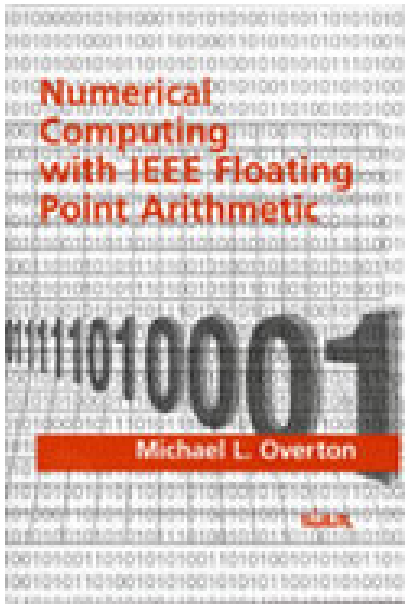
Lecture #1: Floating Point Numbers

Reference

Numerical Computing with IEEE Floating Point Arithmetic

written by Michael L. Overton

published by SIAM



Available at NYU Bookstore

1 Introduction: History of Computing

- Number Systems & Tetrapods

- Rhind Papyrus
  - Egyptian numerals (as Hieroglyphs)
  - Egyptian multiplication (as binary arithmetic)

- Abacus
  - Background
  - Applet

- First Stored Program Computer (EDVAC)

- First Computer Bug

2 Real Numbers:

- Integers, Rational, Irrational Numbers
- Properties (Commutative, Associative, Distributive, Identity, & Inverse)
- Zero
  - As Positional Place Holder
  - As a Number Itself
- Complex Numbers

2 Real Numbers: Positional Number System

- Baseless System (e.g., Roman Numerals)
- Positional system (a.k.a. Digital System)
  - Base 60 (Babylonians … clocks, latitudes)
  - Base 10 (a.k.a. Decimal, Arabic/Indian)
  - Base 2 (a.k.a. Binary, Computerized Logic)

2 Real Numbers: Binary Conversion

- Decimal to Binary
  - Decimal Fractions to Binary
  - Infinite Binary Fractions

- Binary to Decimal

2 Real Numbers: Binary Arithmetic

- Addition
- Subtraction
- Multiplication
- Division

2 Real Numbers: Binary Bits

- System Digits: 0 and 1
- Bit (short for *b*inary dig*it*): a single binary digit
- Bitstring: a string of bits
- LSB (least significant bit): the rightmost bit
- MSB (most significant bit): the leftmost bit

2 Real Numbers: Binary Equivalents

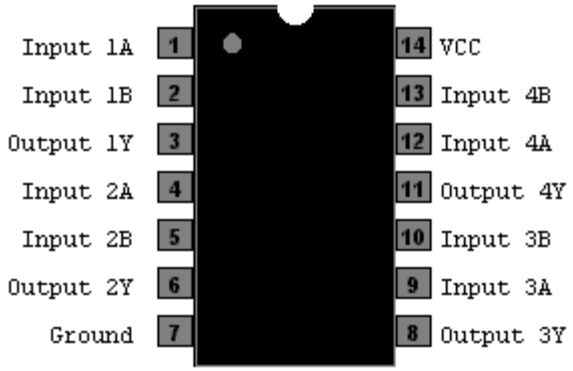
- 1 Nybble (or nibble) = 4 bits

- 1 Byte = 2 nybbles = 8 bits
- Upper Byte : right-hand byte of a pair
- Lower Byte : left-hand byte of a pair
- 1 Kilobyte (KB) = 1024 bytes
- 1 Megabyte (MB) = 1024 kilobytes
- 1 Gigabyte (GB) = 1024 megabytes
- 1 Terabyte (TB) = 1024 gigabytes
- 1 Petabyte (PB) = 1024 terabytes

