UNIVERSITY of VIRGINIA

Floating Point Numbers (From the last class)

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Floating Point Example

101.011_2

1 bit : sign

4 bits: exponent

3 bits: fraction.

① Scientific:

1.01011×2^2

② Calculate biased number for 2

a. 2's complement for 2 : 0010

b. plus the bias: 0010

+0111

1001

01011 → round up

0	1001	011
Sign	exponent	fraction

Floating Point Example

$$101.011_2$$

How do you know what comes after the binary point?

① Positionally: ↳ 3 positions, in total $2^3 = 8$ values

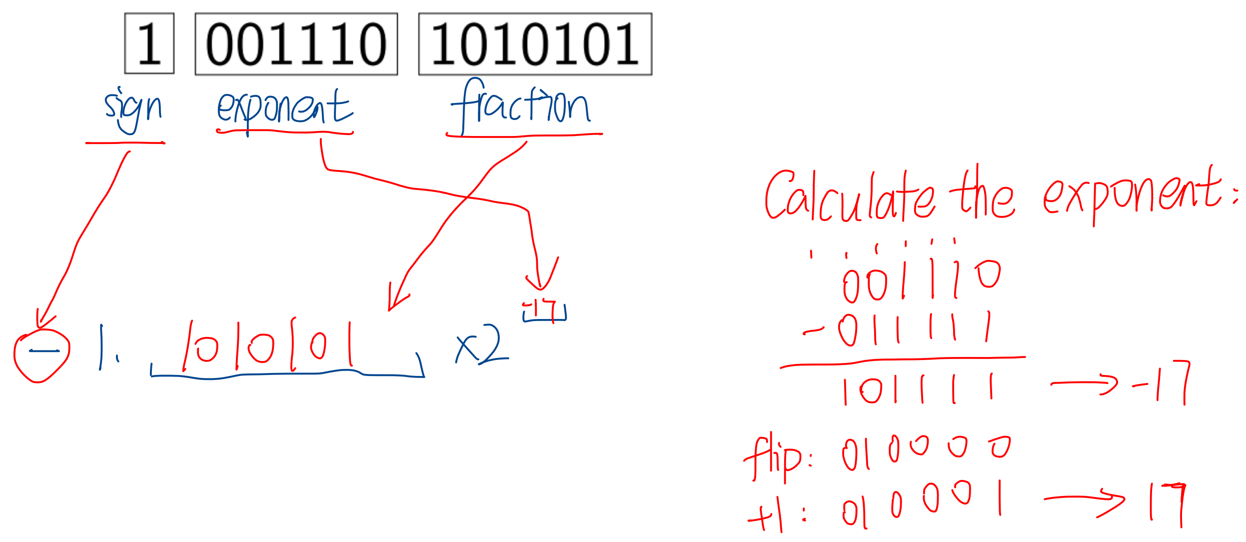
011 is 3, so 3/8

② In fractions: $\overset{2^2}{1}\ \overset{2^1}{0}\ \overset{2^0}{1}\ .\ \overset{2^{-1}}{0}\ \overset{2^{-2}}{1}\ \overset{2^{-3}}{1}$

↳ $0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} = 0 + \frac{1}{4} + \frac{1}{8} = \frac{3}{8}$

Floating Point Example

What does the following encode?



What about 0?

Floating Point Numbers

Four cases:

- **Normalized**: What we have seen today

$$s\ eeee\ ffff = \pm 1.ffff \times 2^{eeee-\text{bias}}$$

- **Denormalized**: Exponent bits all 0

$$s\ eeee\ ffff = \pm 0.ffff \times 2^{1-\text{bias}}$$

- **Infinity**: Exponent bits all 1, fraction bits all 0 (i.e., $\pm\infty$)
- **Not a Number (NaN)**: Exponent bits all 1, fraction bits not all 0

More bits, circuits, adders

CS 2130: Computer Systems and Organization 1

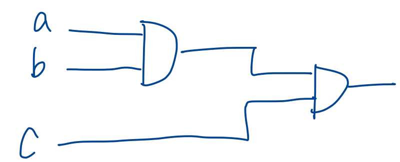
Xinyao Yi Ph.D.
Assistant Professor

Announcements

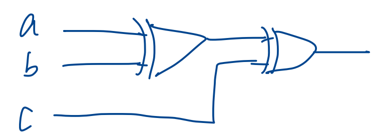
- Homework 1 due September 15

Warm up!

Can I make an n-input AND from 2-input AND gates?



