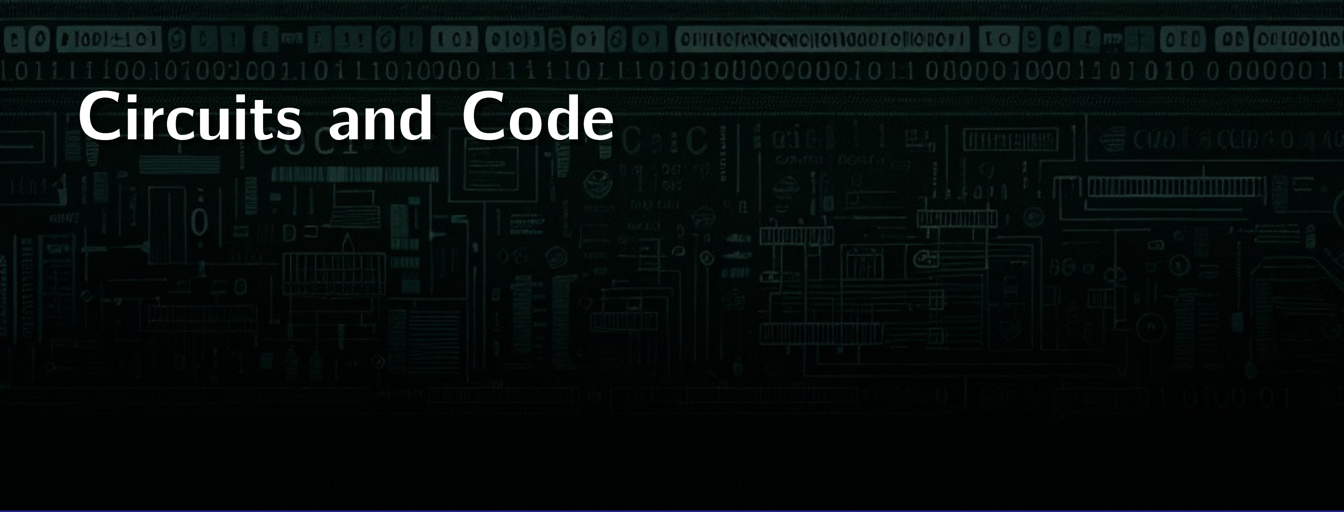


Circuits and Code

CS 2130: Computer Systems and Organization 1
September 15, 2025

Announcements

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

31 = 0 ·· 0 1 1 1 1 1

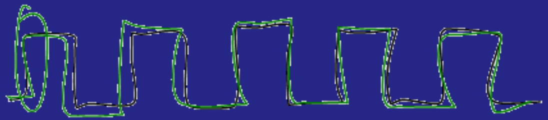
32 16 8 4 2 1

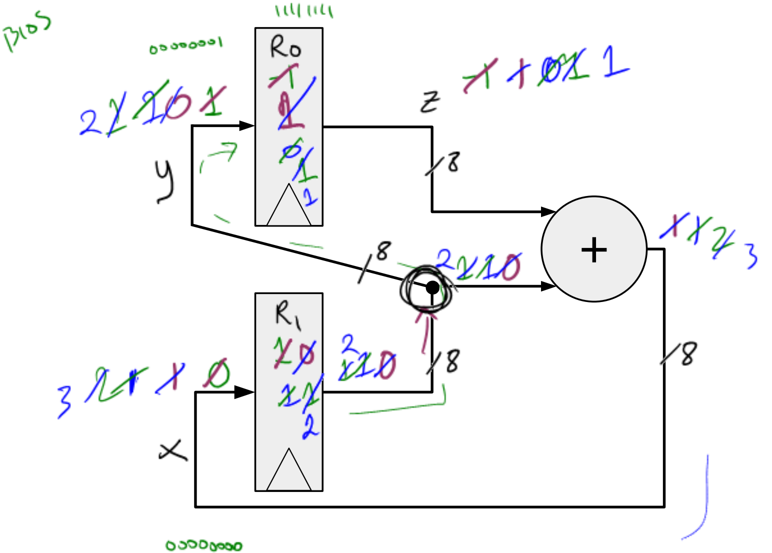
a = 000 ······ 01 << 31

c = 100 ······ 0 >> 31

111 ······ 1

Another Circuit



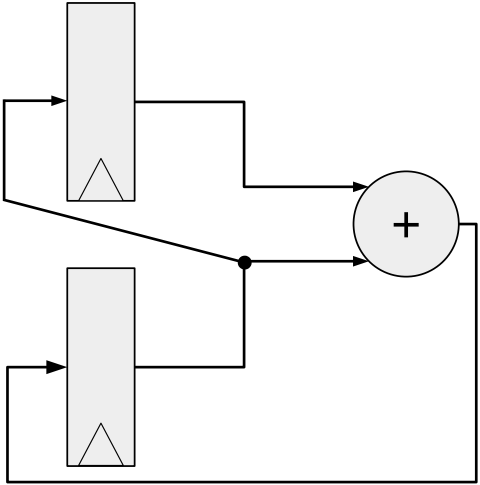


tick	x	y	z	R0	R1
0	0	1	-1	-1	1
1	1	0	1	1	0
2	1	1	0	0	1
3	2	1	1	1	1
4	3	2	1	1	2
		3			

