COMPSCI 1JC3
Introduction to Computational Thinking
Fall 2017

03 Numbers

William M. Farmer

Department of Computing and Software
McMaster University

September 18, 2017

Admin

- Assignment 1 has been posted; it is due Fri, Sep 29.
- M&Ms have been very interesting to read.
- Office hours: To see me please send me a note with times.
- Are there any questions?

Advice

1. Learn to manage your time and work efficiently!
   - Schedule time for your work.
   - Break up your work into small pieces.
   - Work when you are most productive.
   - Take a break from your work occasionally.
   - Keep notes and reorganize them as needed.
   - Review what you have learned.

Review

1. Muhammad Al-Khwarizmi.
2. CPUs and GPUs.
3. Algorithms.
4. Pseudocode.
5. Flowcharts.
6. Digital information.
7. Euclid's GCD algorithm.

Case Study: Ariane 5

- The Ariane 5 is a launch vehicle used by the European Space Agency.
  - Its development took 10 years and cost $7 billion (Wikipedia).
- The European Ariane 5 rocket exploded on its first test flight in 1996.
  - $500 million loss in rocket and cargo value.
- The failure was due to a software error: a 64-bit floating point number was converted to a 16-bit machine integer.
  - The module that did the conversion was written for the Ariane 4 but reused for the Ariane 5 without re-analysis.
- Shows that software developers must have a detailed understanding how numbers are represented.

Number Systems

- The idea of a number is one of the strongest and most important threads in the history of mathematics.
- The family of number systems includes:

  $\mathbb{N} = \{0, 1, 2, \ldots\}$, the natural numbers, for counting and ordering.
  $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$, the integers, for counting forwards and backwards.
  $\mathbb{Q}$, the rational numbers, for measuring.
  $\mathbb{R}$, the real numbers, for solving geometric problems.
  $\mathbb{C}$, the complex numbers, for solving algebraic problems.
  $\mathbb{Z}_n$, the modular integers, for integer arithmetic modulo n where $n \geq 1$ (clock arithmetic).
- These number systems are closely related to each other.
  - $\mathbb{Z}_n \subseteq \mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R} \subseteq \mathbb{C}$.
  - Addition and multiplication is defined in each system.

Decimal versus Binary

- Base 10: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Base 2: 0, 1
- Position of place value specifies magnitude.
- $(86409)_{10} = 9 \times 1 + 0 \times 10 + 4 \times 100 + 6 \times 1000 + 8 \times 10000$
- $(86409)_{10} = 9 \times 10^0 + 0 \times 10^1 + 4 \times 10^2 + 6 \times 10^3 + 8 \times 10^4$
- $(10101101)_2 = 1 \times 1 + 0 \times 2 + 1 \times 4 + 1 \times 8 + 0 \times 16 + 1 \times 32 + 0 \times 64 + 1 \times 128 = 173$
- $(10101101)_2 = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 + 0 \times 2^6 + 1 \times 2^7 = 173$

Binary to Decimal (iClicker)

What is the binary number 1011 equal to in decimal?

A. 3.
B. 6.
C. 7.
D. 9.
E. $\boxed{11.}$

Representation of Numbers in Different Bases

$$(a_n a_{n-1} \ldots a_1 a_0)_b = \sum_{k=0}^{n} a_k b^k$$

- The common bases found in history: 10, 12, 20, 60.
- The common bases used in computing: 2, 10, 16.
- For base 16, more symbols are needed — we use alpha characters:

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Hexadecimal

Binary	Decimal	Hexadecimal	Binary	Decimal	Hexadecimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

Binary to Hexadecimal (iClicker)

What is the binary number 11101011 equal to in hexadecimal?

A. A3.

B. B6.

C. C7.

D. D9.

E. $\boxed{\text{EB.}}$

Problem: How to Represent Number Systems

- The first, and still most important, computer application is the processing of number-based computations.
- Problem: How can an infinite number system be represented on a computer?
- Solution 1: Represent each member of the system using a fixed number of bits.
  - Advantage: Efficiency in the use of space and time.
  - Disadvantage: Only finitely many members can be represented.
- Solution 2: Represent each member of the system using an unbounded number of bits.
  - Advantage: All members can be represented.
  - Disadvantage: Inefficiency in the use of space and time.

Machine Integers

- Integers are represented with a fixed number of bits.
- Signed magnitude approach.
  - First bit is 0 for positive, 1 for negative.
  - Has the problem of two zeros.
  - Most computers actually use 2's complement.
- Two's complement using 2^n bits.
  - Has one 0; $2^{n-1} - 1$ positives; 2^{n-1} negatives.
  - Positive numbers look as expected; negative numbers have 1 as the most significant bit.
  - Negate a number by inverting its bits and adding 1.
  - x + (-x) = 0.
  - Addition and multiplication are performed using modular arithmetic.
- Arithmetic operations like addition and multiplication on machine integers can cause overflow.

Example: 8-Bit Machine Integers

Integer	8-Bit Representation
127	01111111
126	01111110
⋮	
1	00000001
0	00000000
-1	11111111
-2	11111110
⋮	
-127	10000001
-128	10000000

Floating Point Numbers

- Rational numbers are represented in base 2 scientific notation with a fixed number of bits:

  $$\pm m * 2^e$$

  where m is called the mantissa and e the exponent.
- Single-precision floating numbers use 32 bits with 1 bit for the sign, 23 bits for the mantissa, and 8 bits for the exponent.
- For convenience, floating point numbers can be expressed in Haskell in base 10:
  - `23.678`, `-0.04`.
  - `59.78e20`, `-59.78e-20`.
- Since the $(0.1)_{10} = (0.00011001100\ldots)_2$, 0.1 cannot be exactly represented as a floating point number.

Floating Point Arithmetic

- Arithmetic operations on floating point numbers return the floating point number that is the best approximation to the true value.
  - `Infinity` is returned if the result is too large to be represented correctly.
  - `-Infinity` is returned if the result is too small to be represented correctly.
  - `NaN` (for "not a number") is returned if the result is not defined (e.g., `sqrt (-1)`).
- Since floating point numbers cannot precisely represent all real (or even rational) numbers, floating point arithmetic can produce inaccurate or even completely bogus results!
  - Addition and multiplication are not associative.
  - Floating point numbers must be used with care!

Floating Point Arithmetic (iClicker)

What are the values of:

1. `(1.0e30 + (-1.0e30)) + 1.0`.
2. `1.0e30 + (-1.0e30 + 1.0)`.

A. 0.0 and 0.0.
B. 0.0 and 1.0.
C. $\boxed{\text{1.0 and 0.0.}}$
D. 1.0 and 1.0.

Numeric Types in Haskell

- A numeric type is a type of values that represents a number system.
- Haskell contains the following built-in numeric types:
  - `Int` (32-bit or 64-bit machine integers).
  - `Integer` (all integers).
  - `Float` (32-bit floating point numbers).
  - `Double` (64-bit floating point numbers).
  - `Rational` (all rational numbers).
- These types have separate arithmetic operators with overloaded names:
  - `+`: addition.
  - `-`: subtraction.
  - `*`: multiplication.
  - `/`: division.
  - `^` and `**`: exponentiation.