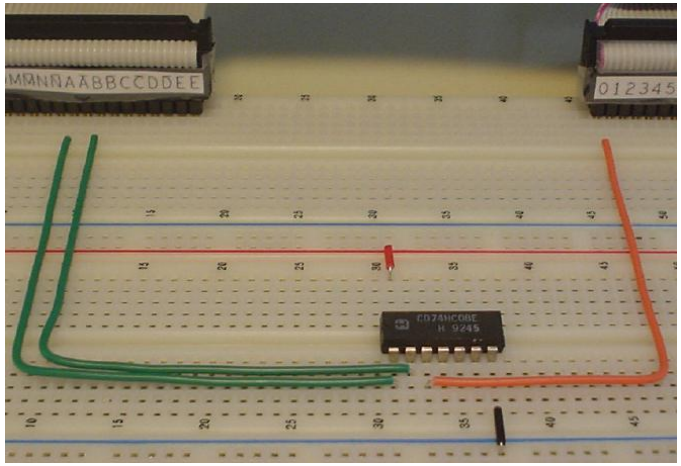
- Word: 4 consecutive bytes of storage
- Double word: 8 consecutive bytes of storage

3 Comp. Rep. of Num.: Logic Gates

- Logic Gates
  - Basic Functions
  - Binary Addition



- History of Gate Size/Speed
  - Vacuum Tube
  - Transistor
  - Integrated Circuit
  - Microprocessor



3 Comp. Rep. of Num.: 2's Complement

- Most computers represent negative integers as 2's complements
- Algorithm
  - Express number in binary
  - Invert the digits
  - Add one to the result
- Allows subtraction to be replace by (negative addition)
- Worked example

3 Comp. Rep. of Num.: Over/Under Flow

- Underflow
- Overflow (ditto)

3 Comp. Rep. of Num.: Fixed Point

- Fixed Point Number – A value with an integer and a fractional part

- Fixed Point Arithmetic

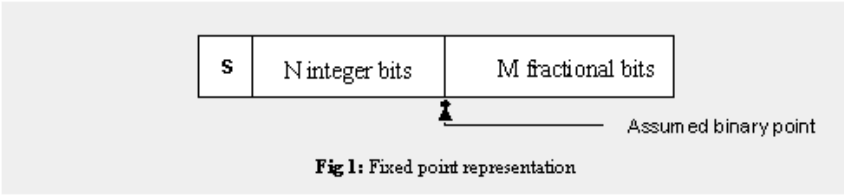


Fig 1: Fixed point representation

3 Comp. Rep. of Num.: Scientific Notation

- Express a real number *x* in scientific notation
  - where S is the *mantissa* (a.k.a. *significand*)
  - and E is the integer exponent

$$x = \pm S \times 10^{E}$$

$$1 \leq S \leq 9$$

- Decimal place ‘floats’ to position immediately after first non-zero digit in the decimal expansion of a number

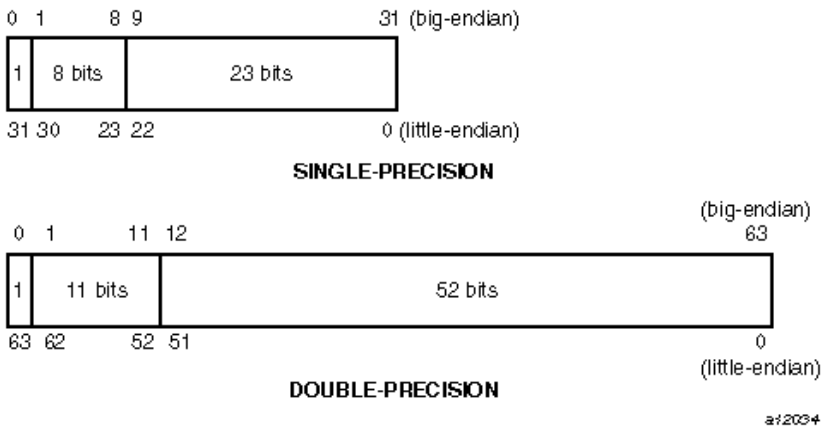
3 Comp. Rep. of Num.: Floating Point

- On a computer, we use base 2: $x = \pm S \times 10^{E}$ $1 \le S \le 2$
- Binary expansion of significand is: (with $b_o = 1$) $S = (b_o . b_1 b_2 b_3 \ldots)$
- Also called *normalized* representation
- Since $b_o = 1$ we represent significand as: $S = (1 . b_1 b_2 b_3 \ldots)$
  - Leading bit is not explicitly stored (implied)

3 Comp. Rep. of Num.: Number Storage

- Computer words shared into three fields (sign, exponent, and significant)

- e.g., a 32-bit word shared as: 1 sign bit, 8 exponent bits, 23 significant
  - Sign bit (0 for positive; 1 for negative)
  - Exponent (represents -128 to 127)
  - Significant (stores first 23 bits after $b_{o,}$, namely $b_1 \ldots b_{23}$)



- Real number *x* exactly representable is a floating point number
  - Otherwise, real number must be rounded to next floating point number

3 Comp. Rep. of Num.: Precision

- precision versus accuracy (scientific meanings)

- precision (denoted *p*) of the floating point system is the number of bits in the significand (including the hidden bit).

