Computer Architecture

Discussion 1

Number Representation

Yanpeng Zhao
Feb 23, 2016

Numeration systems

- Decimal code
  - Positive integer
    - $125_{10} = 1 * 100 + 2 * 10 + 5 * 1 = 1 * 10^2 + 2 * 10^1 + 5 * 10^0$
  - Fraction
    - $25.43_{10} = 2 * 10 + 5 * 1 + 4 * 0.1 + 3 * 0.01$

      $= 2 * 10^1 + 5 * 10^0 + 4 * 10^{-1} + 3 * 10^{-2}$
- Binary code
  - Positive integer
    - $86_{10} = 1 * 64 + 0 * 32 + 1 * 16 + 0 * 8 + 1 * 4 + 1 * 2 + 0 * 1$

      $= 1 * 2^6 + 0 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$
  - Fraction
    - Think about it.

Signed number representations

- Aim
  - In computing, signed number representations are required to encode negative numbers in binary number systems.
- Representation schemes
  - Signed magnitude representation
  - Ones' complement
  - Two's complement
  - ... …
- Require one bit to be used as the sign bit

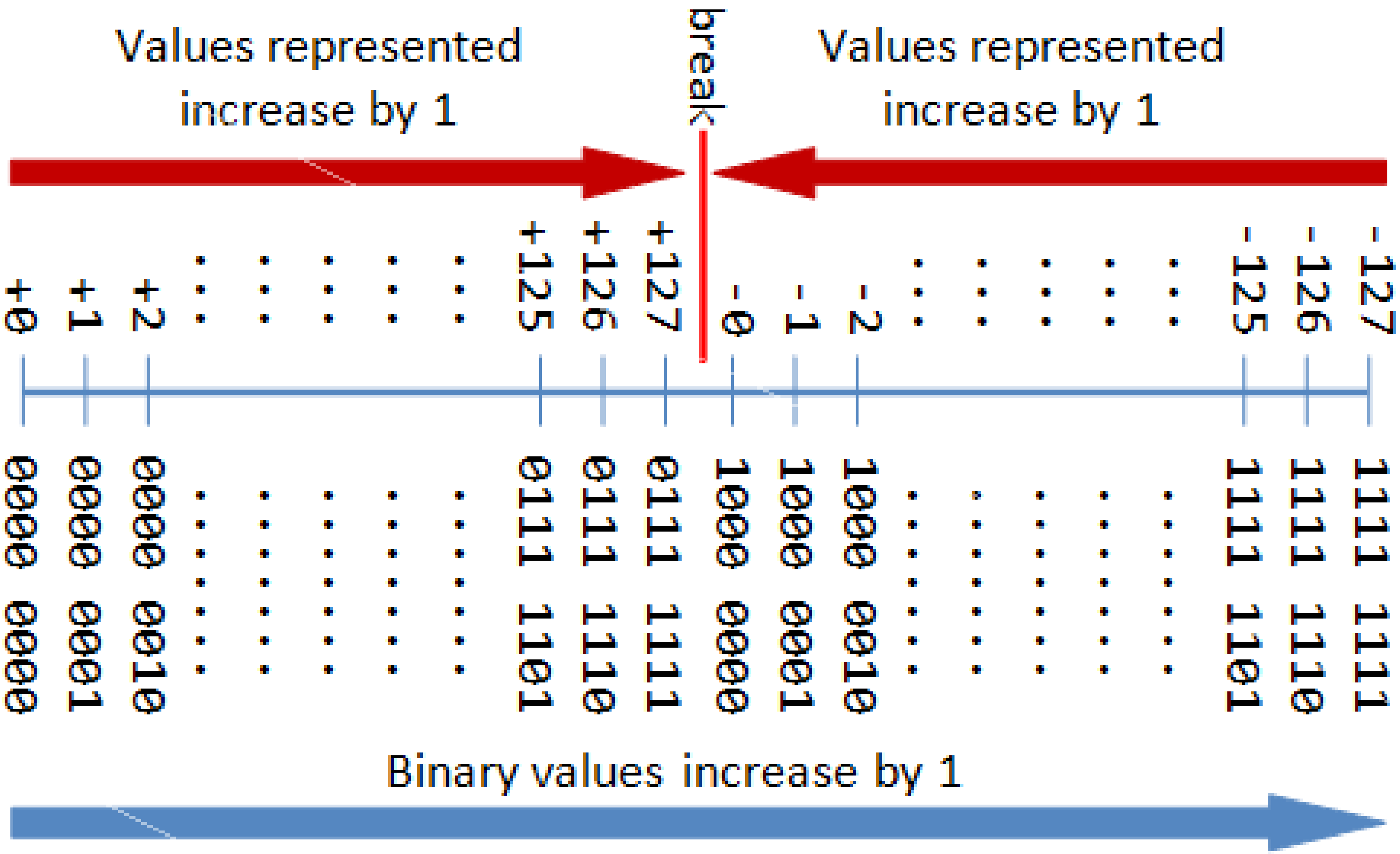
Signed magnitude representation

- Scheme
  - Setting the most significant bit to 0 is for a positive number, and setting it to 1 is for a negative number. The remaining bits in the number indicate the magnitude (or absolute value).
- Example
  - +1 = 0000 0001; -1 = 1000 0001
  - Ranging from -127_{10} to $+127_{10}$
  - 00000000 (0) = 10000000 (−0)

Signed magnitude representation

- Scheme
  - Setting the most significant bit to 0 is for a positive number, and setting it to 1 is for a negative number. The remaining bits in the number indicate the magnitude (or absolute value).
- Example
  - +1 = 0000 0001; -1 = 1000 0001
  - Ranging from -127_{10} to $+127_{10}$
  - 00000000 (0) = 10000000 (−0)

