04-binary-notes

Binary

Agenda

- 0. Re-Orienting
- 1. Binary
- 2. Comparing two binary numbers
- 3. Doing other things with a binary number
- 4. Generalizing

---

0. Re-Orienting

Electricity

---

0 and 1

Logic gates

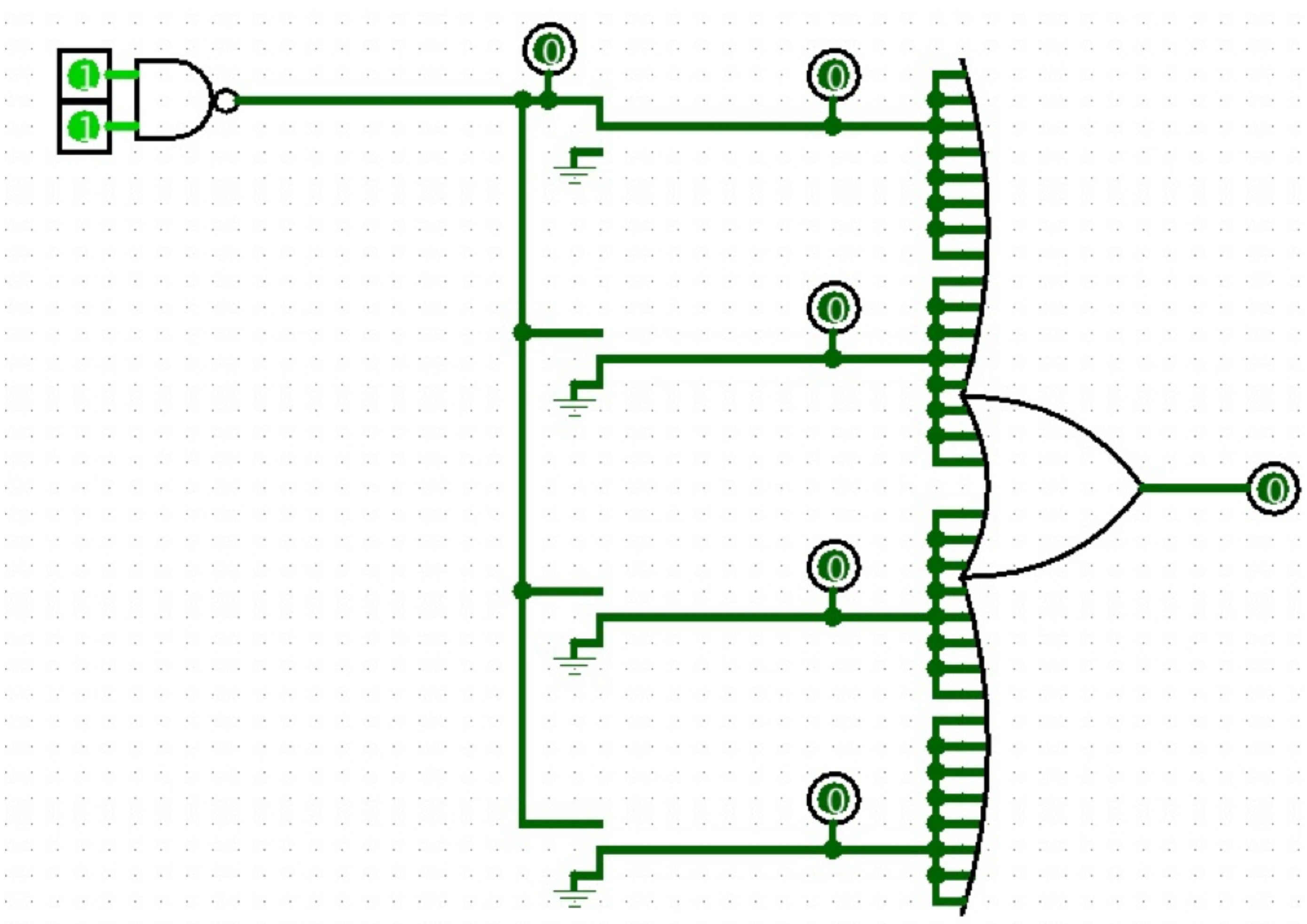
Recall: building logic gates from transistors

