Binary Arithmetic

CS 2130: Computer Systems and Organization 1

September 2, 2022

Hi!

Announcements

- Quiz 0 due tonight at 5pm (when Quiz 1 opens)
- Quiz 1 opens at 5pm (due Monday at 8am)
- Lab 1 late check-off through Monday
- TA office hours start tonight!
  - **In-person**: Olsson 001, Wed-Sun, 5-7pm
  - **Online**: Discord, Wed-Sun, varies
  - Discord is now available
- My office hours
  - Tuesday, 4-5pm, Discord/Zoom
  - Wednesday, 4:30-6pm, Rice 210 (masks requested)
  - Thursday, 11am-12pm, Discord/Zoom

Negative Integers

Representing negative integers

- Can we use the minus sign?
- In binary we only have 2 symbols, must do something else!
- Almost all hardware uses the following observation:

Negative Integers

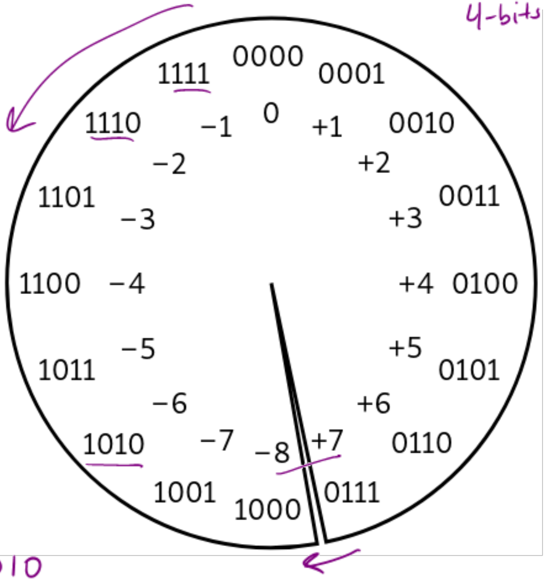
Representing negative integers

- Computers store numbers in fixed number of wires
- Ex: consider 4-digit decimal numbers
- Throw away the last borrow:
  - 0000 - 0001 = 9999
  - 9999 - 0001 = 9998
  - Normal subtraction/addition still works
- This works the same in binary

Two's Complement

This scheme is called **Two's Complement**

- More generically, a *signed* integer
- There is a break as far away from 0 as possible
- First bit acts vaguely like a minus sign
- Works as long as we do not pass number too large to represent



8-bit: 0000 1010

Questions?

Two’s Complement

Values of Two's Complement Numbers

Consider the following 8-bit binary number in Two's Complement:

11010011

-45

What is its value in decimal?

Values of Two’s Complement Numbers

Consider the following 8-bit binary number in Two’s Complement:

11010011

What is its value in decimal?

$$\begin{array}{r} 00101100 \\ +\quad 1 \\ \hline 00101101 \end{array} = 45$$

32 16 8 4 2 1

1. Flip all bits
2. Add 1

4 bits

0000 flip

1111 +1

$$\begin{array}{r} 1111 \\ +\quad 1 \\ \hline 0000 \end{array}$$

What about other kinds of numbers?

Non-Integer Numbers

Floating point numbers

- Decimal: 3.14159

  decimal point

Non-Integer Numbers

Floating point numbers

10^1 10^0 10^{-1} 10^{-2}

- Decimal: 3.14159
- Binary: 11.10110

2^1 2^0 2^{-1} 2^{-2}

Non-Integer Numbers

Floating point numbers

- Decimal: 3.14159
- Binary: 11.10110
- With integers, the point is always fixed after all digits
- With floating point numbers, the point can move!

Non-Integer Numbers

Floating point numbers

- Decimal: 3.14159
- Binary: 11.10110
- With integers, the point is always fixed after all digits
- With floating point numbers, the point can move!

Challenge! only 2 symbols in binary

Scientific Notation

Convert the following decimal to scientific notation:

$$2130.$$

$$2.13 \times 10^3$$

Scientific Notation

Convert the following binary to scientific notation:

$1.01101 = 1.0110100000000$

101101.

1.01101×2^5

Something to Notice

An interesting phenomenon:

- Decimal: first digit can be any number *except* 0

$$2.13 \times 10^3$$

Something to Notice

An interesting phenomenon:

- Decimal: first digit can be any number *except* 0

$$2.13 \times 10^3$$

- Binary: first digit can be any number *except* 0 Wait!

$$1.01101 \times 2^5$$

$$\cancel{0.101101 \times 2^6}$$

Something to Notice

An interesting phenomenon:

- Decimal: first digit can be any number *except* 0

$$2.13 \times 10^3$$

- Binary: first digit can be any number *except* 0 Wait!

$$1.01101 \times 2^5$$

  - First digit can only be 1

Floating Point in Binary

$\boxed{\pm}\ 1.\underbrace{01101} \times 2^{\boxed{5}}$

We must store 3 components

- **sign** (1-bit): 1 if negative, 0 if positive
- **fraction** or **mantissa**: (?-bits): bits after binary point
- **exponent** (?-bits): how far to move binary point

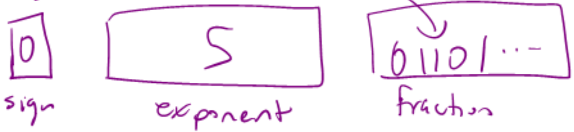
We do not need to store the value before the binary point. Why?