Signed magnitude representation

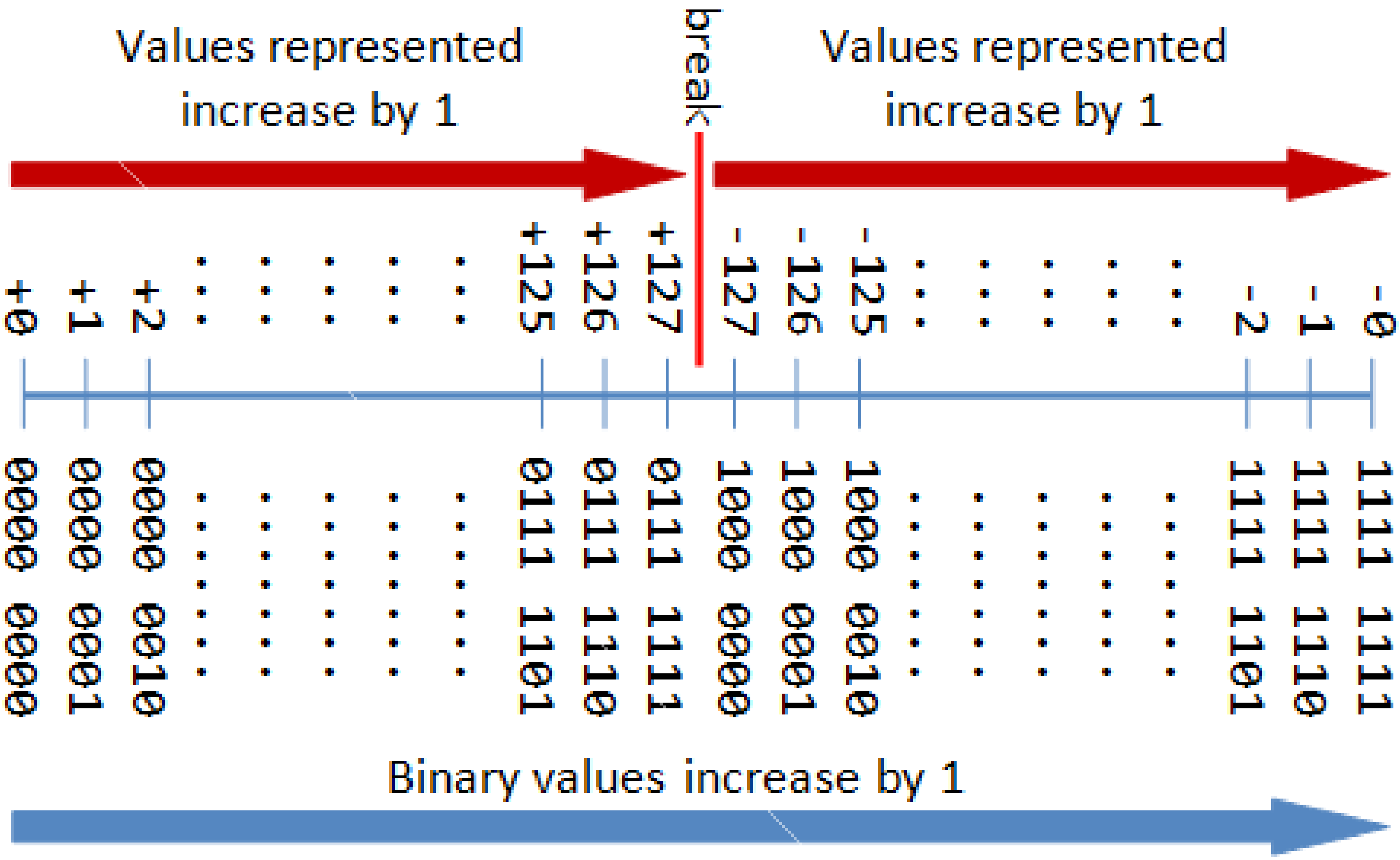


Sign-Magnitude Representation

Ones' Complement

- Scheme
  - The ones' complement form of a negative binary number is the bitwise NOT applied to it — the "complement" of its positive counterpart.
- Example
  - +1 = 0000 0001; -1 = 11111110
  - Ranging from -127_{10} to $+127_{10}$
  - 00000000 (+0) = 11111111 (−0)