- Any normalized floating point number with precision *p* can be expressed:

$$x = \pm \left(1.b_1 b_2 \ldots b_{p-2} b_{p-1}\right)_2 \times 2^E$$

3 Comp. Rep. of Num.: Machine Epsilon

- The smallest x such that x is greater than 1:

$$x = \pm(1.00\ldots01)_2 = 1 + 2^{-(p-1)}$$

- *machine epsilon* defined as the gap between the number above and 1 itself:

$$\varepsilon = \pm(0.00\ldots01)_2 = 2^{-(p-1)}$$

3 Comp. Rep. of Num.: Ulp

- *Ulp* is an acronym for ‘unit in the last place’
- Defined as: $Ulp(x) = \pm(0.00\ldots01)_2 \times 2^E = 2^{-(p-1)} \times 2^E = \varepsilon \times 2^E$
- For x > o, then $ulp(x)$ is the gap between x and the next larger floating point number

3 Comp. Rep. of Num.: Zero

- Zero cannot be normalized
  - the hidden bit is always implied to be 1

- Two solution strategies

  - 1) give up on concept of hidden bit and represent b_o explicitly

  - 2) use special string in exponent field to signal the number is zero (IEEE approach)

4 IEEE Floating Point: Institute(s)

- IEEE – Institute for Electronics and Electrical Engineers
- ANSI – American National Standards Institute
  - on-line store (document 754-1985)
- The need for a standard
  - Portable code was elusive in the 1960s (prior to standard)
  - What every computer scientist should know about floating point arithmetic
- Non-standard implementation
  - How Java's floating point hurts everyone everywhere

4 IEEE Floating Point: Standard Requirements

- Consistent **representation** of floating point numbers by all machines adopting the standard
- Correctly **rounded** floating point operations, using various rounding modes
- Consistent treatment of **exceptional** situations, such as division by zero

4 IEEE Floating Point: Special Numbers

- Zero, and negative zero
- Plus and minus infinity
- Not a number (NaN)

- How to represent these numbers in floating point
  - Special pattern of exponent field ?

4 IEEE Floating Point: Basic Formats

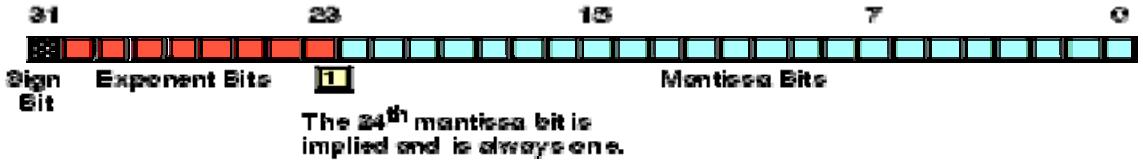
- Single precision – 32 bit word
- Double precision – 64 bit double word
- Extend precision – (up to) 128 bit quad word

- Minimum Range of numbers
  - Single : $2^{-126} \sim 1.2 \times 10^{-38}$
  - Double : $2^{-1022} \sim 2.2 \times 10^{-308}$

- Maximum Range of numbers
  - Single : $2^{+128} \sim 3.4 \times 10^{+38}$
  - Double : $2^{+1024} \sim 1.8 \times 10^{+308}$

- Greater precision needed in some scientific calculations
- Lesser precision needed in some routine operations

4 IEEE Floating Point: Single Precision



- 32 bits available:
  - 1 bit sign; 8 bit exponent; 23 bit significand

- Special numbers: (zero and subnormal)
  - If exponent $(00000000)_2 = (0)_{10}$
  - Then number $(0.b_1b_2\ldots b_{23}) \times 2^{-126}$

- Special numbers: (+/- infinity and Nan)
  - If exponent $(11111111)_2 = (255)_{10}$
  - Then number +/- infinity if $b_1=b_2\ldots=b_{23}=0$
  - Else Nan otherwise

