CS 520
Floating Point Representation

Real Numbers

- Numbers with fractional parts
- Since computer memory is limited, you cannot store numbers with infinite precision
- Fractions can be expressed using different number systems but no matter which technique is used, but you can only be precise to an extent.
- But how much accuracy is needed? And *where* is it needed? How many integer digits and how many fraction digits

Scientific Notation

- The idea is to compose a number of two main parts:
  - A **significand** that contains the number's digits. Negative significands represent negative numbers.
  - An **exponent** that says where the decimal (or binary) point is placed relative to the beginning of the significand. Negative exponents represent numbers that are very small (i.e. close to zero).
- It can represent numbers at wildly different magnitudes (limited by the length of the exponent)
- It provides the same relative accuracy at all magnitudes (limited by the length of the significand)
- It allows calculations across magnitudes: multiplying a very large and a very small number preserves the accuracy of both in the result.
- Decimal floating-point numbers usually take the form of scientific notation with an explicit point always between the 1st and 2nd digits. The exponent is either written explicitly including the base, or an **e** is used to separate it from the significand.

Decimal Scientific Notation

Significand	Exponent	Scientific notation	Fixed-point value
1.5	4	$1.5 \cdot 10^4$	15000
-2.001	2	$-2.001 \cdot 10^2$	-200.1
5	-3	$5 \cdot 10^{-3}$	0.005
6.667	-11	6.667e-11	0.00000000006667

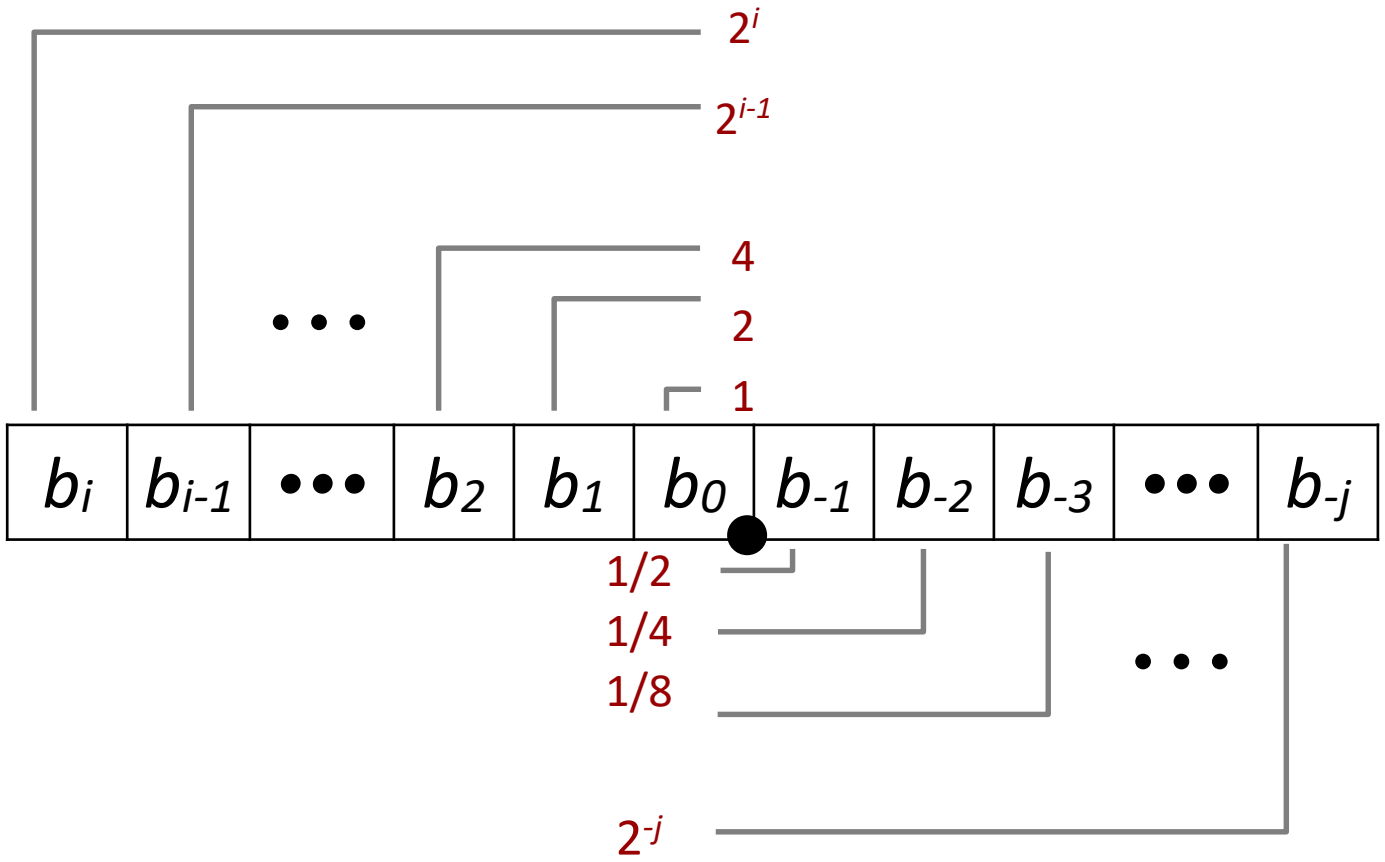
By increasing (or decreasing) the number of digits in the significand portion the value of the exponent is decreased (or increased)