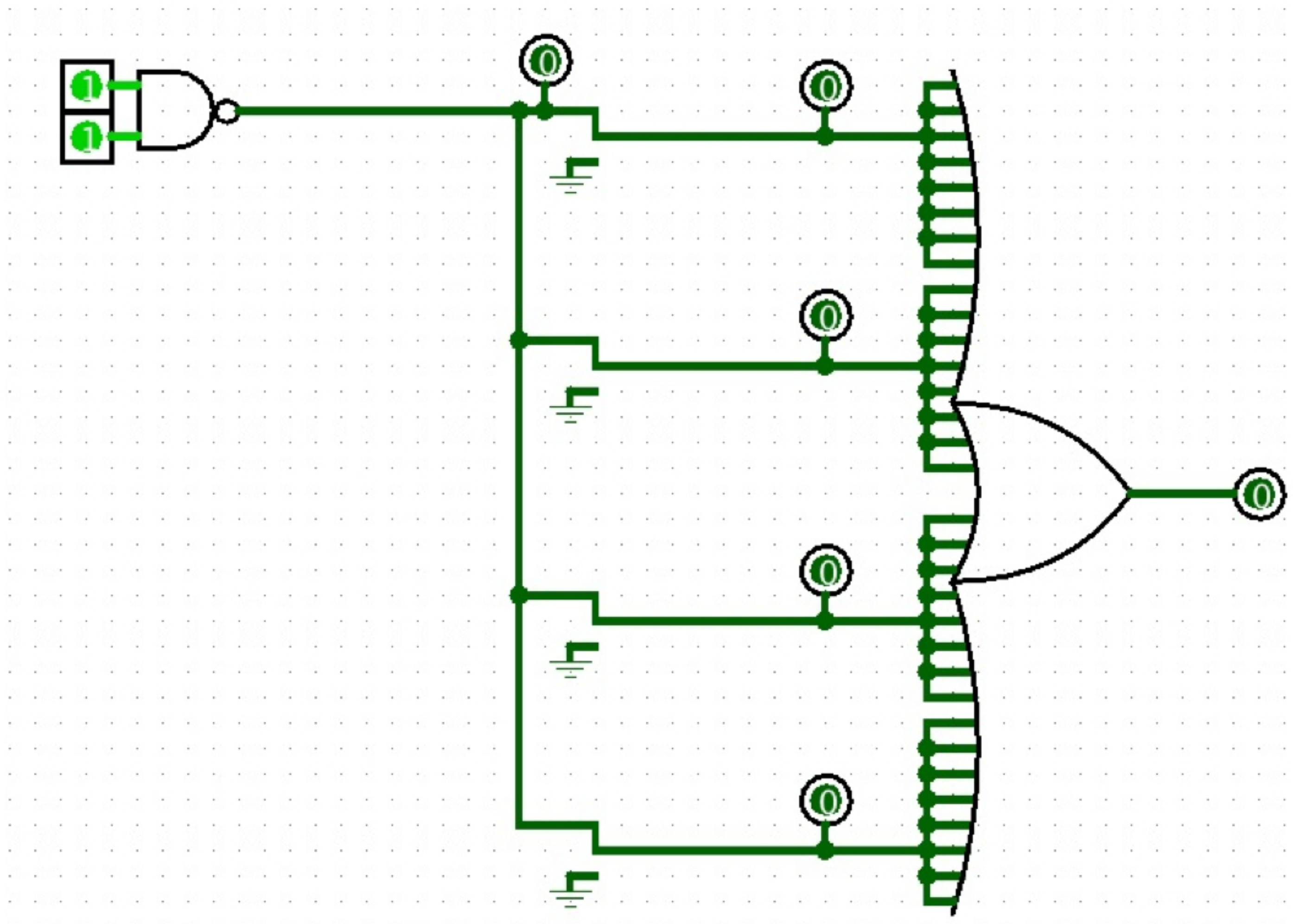
---

the physical reality of the circuits can affect the logic design. (The abstraction barrier has holes in it!)

- gates take time to respond
- wires take time to carry signals
- gates cost
- zeros can cost (due to the convention of asymmetric current)
- one output can't drive too many inputs

Recall we had a 32-input OR gate... The inputs were grouped into blocks of 8 bits. For each block, we could select whether it should be wired to the output of a NAND, or to zero. If the NAND is set up to spit out a zero, then (in theory) it should not matter how we set these switches.





But in practice, that's not what happened!

