Floating Point Numbers (From the last class)

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

- Homework 1 due Monday

Floating Point Example

1 bit: sign
4 bits: exponent
3 bits: fraction

$$101.011_2$$

1.01011×2^2

2's complement for 2: 0010

add the bias: 0010
+0111
= 1001

0 (sign) | 1001 (exponent) | 01011 (fraction)

only 3 bits for fraction?
We are rounding
01011 → 011

011 (fraction)

UNIVERSITY of VIRGINIA

Floating Point Example

101.011_2

$\underbrace{101}_{5}.011$

couple of ways to think about what comes after the binary point.

①. in fractions

1 0 1 . 0 1 1

$2^2\ 2^1\ 2^0\ \ 2^{-1}\ 2^{-2}\ 2^{-3}$

$4+1=5$

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

so: $101.011 = 5\frac{3}{8}$

②. positionally

3 positions : up to $2^3=8$ values (positions)

011 is 3, so 3/8 (three-eight)

Floating Point Example

What does the following encode? 14 bits.

1	001110	1010101
sign	exponent	fraction

copy this down

$-1.1010101 \times 2^{-17}$

```
  001110   biased number
- 011111   (minus bias)
  101111   2's complement
```

flip: 010000

+1: 010001 ⇒ 17

so, answer: −17

What about 0?

all the numbers start with 1, no zero!

IEEE 754: +0: sign: 0
exp: 0
fraction: 0

-0: sign: 1
exp: 0
fraction: 0

1/+0, 1/-0 } may different.

UNIVERSITY of VIRGINIA

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 Used for small numbers close to 0

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

fraction doesn't have "1. ", instead, it's "0.xx"
fix the issue of underflow. (The result is closer to zero than the smallest normalized number).

- **Infinity**: Exponent bits all 1, fraction bits all 0 (i.e., $\pm\infty$) depend on the sign bit. $1/0 \to +\infty$
- **Not a Number (NaN)**: Exponent bits all 1, fraction bits not all 0

Use it when do no meaning calculations in math.
$0/0$, sqrt(−1), $\infty - \infty$ ......

NaN ≠ any number
NaN ≠ itself.

More bits, circuits, adders

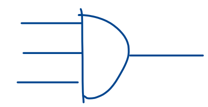
CS 2130: Computer Systems and Organization 1

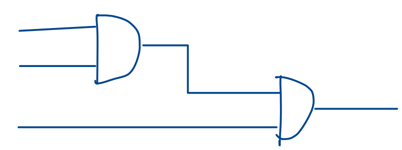
Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Warm up!

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



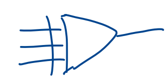


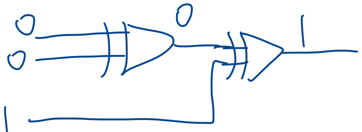
What about XOR gates?

parity → reading

0 - even number of bits that are one 110 → 0.

1 - odd number of bits that are one. 001 → 1





useful in error checking:

send: 101 $1 \wedge 0 \wedge 1 = 0$

receive: 100 $1 \wedge 0 \wedge 0 = 1$

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

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

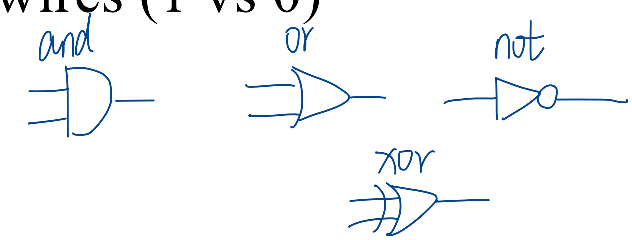
- **Denormalized**: Exponent bits all 0

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

- **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

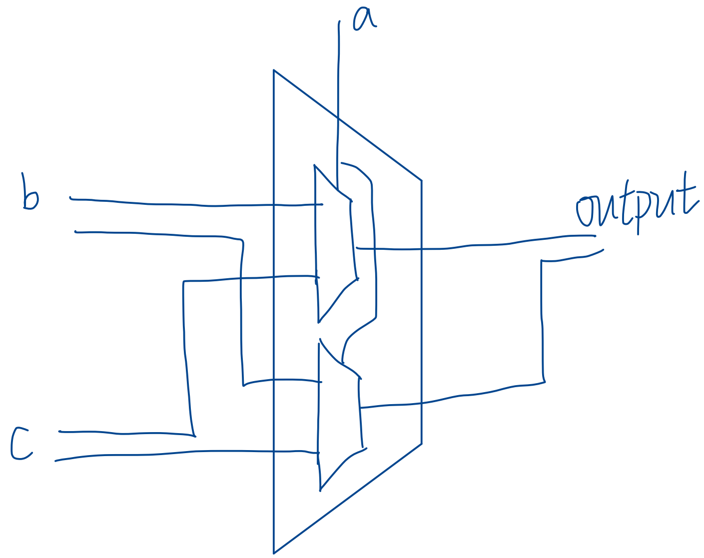
- Transistors
- Information modeled by voltage through wires (1 vs 0)
- Gates: & | ~ ^
- Multi-bit values: representing integers
  - Signed and unsigned
  - Bitwise operators on bit vectors
- Floating point



How to do the work of multi-bit?

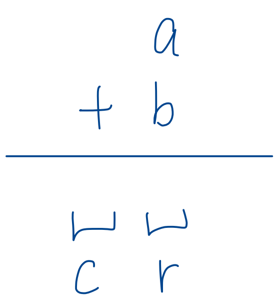
Multi-bit Mux

Our first multi-bit example: mux

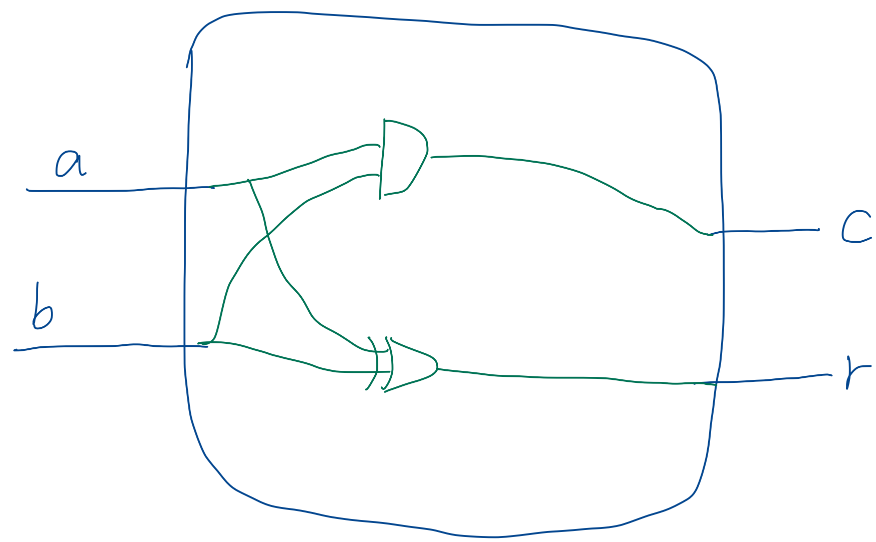


Adder

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



Is that a one bit value?

How big could that value be?

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-inputs adder.

3-input Adder

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