Ones' Complement

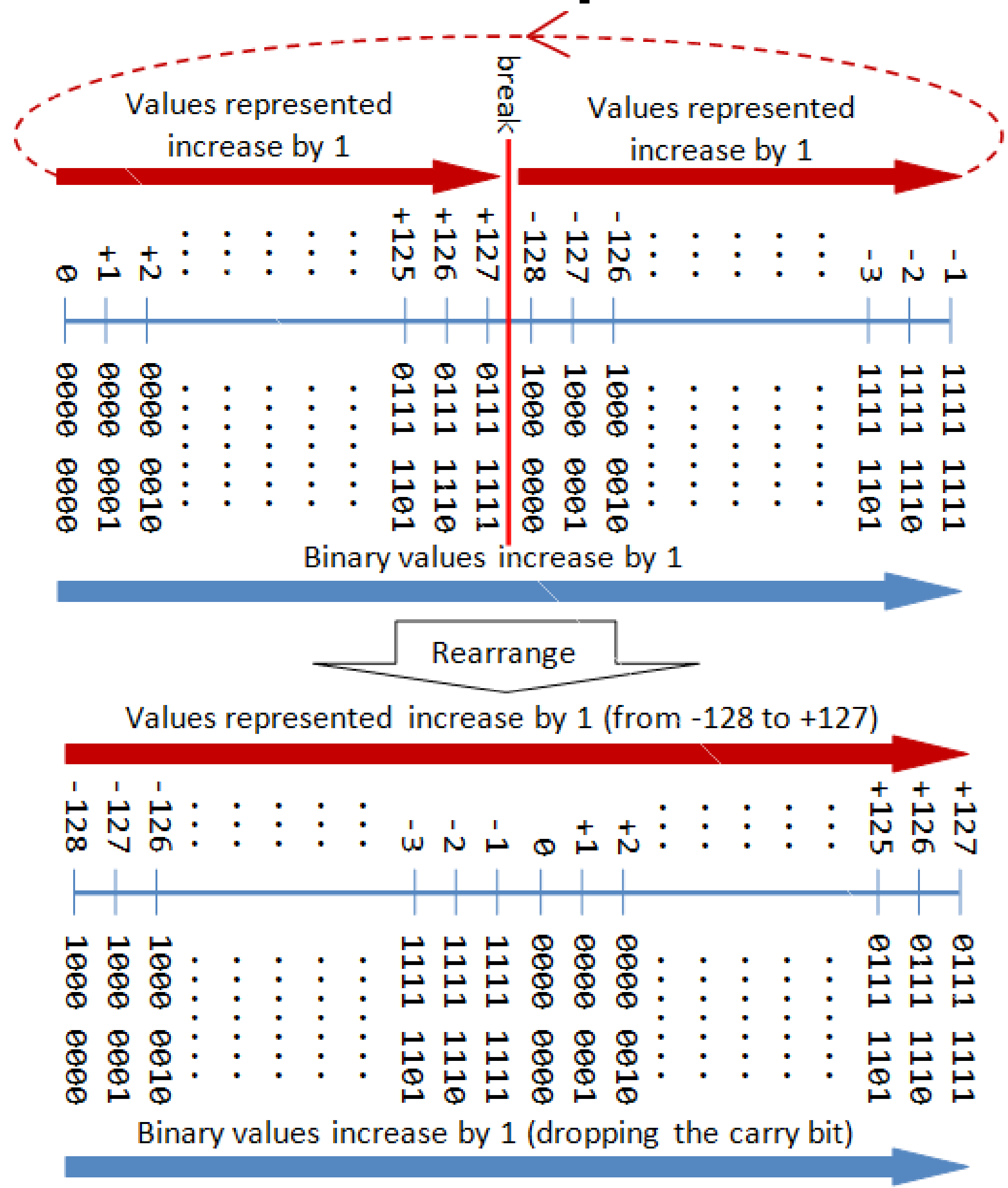


1's Complement Representation

Two's complement

- Scheme
  - In two's complement, negative numbers are represented by the bit pattern which is one greater (in an unsigned sense) than the ones' complement of the positive value.
- Example
  - +1 = 0000 0001; -1 = 11111111
  - Ranging from -128_{10} to $+127_{10}$
  - Only one zero: 00000000 (0)