What about XOR gates?



parity:

even number of bits that are one ⇒ 0 (example: 001 → 1)

odd number of bits that are one ⇒ 1 (110 → 0)

Operations

So far, we have discussed:

- Addition: $x + y$
  - Can get multiplication
- Subtraction: $x - y$
  - Can get division, but more difficult
- Unary minus (negative): $-x$
  - Flip the bits and add 1

Operations (on Integers)

Bit vector: fixed-length sequence of bits (ex: bits in an integer)

- Manipulated by bitwise operations

Bitwise operations: operate over the bits in a bit vector

- Bitwise not: ~x - flips all bits (unary)
- Bitwise and: x & y - set bit to 1 if x, y have 1 in same bit
- Bitwise or: x | y - set bit to 1 if either x or y have 1
- Bitwise xor: x ^ y - set bit to 1 if x, y bit differs

Operations (on Integers)

Logical not: $!x$

- $!0 = 1$ and $!x = 0, \forall x \neq 0$
- Useful in C, no booleans
- Some languages name this one differently

Operations (on Integers)

Left shift: $x << y$ - move bits to the left

- Effectively multiply by powers of 2

Right shift: $x >> y$ - move bits to the right

- Effectively divide by powers of 2
- Signed (extend sign bit) vs unsigned (extend 0)

Floating Point Numbers

Four cases:

- **Normalized**: What we have seen today

$$s\ eeee\ ffff = \pm 1.ffff \times 2^{eeee-\text{bias}}$$

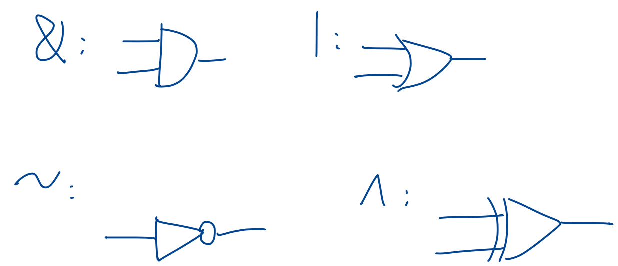
- **Denormalized**: Exponent bits all 0

$$s\ eeee\ ffff = \pm 0.ffff \times 2^{1-\text{bias}}$$

- **Infinity**: Exponent bits all 1, fraction bits all 0 (i.e., $\pm\infty$)
- **Not a Number (NaN)**: Exponent bits all 1, fraction bits not all 0

Our Story So Far

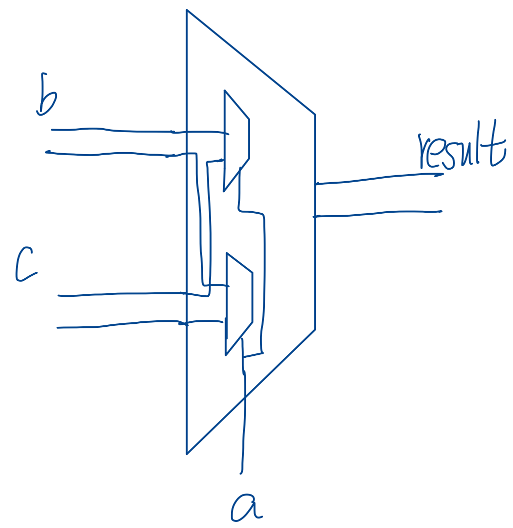
- Transistors
- Information modeled by voltage through wires (1 vs 0)
- Gates: & | ~ ^
- Multi-bit values: representing integers
  - Signed and unsigned
  - Bitwise operators on bit vectors
- Floating point



How to do the work of multi-bit?

