Binary Arithmetic

CS 2130: Computer Systems and Organization 1

September 2, 2022

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

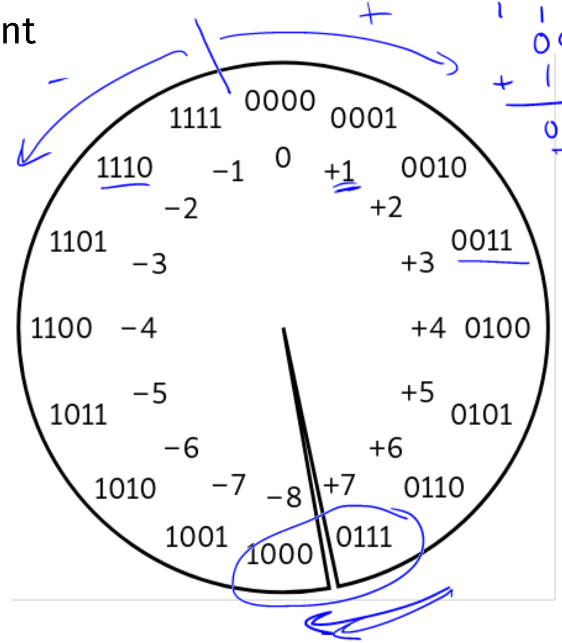
- Computers store numbers in fixed number of wires
- Ex: consider 4-digit decimal numbers
- Throw away the last borrow:
  - 0000 - 0001 = 9999
  - 9999 - 0001 = 9998
  - Normal subtraction/addition still works
- This works the same in binary

2 2 2 2
10 10 10 10
1 1 1 1
0000 0
− 001
1111 −1
− 1
1110 −2

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



$$\begin{array}{r} 0011 \\ +\ 1110 \\ \hline 0001 \end{array}$$

8-bit 0000 1011

Questions?

Two's Complement

4-bit

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

$\cdots 000101$

101

8-bit

$$
\begin{array}{r}
0000\ 0000 \\
- \qquad\qquad 1 \\
\hline
1111\ 1111 \\
- \qquad\qquad 1 \\
\hline
1111\ 1110
\end{array}
$$

Values of Two’s Complement Numbers

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

11010011

What is its value in decimal?

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?

1. Flip all bits
2. Add 1

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

128 64 32 16 8 4 2 1

$$
\begin{array}{rr l}
 & 00000000 & \text{flip} \\
 & \scriptstyle 1\,1\,1\,1\,1\,1\,1 & \\
1 & 11111111 & \text{add 1} \\
+ & 1 & \\ \hline
 & 00000000 &
\end{array}
$$

What about other kinds of numbers?

Non-Integer Numbers

Floating point numbers

- Decimal: 3.14159

decimal point

Non-Integer Numbers

Floating point numbers

- Decimal: 3.14159
- Binary: 11.10110

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

2.13×10^3

Scientific Notation

Convert the following binary to scientific notation:

$$101101$$

$$1.01101 \times 2^5$$

Something to Notice

An interesting phenomenon:

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

$$\underline{2}.13 \times 10^3 \qquad \cancel{0.213 \times 10^4}$$

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!

$$1.01101 \times 2^5$$

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!

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

  - First digit can only be 1

Floating Point in Binary

$1.\underbrace{01101}_{\text{fraction}} \times 2^{5}$

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

$$1.01101 \times 2^5$$

$$\underset{s}{\boxed{0}} \quad \underset{exp}{\boxed{\quad 5 \quad}} \quad \underset{fraction}{\boxed{01101000000}}$$

Example

A rough example in Decimal:

$\frac{42}{100}$

?

$$6.42 \times 10^3$$

1 sign

2 exp

4 mantissa

00 42

$\boxed{0}\ \boxed{3}\ \boxed{4200}$

$\frac{4200}{10000}$

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

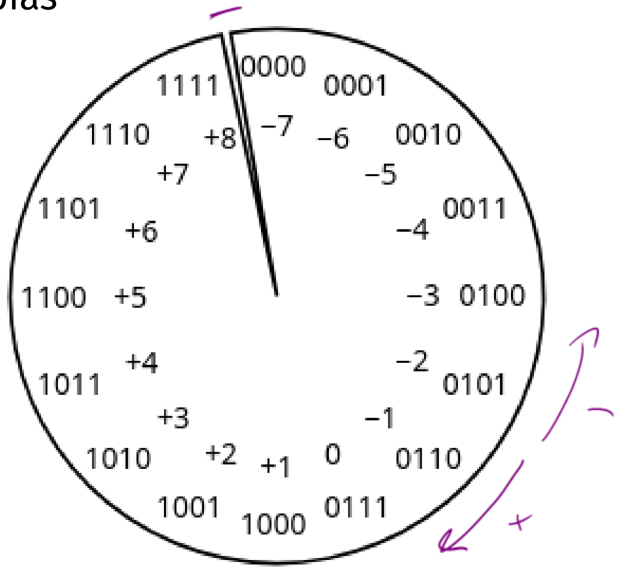
$$2^{-3} = \frac{1}{2^3} = \frac{1}{8}$$

- 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

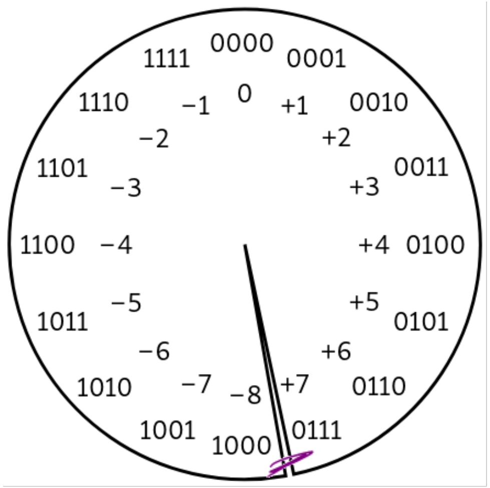
Biased Integers

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

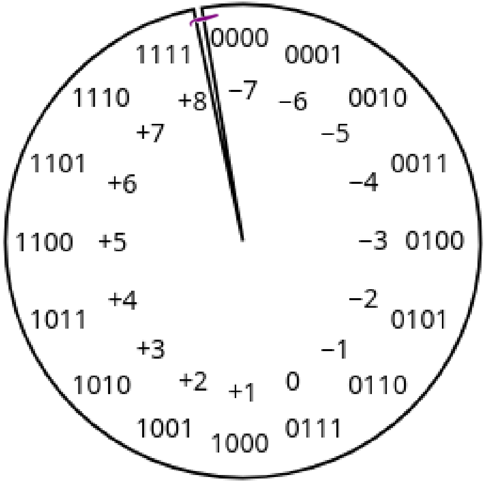
- **Two's Complement**: Define 0 as 00…0
- **Biased**: Define 0 as 0111…1
- Biased wraps from 000…0 to 111…1



Biased Integers



Two's Complement



Biased

Biased Integers Example

Calculate value of biased integers (4-bit example)

0000 $\xrightarrow{\text{bias}}$ 0111

$$\begin{array}{r} \overset{4}{\cancel{8}}\,\overset{12}{2} \\ -\ 7 \\ \hline \end{array}$$

$$\begin{array}{r} \overset{1}{0}\overset{1}{0}\overset{1}{1}\overset{1}{0} \\ -\ 0111 \\ \hline 1011 \end{array} \qquad 2 - 7 = -5$$

$$\begin{array}{r} 0100 \\ +\ \ \ \ 1 \\ \hline 0101 \end{array} = 5 \quad \rightarrow \quad -5$$

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?

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

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