Two's complement



2's Complement Representation

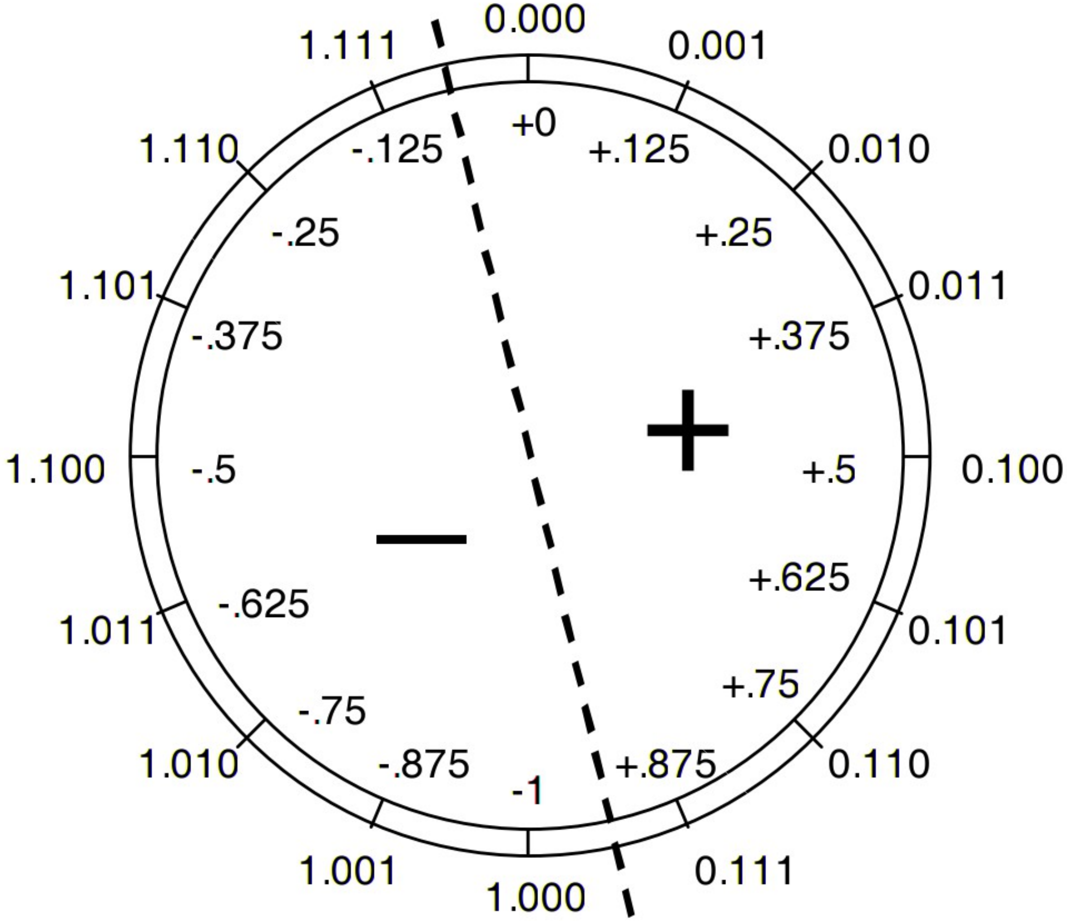
Why do we use Two's complement

- To involve sign bit in the calculation correctly
- Signed magnitude representation
  - $1 - 1 = [0000\ 0001]_S + [1000\ 0001]_S = [1000\ 0010]_S = -2$
  - Incorrect
- Ones' complement
  - $1 - 1 = [0000\ 0001]_O + [1111\ 1110]_O = [1111\ 1111]_O = [1000\ 0000]_S = -0$
  - Meaningless sign bit since -0 = +0 in terms of the magnitude
- Two's complement
  - $1 - 1 = [0000\ 0001]_T + [1111\ 1111]_T = [0000\ 0000]_T=[0000\ 0000]_S = 0$
  - $-128_{10} = [1000\ 0000]_T$;

Fixed-Point Numbers

- Fixed-point number
  - A fixed-point number consists of a whole or integral part and a fractional part, with the two parts separated by a radix point (decimal point in radix 10, binary point in radix 2, and so on)
- A fixed-point number has *k* whole digits and *l* fractional digits
  - $x = \sum_{i=-l \text{ to } k-1} x_i r^i = (x_{k-1}x_{k-2} \dots x_1x_0 \bullet x_{-1}x_{-2} \dots x_{-l})_r$
  - $2.375 = (1 \times 2^1) + (0 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) = (10\bullet011)_{\text{two}}$
- Value range
  - The positive values ranges from 0 to $2^{k-1} - 2^{-l}$
  - The negative values ranges from -2^{-l} to -2^{k-1}

Fixed-Point Numbers



Schematic representation of 4-bit 2's-complement encoding for (1 + 3)-bit fixedpoint numbers in the range [–1, +7/8].

Fixed-Point Numbers

- Fixed-point number
  - The two important properties of 2's-complement numbers, previously mentioned in connection with integers, are valid here as well.
    - The leftmost bit of the number acts a the sign bit
    - The value represented by a particular bit pattern can be derived by considering the sign bit as having a negative weight
- Example
  - $(01.011)_{2\text{'s-compl}} = (-0 \times 2^{1}) + (1 \times 2^{0}) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) = +1.375$

    $(11.011)_{2\text{'s-compl}} = (-1 \times 2^{1}) + (1 \times 2^{0}) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) = -0.625$
  - How to distinguish the negative ones from the positive ones, i.e, 1.01 of (2 + 2)-bit fixed point numbers?
  - $1.1_T = (-1)^1 * (0b0.0 + 1)$ where $0b0.0 = \sim 0b1.1$