---

and (for cultural edification) we talked about how things get built

- discrete transistors, or small ICs, or large ICs (including FPGAs or ASICs)
- on breadboards or wirewrapping or printed circuit boards
- probably via a "program" in an HDL, which gets compiled (and optimized) down to a circuit layout

---

---

1. Binary

(See Sections 2.1.1, 2.2.2-2.2.4, 2.3.1-2.3.4 in the textbook)

Base 10

$$\text{“}d_3d_2d_1d_0\text{”} = d_3 \cdot 10^3 + d_2 \cdot 10^2 + d_1 \cdot 10^1 + d_0 \cdot 10^0 \qquad 0 \le d_i < 10$$

---

generalizing to other bases, such as binary

$$\text{“}d_3d_2d_1d_0\text{”} = d_3 \cdot 2^3 + d_2 \cdot 2^2 + d_1 \cdot 2^1 + d_0 \cdot 2^0 \qquad 0 \le d_i < 2$$

---

powers of 2. (memorize these!)

$$\begin{aligned}
2^0 &= 1 \\
2^1 &= 2 \\
2^2 &= 4 \\
2^3 &= 8 \\
2^4 &= 16 \\
2^5 &= 32 \\
2^6 &= 64 \\
2^7 &= 128 \\
2^8 &= 256 \\
2^9 &= 512 \\
2^{10} &= 1024
\end{aligned}$$

From the Risks Digest:

```
Date: Thu, 19 Mar 2009 19:11:47 -0400 (EDT)
From: msb@vex.net (Mark Brader)
Subject: You have won $[2^32-1]/100, no wait, we mean nothing

It was reported recently that at an Ontario casino in December, a slot
machine flashed its lights and displayed a message to the effect that "You
have won $42.9 million" (Canadian, about $34 million US).  The gambler, Paul
Kusznirewicz, had 5 minutes to be ecstatic before being told the machine had
malfunctioned and he hadn't won anything.  (They did give him some dinner
coupons.)  In fact, according to the Ontario Lottery and Gaming Corp., its
highest possible payout was $9,025 (Canadian).  This amount was not marked
on the machine, but there was a notice that nothing was payable in case of
malfunction.  Kusznirewicz is suing, so there probably won't be any further
```

```
details unless the case makes it to court.

In a followup story today, Ryerson University computer professor Sophie
Quigley suggests that the number -1, as a 32-bit 2's complement signed
integer, was interpreted as an unsigned integer in cents: $42,949,672.95.
"A casting error", as she put it.  (Not necessarily in the strict C sense.)

Incidentally, the Toronto Star ran the followup next to a story about Marie
Douglas-David, who is involved in a divorce case and allegedly claims that
"she cannot live on $43 million" (US).  The paper put the two pieces side by
side under a common headline: "Two very different $43 million questions".

http://www.cbc.ca/consumer/story/2009/03/17/slot.html
http://www.thestar.com/News/Ontario/article/604035-
```

Exercise: convert binary 1111 to decimal.

(and BCD)