Fractional binary numbers

- Can we represent a binary number with a decimal point?
- What is 1011.101_2?
  - $(1x2^3 + 0x2^2 + 1x2^1 + 1x2^0) + (1x1/2 \text{ x } 0x1/4 + 1+1/8)$

  --> $11 + (0.5 + 0.125) = 11.625$
- There is no reason why the same scientific notation cannot be used with binary numbers

Fractional Binary Numbers



- Representation
  - Bits to right of "binary point" represent fractional powers of 2
  - Represents rational number: $\sum_{k=-j}^{i} b_k \times 2^k$

Fractional Binary Numbers

Value	Representation
5 3/4	101.11_2
2 7/8	10.111_2
1 7/16	1.0111_2

- Limitation
  - Can only exactly represent numbers of the form $x/2^k$
    - Other rational numbers have repeating bit representations
  - Value Representation
    - 1/3 $0.0101010101[01]\ldots_2$
    - 1/5 $0.001100110011[0011]\ldots_2$
    - 1/10 $0.000110011001 1[0011]\ldots_2$

Floating Point

- A floating point number is computer support for real numbers that uses binary numbers in scientific notation
- IEEE Standard 754
  - Established in 1985 as uniform standard for floating point arithmetic
    - Before that, many idiosyncratic formats
  - Supported by all major CPUs
  - Many hardware floating point units use the IEEE 754 standard.

Floating Point Representation

- Numerical Form:

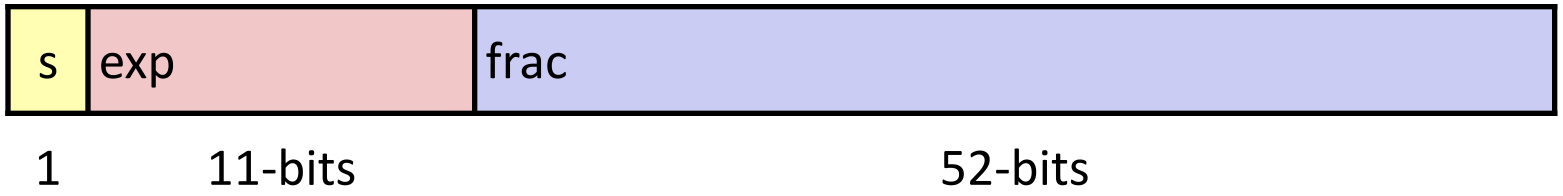
$$(-1)^s \, M \, 2^E$$

  - **Sign bit *s*** determines whether number is negative or positive
  - **Significand *M*** normally a fractional value in range [1.0,2.0).
  - **Exponent *E*** weights value by power of two
- Encoding
  - MSB s is sign bit ***s***
  - exp field encodes ***E*** (but is not equal to E)
  - frac field encodes ***M*** (but is not equal to M)

FP - Precision options

- Single precision: 32 bits

- Double precision: 64 bits



- Extended precision: 80 bits (Intel only)

IEEE FP

- IEEE floating point format has a normalized form (no matter what the value is it must be represented in this form)

- $1.XXXX....X \times 2^{exp}$
  - Don't represent this since this "one" must be present

- ***In a Normalized floating-point value there are no leading zeros in the significand***
  - ***Instead leading zeros are removed by adjusting the exponent***
  - ***0.0123 would be written as 1.23×10^{-2}***