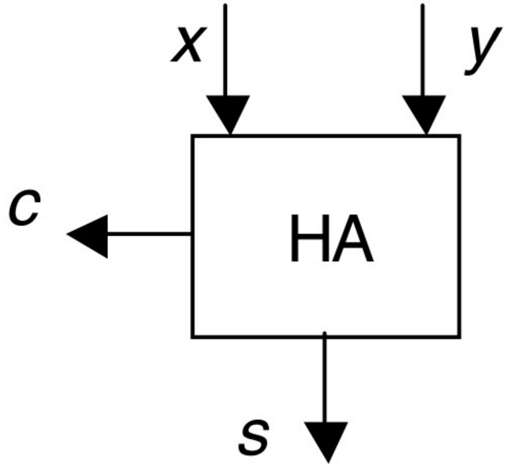
Fixed-Point Numbers

- How to convert a decimal fraction into the binary fraction?
  - Idea: Multiplied or divided by radix-2 means to move the point to the right or the left by 1 bit. We only need to record the rightmost bit to the left of the point every time when we do multiplication or division.
  - Have a try: -0.5 = 0b(?), 0.5 = 0b(?)
- Numerical error
  - $0.4_{10} = 0.01100110(0110)\ldots$

Addition and Subtraction

- Half-adder
  - The circuit that can compute the sum and carry bits is known as a *half-adder* (HA)

Inputs		Outputs	
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

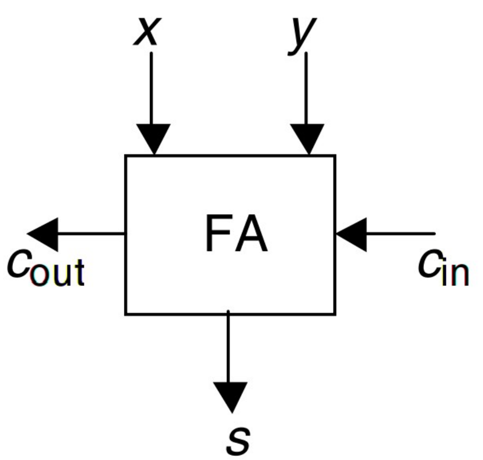


Truth table and schematic diagram for a binary half-adder. The carry output is the logical AND of the two inputs, while the sum output is the exclusive OR (XOR) of the inputs.

Addition and Subtraction

- Full-adder
  - By adding a carry input to a half-adder, we get a binary *full adder* (FA)

Inputs			Outputs	
x	y	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Truth table and schematic diagram for a binary full-adder. A full-adder, connected to a flip-flop for holding the carry bit from one cycle to the next, functions as a bit-serial adder.

Addition and Subtraction

- Have a try!
  - -3 + 2; 4 + 3; 1.2 + 2.3; -0.5 +1.5
- Think about this.

  Overflow occurs only in the addition of two positive numbers or negative numbers, wouldn’t occur in the addition of the positive number and negative number.

Multiplication and Division

- Not covered

Real Numbers

- Most real numbers must be approximated within the machine’s finite word width. (i.e, sizeof(float) = 4)
- Drawbacks of fixed-point representation
  - Not very good for dealing with very large and extremely small numbers at the same time.

    $x = (0000\ 0000\ .\ 0000\ 1001)_{\text{two}}$ Small number

    $y = (1001\ 0000\ .\ 0000\ 0000)_{\text{two}}$ Large number

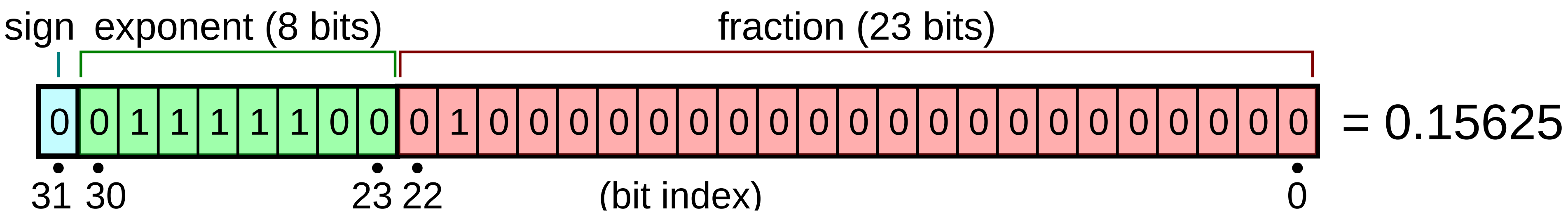
  - The relative representation error due to truncation or rounding of digits beyond the -8^{th} position is quite significant for x, but it is much less severe for y.