---

hexadecimal

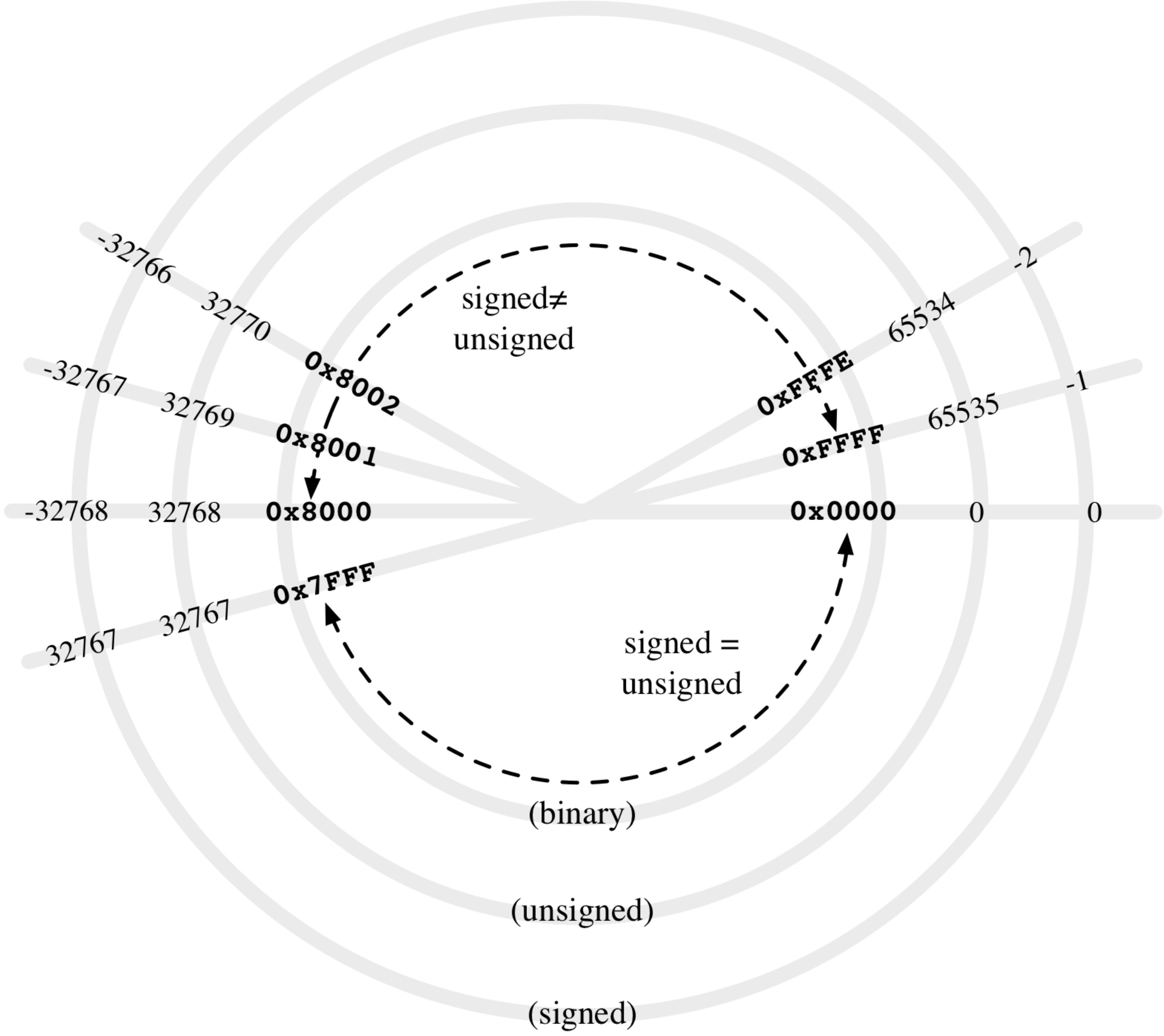
$$\text{“}d_3 d_2 d_1 d_0\text{”} = d_3 \cdot 16^3 + d_2 \cdot 16^2 + d_1 \cdot 16^1 + d_0 \cdot 16^0 \qquad 0 \leq d_i < 16$$

usually, we talk about binary numbers by breaking them into 4 bit chunks and thinking about each chunk as a hex digit.

2's complement, for signed integers

- modular arithmetic. (like an odometer)
- intuition: so it does the right thing

Here's a figure I drew for the S&M book (for COSC55)



(But note the inadvertent ambiguity: what about the edge case?)

But there are also other formats: 1's comp, floating point, etc

---

[hex2.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/04-binary/demo/hex2.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/04-binary/demo/hex2.zip)

---

2. Comparing two binary numbers

---

comparator: are two 4-bit numbers equal?

[compare.circ (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/04-binary/demo/compare.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/04-binary/demo/compare.zip) shows a starting point

Walk through LogiSim tricks...

- "width" of wires
- splitter
- multi-line "pins"
- clicking on a wire
- multi-line probes

Then build up a comparator, from basic gates!

The idea of **combinatorial logic**

---

3. Doing things with a binary number

---

decoder: turn on output line k if the binary number coming in was k

[decoder-partial.circc (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/04-binary/demo/decoder-partial.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/04-binary/demo/decoder-partial.zip)

note: general way to detect a particular binary number

note:... drafting (and some simulation) uses gates with inverters attached to the inputs

4. Generalizing

how about an arbitrary binary function?

If you've read this far

[here is a zipfile (https://ssl.cs.dartmouth.edu/~sws/cs51-s15/04-binary/demo/compare-decode-full.zip)](https://ssl.cs.dartmouth.edu/~sws/cs51-s15/04-binary/demo/compare-decode-full.zip) with more complete and annotated version of these demos