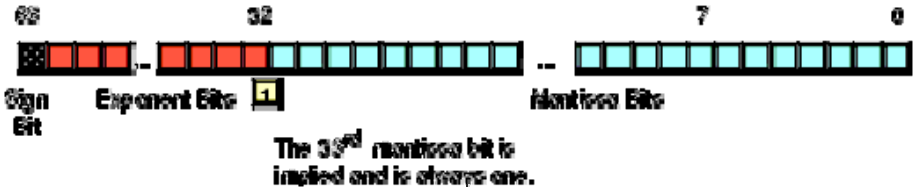
- Normal Numbers
  - Exponent is biased (need to subtract 127)
  - Examples of single precision standard (Pittsburgh Supercomputing Center)

4 IEEE Floating Point: Subnormals

- Arises in situation of zero exponent (hence zero hidden bit)
  - If all fraction bits are zero, $b_1=b_2\ldots=b_{23}=0$
  - Then the number is zero
  - Else a subnormal number is obtained
    - Can represent 2^{-127} through to 2^{-149}
- Subnormals cannot be normalized
  - (if they did, the exponent would not fit in the exponent field)

4 IEEE Floating Point: Double Precision

- 64 bits available:
  - 1 bit sign; 11 bit exponent; 52 bit significand
- Conceptually identical to single precision
- Examples of double precision standard (Pittsburgh Supercomputing Center)
- Under what circumstances does user want single versus double precision ?
  - Scientific measurement ?
  - Scientific calculation ?



4 IEEE Floating Point: Extended Precision

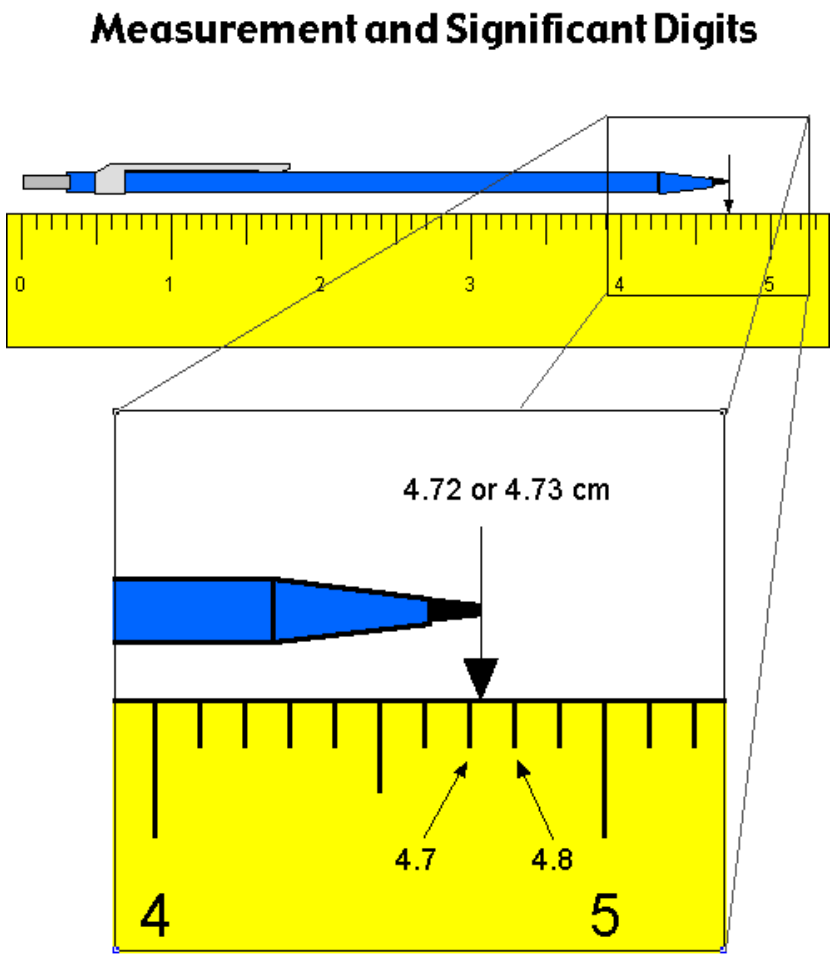
- 80 bits available:
  - 1 bit sign; 15 bit exponent; 64 bit significand

- Differ from single/double precision in that no ‘hidden bit’