IEEE FP

- If exponent is all zero bits
  - If significand is all zero bits value is 0.0 (+/-)
  - If significand is not all zero bits value is de-normalized (*some values very close to zero are allowed*)

    Value is $0.XXX…XXx2^{-126}$
- If exponent is all one bits
  - If significand is all zero bits value is “infinity” (+/-)

    (divide by zero, overflow)
  - If significand is not all zero bits value is “NaN”

    (zero divided by zero, square root of a negative number)

- ***De-normalized values are numbers where this representation would result in an exponent that is below the smallest representable exponent (the exponent usually having a limited range)***
  - ***Such numbers are represented using leading zeros in the significand***

IEEE FP

- All other cases exponent is a “biased” exponent
  - Actual exponent = Stored exponent minus Biased exponent
- Single precision bias is 127
- Double precision bias is 1023
- In other words:
  - Treat exponent as unsigned integer
  - Subtract bias amount from stored value to get actual exponent

Decimal to IEEE Single-Precision

- -0.75_{10}

--> (1/2 + 1/4)

--> .11 x 2^0

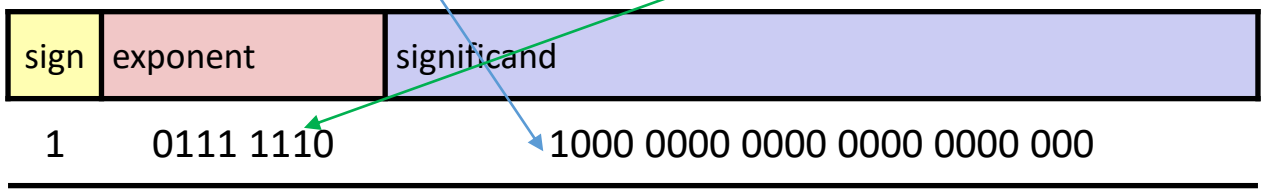
-->1.1000….0 x 2^{-1} (normalized, also fill with trailing zeroes in significand)

not stored!

Actual exponent = Stored exponent – 127

-1 = Stored exponent – 127

Stored exponent = +126 --> $0111\ 1110_2$



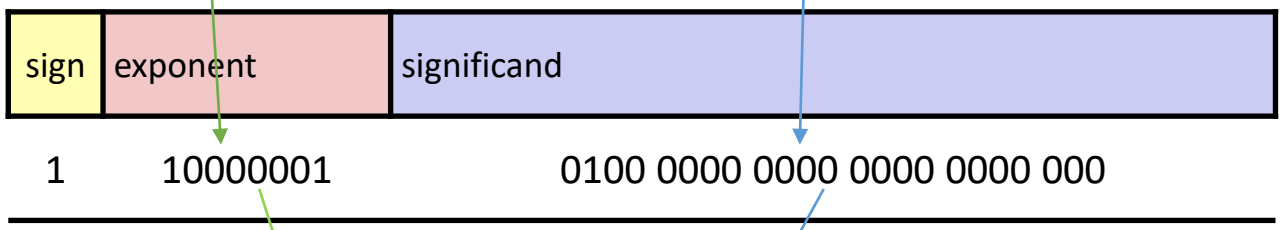
- 1011 1111 0100 0000 0000 0000 0000 0000
- B F 4 0 0 0 0 0

Little Endian	Big Endian
00	BF
00	40
40	00
BF	00

IEEE Single-Precision to Decimal

- IEEE FP Value: $C0A00000_{16}$

→1100 0000 1010 0000 0000 0000 0000 0000



Stored Value = 129

Actual Value = 129 – 127 = 2

--> $1.01000…0 \times 2^{2}$

--> $101.000…0 \times 2^{0}$

--> -5.0_{10}

Double-Precision

- -5.0_{10} to Double Precision

  --> $101.000…0 \times 2^0$

  --> $1.01000…0 \times 2^2$

  Actual exponent = Stored value – 1023

  2 = Stored value – 1023

  Stored value = 2 + 1023 = 1025 --> 100 0000 0001

  1 100 0000 0001 0100 ……..0 (..52 significand bits total)

1100 0000 0001 0100 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

C 0 1 4 0 0 0 0 0 0 0 0 0 0 0 0

(Store as Little Endian Or Big Endian)

Floating Point Overflow & Underflow

- Like with integers overflow can occur when value cannot be fit into a fixed sized container
- The component of the floating point number that actually overflows is the ***exponent***
- Underflow can occur when the value is too close to zero
  - *Exponent gets too negative*