

More Kinds of Data

CS 350, Lecture 4

A. Why?

- Floating-point numbers provide a way to separate the magnitude of a number from the number of bits of significance it has.

B. Outcomes

At the end of today, you should:

- Know why we have floating-point numbers and how they are represented.

C. Non-Whole and Floating-Point Numbers

- In decimal, non-whole numbers are represented using a decimal point and a fractional part that sums up negative powers of 10.
- In binary, non-whole numbers use a binary point and negative powers of 2.
 - E.g., $1.01101_2 = 1 + 1/4 + 1/8 + 1/32 = 1 \text{ }^{13}/_{32} = 1.40625_{10}$
- True real numbers can take an infinite number of digits.
 - We approximate them with a finite number of digits.
 - Tradeoff representing numbers with a large magnitude vs numbers with a lot of precision. **Magnitude**: Distance from zero. **Precision**: Number of significant digits. The **range** of a set of numbers is the distance between the most positive and most negative numbers.
- Floating-point numbers use separate bits for magnitude and precision
 - Based on scientific notation. E.g. $1.011_2 \times 2^{56}$ or $-1.101_2 \times 2^{12}$.
 - Notice sign of whole number doesn't have to equal sign of exponent.

D. IEEE Floating-Point Number Standard

- IEEE = Institute of Electrical & Electronics Engineers.
- Break up floating-point number into a sign bit S , an exponent, and a fraction.
 - A 32-bit IEEE floating-point number has a 1-bit sign, an 8-bit exponent, and a 23-bit fraction.
 - In typical case, fraction begins with 1.the 23 bits, and exponent is ≥ -127 and ≤ 128 .
 - E.g. $-1011010000000.0 = -1.01101_2 \times 2^{12}$.
- **Important!!: 32-bit IEEE representation drops leading "1." from 23-bit fraction and adds 127 to 8-bit exponent.**
- Example: For $-1.01101_2 \times 2^{12}$
 - Use sign bit 1 (negative number).
 - Use exponent $12+127 = 139$ ($= 1000 \ 1011_2$).
 - Drop "1." from 1.01101 to get fraction 01101, add (eighteen) 0s to get 23 bits.

- Concatenate (sign, exponent, fraction) to get the bitstring.
 - Sign = 1, exponent = 1000 1011, fraction = 01101 plus 18 0's
 - Result = 1 1000 1011 01101 00000 00000 00000 000
 - = 1100 0101 1011 0100 0000 0000 0000 0000 as the representation of $-1.01101_2 \times 2^{12}$.
 - More generally, let S be sign bit, E be 8-bit exponent.
 - Case $1 \leq E < 255$ is the standard case above:
 - The bit string represents $(-1)^S \times 1.Fraction_2 \times 2^X$ where $X = E - 127$.
 - Case $E = 0$ is used for floating-point zero and for extremely small numbers (very close to zero).
 - If $E = 0$ and $Fraction = 0$ we have $+0$ or -0 .
 - If $E = 0$ and $Fraction \neq 0$ we have $N = (-1)^S \times 0.Fraction_2 \times 2^{-126}$ (***)
 - Case $E = 255$ used for $+\infty$, $-\infty$, and NaN (Not a Number). (***)
- (***) Not critical you know this — I.e., it won't be on the exams or quizzes.