Floating Point in Binary

How do we store them?

- Originally many different systems
- IEEE standardized system (IEEE 754 and IEEE 854)
- Agreed-upon order, format, and number of bits for each

$$1.01101 \times 2^5$$



Example

A rough example in Decimal:

$$\frac{42}{100}$$

$$6.\underline{42} \times 10^3$$

8 digits

1 - sign
2 - exp
4 - mantissa

0042 ?

0 03 .4200

$$\frac{.4200}{10000}$$

Exponent

How do we store the exponent?

- Exponents *can* be negative

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

- Need positive and negative ints (but no minus sign)

Exponent

How do we store the exponent?

- Exponents *can* be negative

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

- Need positive and negative ints (but no minus sign)
- *Don't we always use Two's Complement?*

Exponent

How do we store the exponent?

- Exponents *can* be negative

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

- Need positive and negative ints (but no minus sign)
- *Don't we always use Two's Complement?* Unfortunately Not

Exponent

How do we store the exponent?

- Exponents *can* be negative

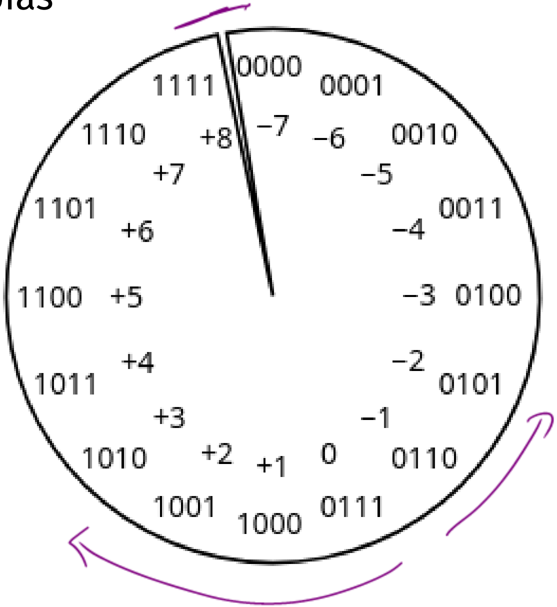
$$2^{-3} = \frac{1}{2^3} = \frac{1}{8}$$

- Need positive and negative ints (but no minus sign)
- *Don't we always use Two's Complement?* Unfortunately Not
- Biased integers
  - Make comparison operations run more smoothly
  - Hardware more efficient to build
  - Other valid reasons

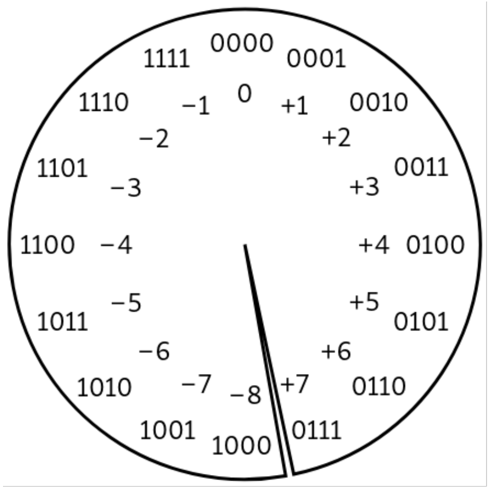
Biased Integers

Similar to Two's Complement, but add **bias**

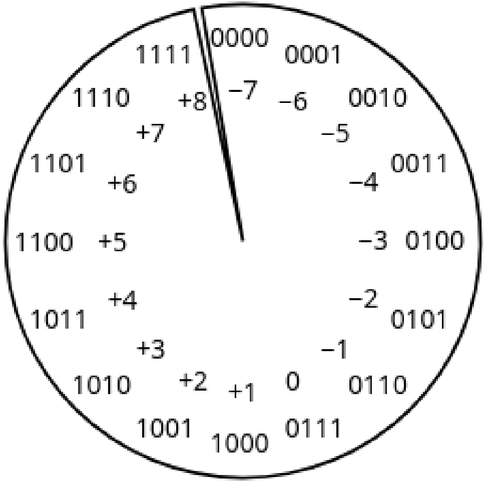
- **Two's Complement**: Define 0 as 00...0
- **Biased**: Define 0 as 0111...1
- Biased wraps from 000...0 to 111...1



Biased Integers



Two’s Complement



Biased

Biased Integers Example

Calculate value of biased integers (4-bit example)

```
  0010
- 0111
------
  1011
```

```
  0010      2
- 0111    - 7     1) subtract bias
------
  1011            2) 2's complement
  0100
+    1
------
  0101    = -5
```

Biased Integers

Floating Point Example

$$101.011_2$$

Floating Point Example

$$101.011_2$$

Floating Point Example

What does the following encode?

$\boxed{1}\ \boxed{001110}\ \boxed{1010101}$

Floating Point Example

What does the following encode?

1	001110	1010101

What about 0?

Floating Point Numbers

Four cases:

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

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

- **Denormalized**: Exponent bits all 0

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

- **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