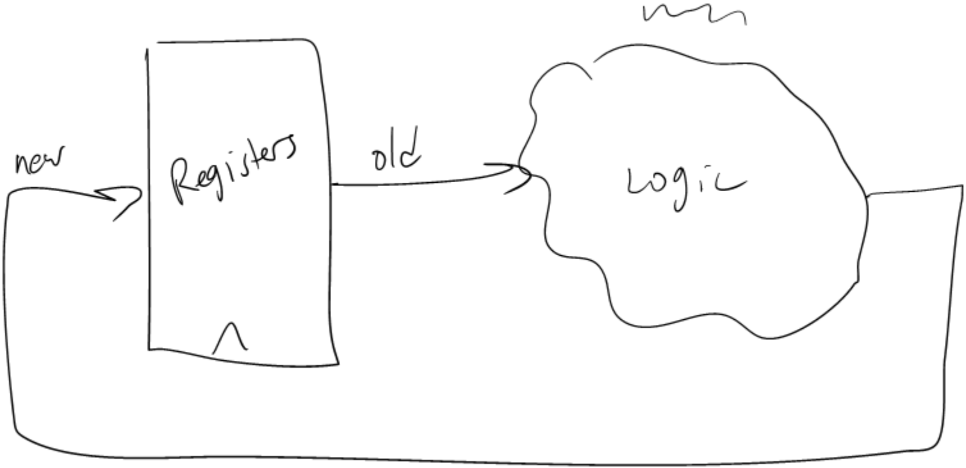
y = x

fibonacci → ~~x~~ x = x(old) + y(old)

Another Circuit



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: *

$$\begin{array}{r} 2130 \\ \times\ 1101 \\ \hline 2130 \\ 00000 \\ 213000 \\ +\ 2130000 \end{array}$$

← 6

<<

/, %

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: `?` `:`

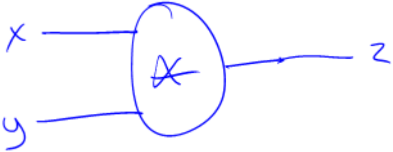
z= a ? b : c

w = (a==b ? 32 : y) * x

Equals

Equals: =

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

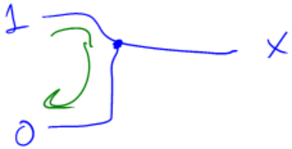


Equals

Equals: =

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

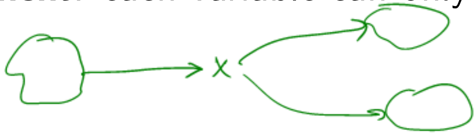
```
x = 1
x = 0
```



Equals

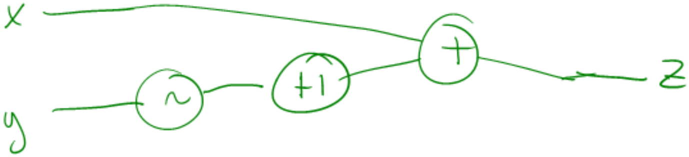
Equals: `=`

- 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

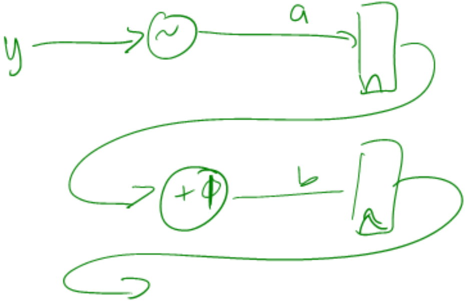


Subtraction

```
z = x + ~y + 1
```



```
a = ~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

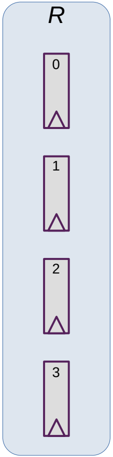
Indexing

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*

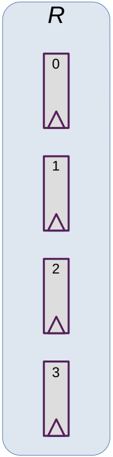


Aside: 4-input Mux

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 $\approx$ KiB

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

These do not persist between power cycles

Memory and Storage

Memory $\approx$ 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 $\approx$ 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!