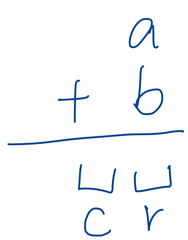
Multi-bit Mux

Our first multi-bit example: mux

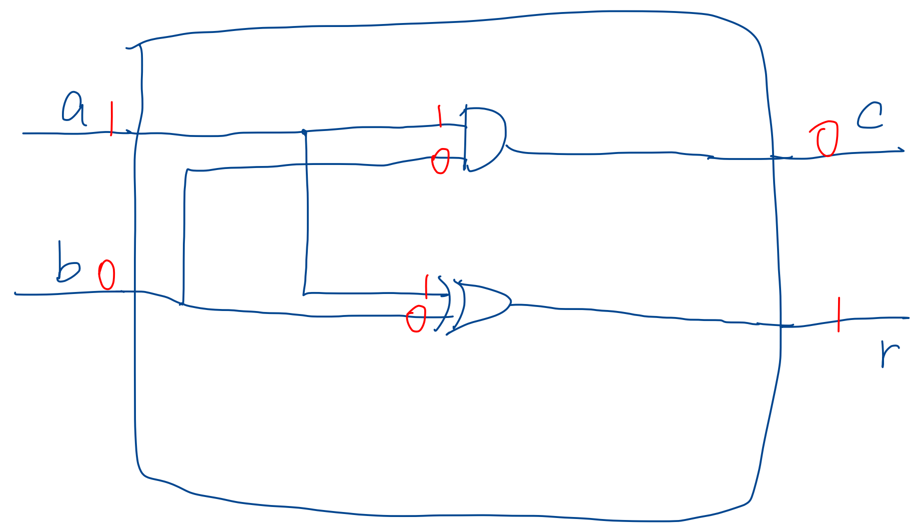


Adder

Add 2 1-bit numbers: a, b



a	b	c	r
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Adder

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

Adder

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

What is missing?

Consider:

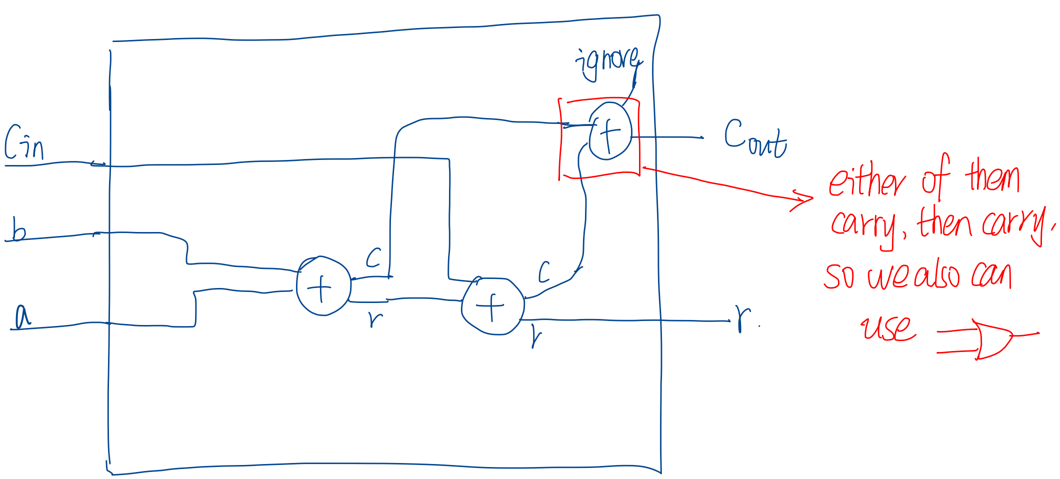
```
  1 1
  11
+01
----
 100
```

Since I have a carry-in

2 inputs ⇒ 3 input adder

3-input Adder

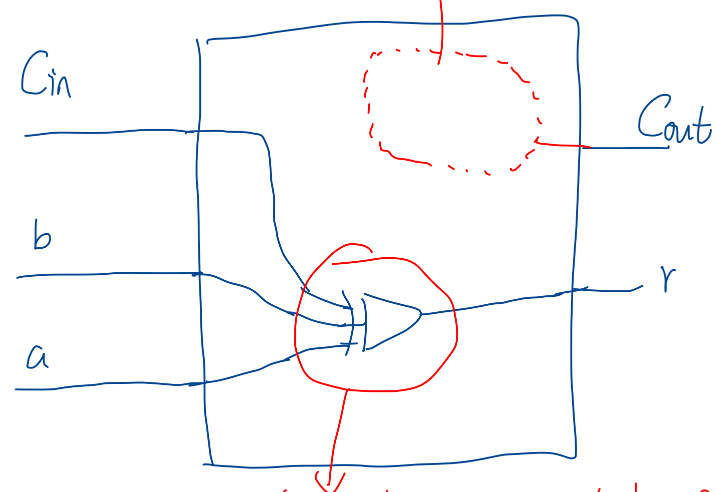
Add 3 1-bit numbers: *a*, *b*, *c*



for this part. you can think:

1 if and only if ≥2,

Think about using and/or ?



2 ones ⇒ even ⇒ lowest bit is going to be 0

1 one and 2 zeros ⇒ lowest bit : 1