Float-Point Numbers

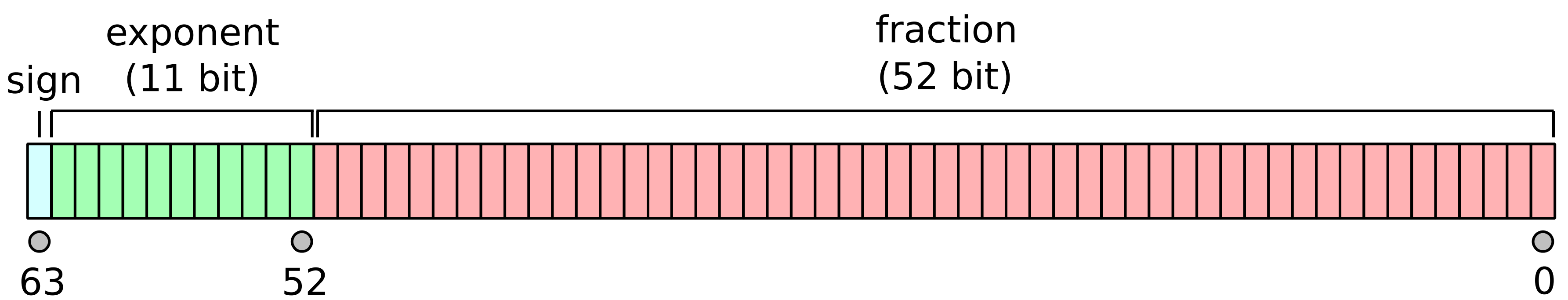
- Floating point
  - Floating point is the formulaic representation that approximates a real number so as to support a trade-off between range and precision.
- Representation
  - A number is, in general, represented approximately to a fixed number of significant digits (the significand) and scaled using an exponent
  - Scientific notation: significand x base^{exponent}
  - Significand ranges from 1(inclusive) to 2(exclusive)

IEEE floating point

- Single-precision floating-point format



- Double-precision floating-point format



IEEE floating point

- Exponent bias
  - The exponent is biased in the engineering sense of the word – the value stored is offset from the actual value by the exponent bias.
  - To calculate the bias for an arbitarily sized floating point number apply the formula $2^{k-1} - 1$ where k is the number of bits in the exponent.
  - For a single-precision number, an exponent in the range −126 .. +127 is biased by adding 127 to get a value in the range 1 .. 254 (0 and 255 have special meanings).
  - For a double-precision number, an exponent in the range −1022 .. +1023 is biased by adding 1023.

IEEE floating point

- An example
  - Represent 38414.4 in double
    - Integral part: 0x960E
    - Fraction part: 0.4=0.5×0+0.25×1+0.125×1+……+0.5×（1 or 0）/n+…
    - 38414.4_{10} = b1001011000001110.0110011001100110011001100110011001100 (52 + 1 = 53 bits)
    - Scientific notation: $1.001011000001110011001100110011001100110011001100110 0\times2^{15}$
    - Biased exponent: $(15+1023=1038)_{10}$ = 10000001110
    - Sign bit: 0
    - Output:
      - 0 10000001110 0010110000011100110011001100110011001100110011001100
- Try to convert the output above into decimal number
  - Tip: take care of the significand
- How to represent -12.5?
  - 1 10000010 10010000000000000000000
  - Refer to the code
    http://www.piazza.com/class_profile/get_resource/iksfk6wahl15bn/ikziuwxghst4fk

References

- https://en.wikipedia.org/
- https://www.ece.ucsb.edu/~parhami/pubs_folder/parh02-arith-encycl-infosys.pdf
- http://www.swarthmore.edu/NatSci/echeeve1/Ref/BinaryMath/NumSys.html
- http://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html
- You can get all kinds of resources here

  https://www.google.com/search?q=number%20representation

Thanks!

Q & A