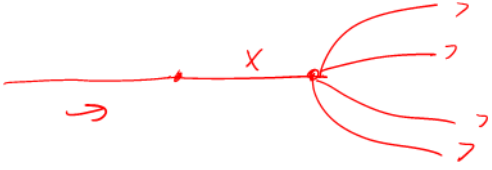
Equals

Equals: `=`

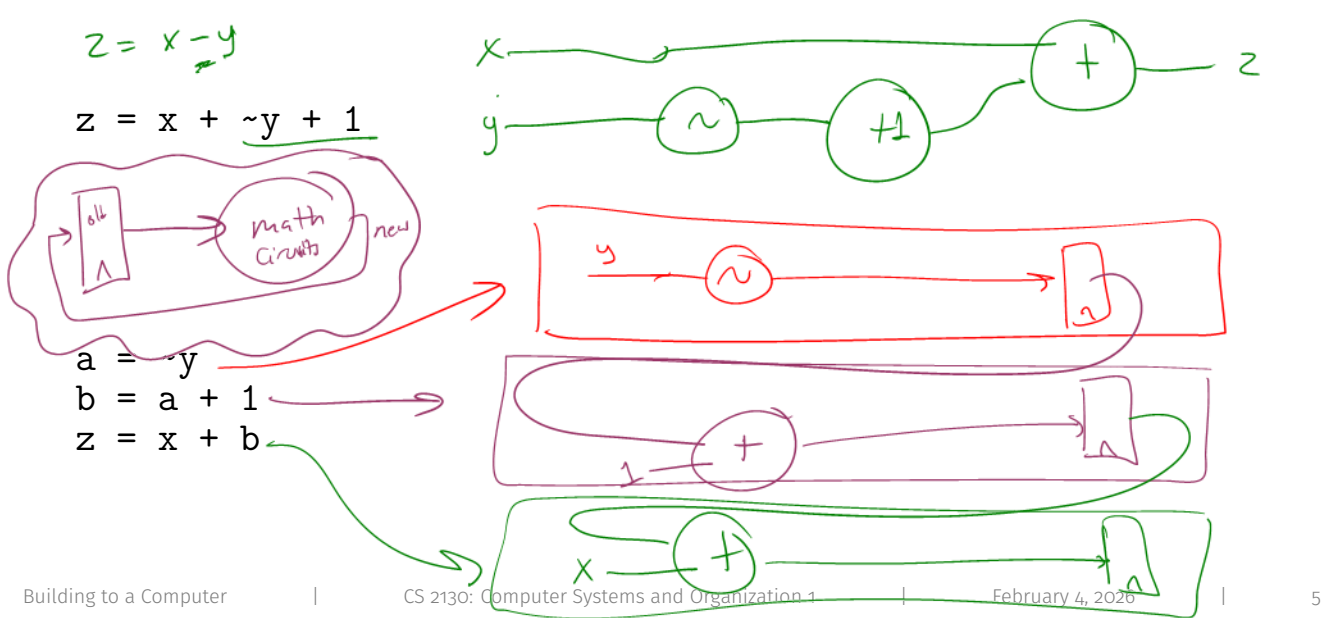
- Attach with a wire (i.e., connect things)
- Ex: `z = x * y`
- What about the following?

```
x = 1
x = 0
```

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



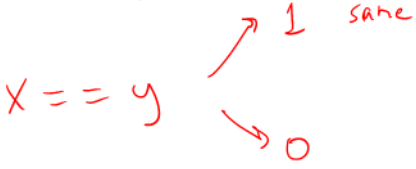
Subtraction

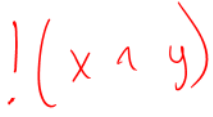


Comparisons

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:

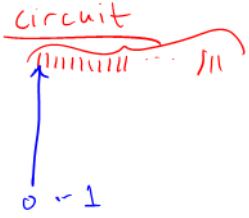
- `==` - xor then nor bits of output
- `!=` - same as `==` without not of output
- `<` - consider `x < 0`

$x < y \longrightarrow x - y < 0$

x is 32-bits

code

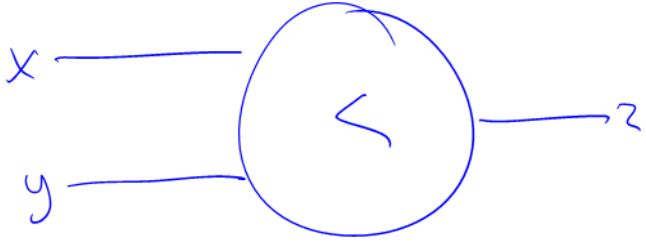
`(x >> 31) & 1`



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

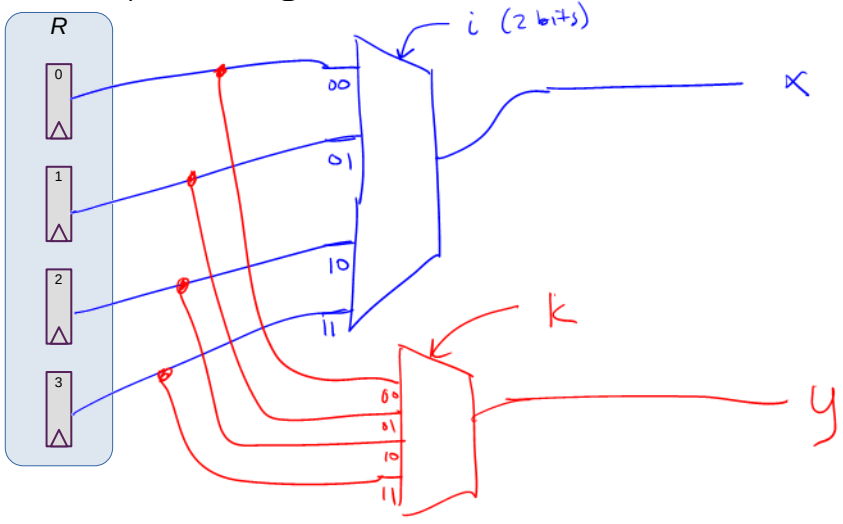
Indexing with square brackets: `[ ]`

- **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

R[1] = R[1] + R[2]

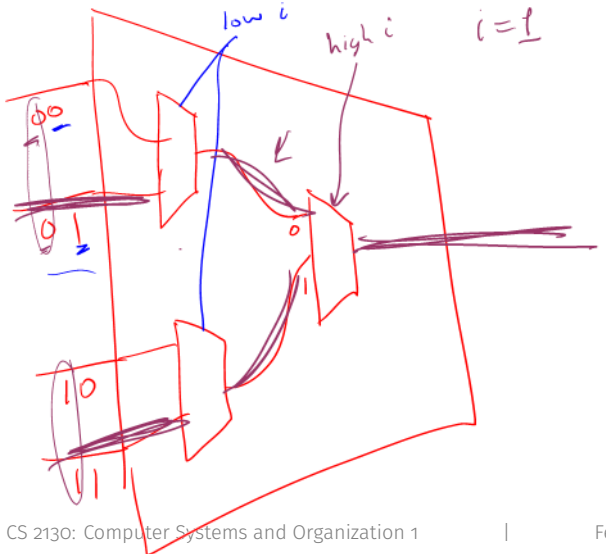
Reading

`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*