Floating Point Numbers

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

Announcements

- Homework 1 due September 15
- Lab 2 tomorrow!

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)

Non-Integer Numbers

- 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

binary point

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

2.130×10^3

Scientific Notation

Convert the following binary to scientific notation:

101101

1.01101×2^5

UNIVERSITY of VIRGINIA

Something to Notice

An interesting phenomenon:

- Decimal: first digit can be any number except 0

$$2.13 \times 10^3$$

0.213×10^4

Something to Notice

An interesting phenomenon:

- Decimal: first digit can be any number except 0

  2.13×10^3

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

  1.01101×2^5

Something to Notice

An interesting phenomenon:

- Decimal: first digit can be any number except 0

  2.13×10^3

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

  1.01101×2^5

  – First digit can only be 1

Floating Point in Binary

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

depends on the hardware design.

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

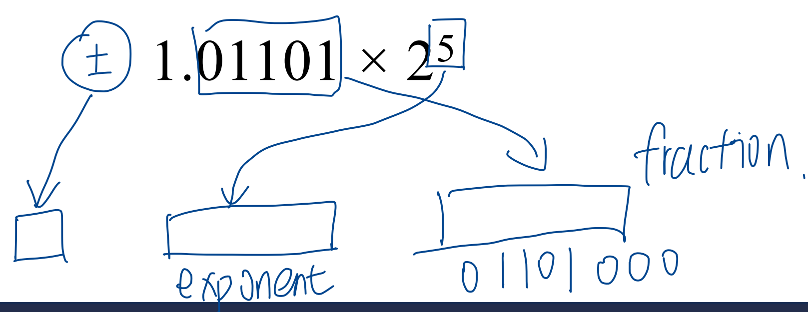
alway 1

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

$$\pm\ 1.01101 \times 2^{5}$$



Example

A rough example in Decimal:

$$6.42 \times 10^3$$

0 | 03 | 420
sign | exp | fraction

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

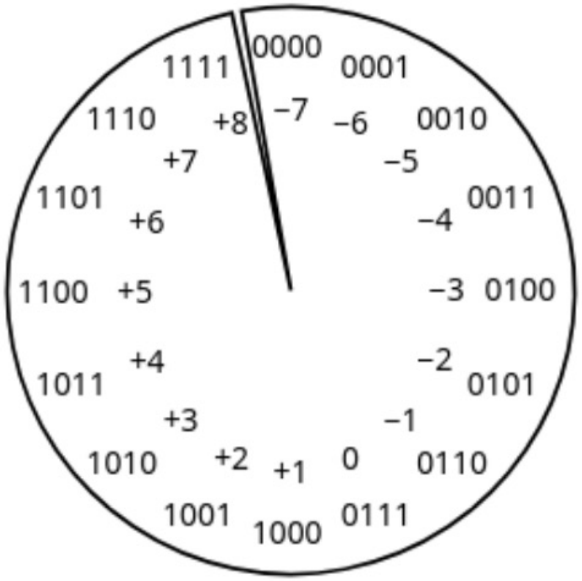
How do we store the exponent?

- Biased integers
  – Make comparison operations run more smoothly
  – Hardware more efficient to build
  – Other valid reasons

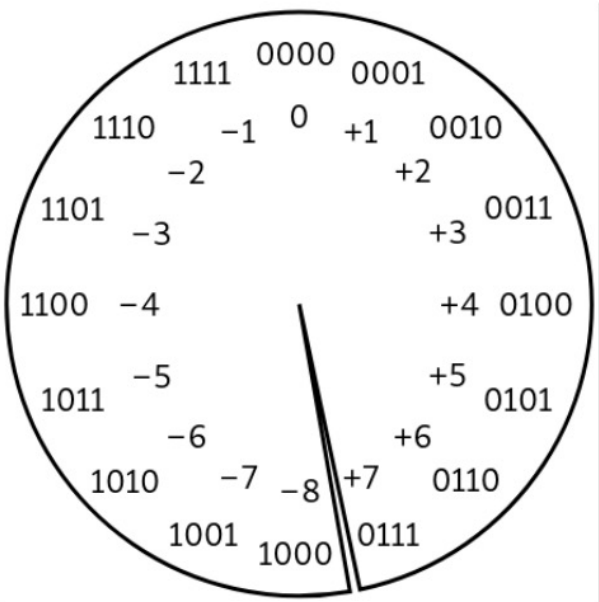
Biased Integers

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

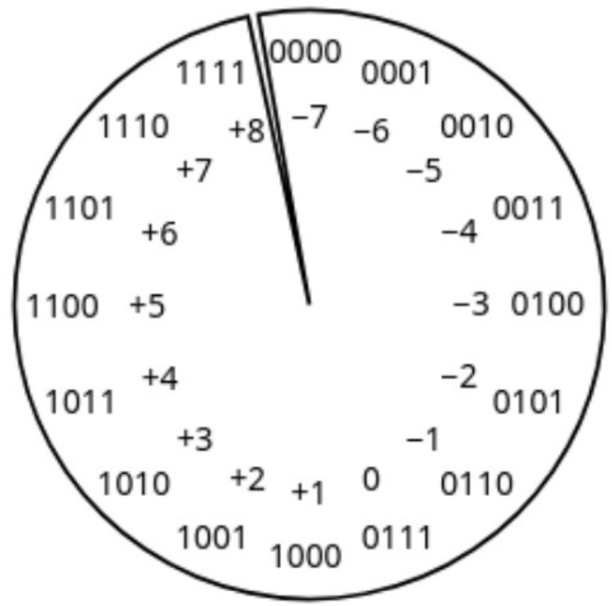
- **Two's Complement**: Define 0 as 00...0
- **Biased**: Define 0 as 0111...1
- Biased wraps from 000...0 to 111...1



Biased Integers



Two's Complement



Biased

Biased Integers Example

Calculate value of biased integers (4-bit example)

0010

```
  0010
- 0111
------
  1011 -> -5
flip: 0100
+1: 0101 -> +5
```

2's complement → Biased Integers : add the bias.

Biased Integers → 2's complement : substract the bias.

Floating Point Example

1 bit : sign

4 bits : exponent

3 bits: fraction

101.011_2 —Scientific→ 1.01011×2^2

↓ round to "011"

① 2's complement for 2:

0010

② add the bias to get the biased integer:

$$\begin{array}{r} 0010 \\ +\,0111 \\ \hline 1001 \end{array}$$

0	1001	011
sign	exponent	fraction