- Intel implements extended precision in hardware (fast);
- Sparc implements in 128 bit software (slow)

4 IEEE Floating Point: Significant Digits

- Single precision p=24 bits gives ~7 significant digits
- Double precision p=53 bits gives ~15 significant digits
- Single precision p=64 bits gives ~19 significant digits



4 IEEE Floating Point: Big/Little Endian

- Given a single precision number is stored as a word (i.e., 4 contiguous bytes)
  - where to store sign bit and first seven exponent bits? (to the left or to the right of a 4 byte sequence)

$$\sigma a_1 a_2 \ldots a_7 a_8 b_1 b_2 \ldots b_{22} b_{23}$$

  - To the left is BIG ENDIAN
    - (e.g., SUN, IBM)
  - To the right is LITTLE ENDIAN
    - (e.g., Intel)

- Important when passing data between different Endian computers (some example software)

- Word *Endian* originates in reference to Gulliver's Travels in discussion of which end of a boiled egg to open … the little or big end?

5 Rounding: Real Numbers

- Finite IEEE floating point numbers: $\pm\left(1.b_1 b_2 \ldots b_{p-2} b_{p-1}\right)_2 \times 2^E$
- Real number x is in the normalized range of the floating point system if: $N_{\min} \le |x| \le N_{\max}$
- If a real number x is not a floating point number, one (or both) of the following is true:
  - x is outside the normalized range (N_{max}, N_{min})
  - binary expansion of x requires more than p bits to exactly specify the number

5 Rounding: Neighbor Numbers

- Consider a real number: $x = \pm\left(1.b_1 b_2 \ldots b_{p-2} b_{p-1} b_p b_{p+1} \ldots\right)_2 \times 2^E$

- The closest floating point number less than or equal is (obtained by truncating the significand) $x_- = \pm\left(1.b_1 b_2 \ldots b_{p-2} b_{p-1}\right)_2 \times 2^E$

- The closest floating point number greater than or equal is (obtained by increment the least significant bit) $x_+ = \left(1.b_1 b_2 \ldots b_{p-2} b_{p-1}\right)_2 \times 2^E + \left(0.00\ldots01\right)_2 \times 2^E$

- The gap between x_- and x_+ is *ulp(x):* $Ulp(x) = 2^{-(p-1)} \times 2^E$

5 Rounding: Correctly Rounded Values

- IEEE defines correctly rounded as being *round(x)*

- Four rounding modes in effect:
  - Round down:
    - $round(x)=x_-$
  - Round up:
    - $round(x)=x_+$
  - Round towards zero:
    - If $x > 0$ then $round(x)=x_-$
    - If $x<0$ then $round(x)=x_+$
  - Round to nearest:
    - *round(x)* is either x_- or x_+, whichever is nearer to x
    - In case of tie, the one with its least significant bit equal to zero is chosen

5 Rounding: Absolute Rounding Error

- Define absolute rounding error: $abserr(x) = \left| round(x) - x \right|$
- Consider a real number: $x = \pm \left(1.b_1 b_2 \ldots b_{p-2} b_{p-1} b_p b_{p+1} \ldots\right)_2 \times 2^E$
- The absolute rounding error is less than the gap between x_- and x_+

$$abserr(x) = \left| round(x) - x \right| < 2^{-(p-1)} \times 2^E$$

- Absolute rounding error is less than one Ulp
- When round to nearest in effect, then absolute rounding error is less than or equal to ½ of an Ulp

$$abserr(x) = \left| round(x) - x \right| < 2^{-p} \times 2^E$$

5 Rounding: Relative Rounding Error

- Relative rounding error (for nonzero x) is defined:

$$relerr(x) = |\delta|$$

$$\delta = \frac{|round(x) - x|}{x}$$

- Relative error satisfies the bound:

$$\delta = \frac{|round(x) - x|}{x} < \frac{2^{-(p-1)} \times 2^{E}}{2^{E}} = 2^{-(p-1)} = \varepsilon$$

$$where \ \ |x| > 2^{E}$$

- For round to nearest we get:

$$\delta < \frac{\varepsilon}{2}$$

- Theorem:

$$round(x) = x(1+\delta)$$

$$for \ \ some \ \ |\delta| < \varepsilon$$

- Thus, no matter how x is stored, we can think of its values as exact tihin a factor of *1+epsilon*