Floating Point Numbers

Integer Representation & Arithmetic

- Unsigned
- Signed magnitude
- 2's compliment

Unsigned Number

- $A = \sum_{i=0}^{n-1} 2^i a_i$
  - Ex: $01010101_2 = 85_{10}$
    $00000000_2 = 0_{10}$
    $11111111_2 = 255_{10}$
- The range of the number $[0, 2^n-1]$. (for n bits number)
  - For 8-bit
    - min: 00000000, max: 11111111
- No negative number !

Sign-Magnitude

- sign + magnitude
- sign
  - 0 means positive, 1 means negative
  - Ex:
    - $+18_{10} = 00010010_2$, $-18_{10} = 10010010_2$
- Range: $[-(2^{(n-1)} – 1), (2^{(n-1)} – 1)]$
  - For 8 bit: max = $01111111_2 = 127_{10}$, min = $11111111_2 = -127_{10}$
- Problems
  - Sign check
  - Two representations of zero (+0 and -0)

Two's Compliment

- MSB is the sign bit:
  - 0 – positive, 1 – negative
- For the rest of the bits
  - ***When MSB = 0, same as the unsigned binary number***
  - ***When MSB = 1, invert each bit in the unsigned number and then add 1***
- Ex:
  - unsigned number $1710_{10} = 00010001_2$
    2's complement number for $1710_{10} = 00010001_2$
    2's complement number for $-1710_{10} = 11101111_2$

Two's Compliment

- **Number range $[-2^{(n-1)}, 2^{(n-1)}-1]$**
  - For 8 bit number: min = $10000000_2 = -128_{10}$
    max = $01111111_2 = 127$
- **Converting n-bit to m-bit (n<m)**
  - **Appending m-n copies of sign bit**

Ex: 4-bit	***0101***	***1101***
8-bit	***00000101***	***11111101***

2's Compliment Addition/Subtraction

- **Addition**
  - **Just like the pencil-and-paper algorithm**
    - ***Fixed number of bits***
    - ***Don't have to worry about the sign bits***
  - Overflow problem
    - ***The result exceeds the range of the number system***
      - Ex: -3 – 6 = -3 (1101) + (-6) (1010) = -9 (0111 -→ +7)
    - ***Adding two large numbers with the same sign***
  - ***How to check it:***
    - **Two operands have the same sign but the result have different sign.**
- **Subtraction**
  - **Transform to an addition**

Examples

- Overflow or not?
  - 1101101110 – 11011
  - 1100100110 - 0110110011
  - 1101101110 – 011111

Real Numbers

- Numbers with fractions
- Could be done in pure binary

  1001.1010 = $2^4 + 2^0 + 2^{-1} + 2^{-3}$ =9.625
- Where is the binary point?
  - Fixed?
    - Very limited
  - Floating?
    - How do you show where it is?

Floating Point Representation

- Scientific notation
  - Ex: 27600000 = 2.76 x 10^{7} 0.000000276= 2.76x 10^{-7}
- Floating point binary representation
  - +/- .significand x $2^{exponent}$
  - Ex: 32-bit floating number

Sign bit (1)	Exponent (8)	Significand or Mantissa (23)

Floating Point Representation

- Sign bit
  - 0 – positive
  - 1-- negative
- Exponent is in biased notation
  - What we mean by "biased notation"?
    - The bias: A fixed value, usually equals $2^{K-1} - 1$ (where K is length of the exponent)
    - The true exponent should be the one that subtract the bias from the value in the exponent field

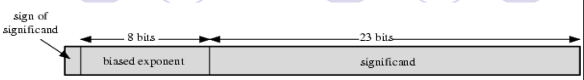
Floating Point Representation

- Exponent is in biased notation
  - Example
    - 8 bit exponent field (K=8)
    - value in the exponent field 0b10101010 = 170
    - bias $2^{K-1} - 1$= 127
    - Pure value range 0-255, current value = 170
    - Subtract 127 to get correct value, i.e. 170 – 127 = 43
    - Range -127 to +128
  - Why biased notation?
    - Bias numbers can be treated similar to unsigned integers with order of the number unchanged
    - Easy for comparing two floating numbers

Floating Point Representation

- Sign
- Exponent is in biased notation
- The significand (mantissa)
  - Normalization
    - Exponent is adjusted so that the leading bit (MSB) of mantissa is 1.
    - Normalized form +/- 1.bbbbbbb x 2 (+/- E)
    - Ex:
      - Not normalized 0.110 x 25
      - Normalized 1.100 x 24
    - Since it is always 1 there is no need to store it

Floating Point Examples



(a) Format

$$\begin{aligned}
1.1010001 \times 2^{10100} &= 0\ 10010011\ 10100010000000000000000 = 1.638125 \times 2^{20} \\
-1.1010001 \times 2^{10100} &= 1\ 10010011\ 10100010000000000000000 = -1.638125 \times 2^{20} \\
1.1010001 \times 2^{-10100} &= 0\ 01101011\ 10100010000000000000000 = 1.638125 \times 2^{-20} \\
-1.1010001 \times 2^{-10100} &= 1\ 01101011\ 10100010000000000000000 = -1.638125 \times 2^{-20}
\end{aligned}$$

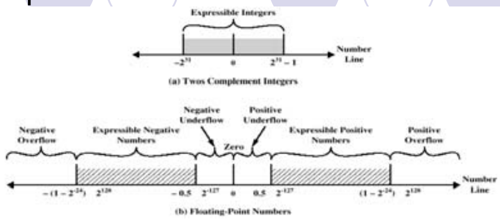
From actual exponent to machine representation:
$10100_2 = 20_{10}$ $20_{10} + 127_{10}$(bias) $= 147_{10} = 10010011_2$
-10100_2 + bias $(0111\ 1111_2) = 01101011_2$

From machine representation to actual exponent:
biased value $= 10010011_2 = 147$ actual value $= 147_{10} - 127_{10} = 20$
biased value $= 01101011_2 = 107$ actual value $= 107_{10} - 127_{10} = -20$

Several Issues

- Expressible numbers (for a 32 bit number)
- Overflow/underflow
  - Negative/Positive overflow/underflow
- Representation of zero?
  - Special pattern, e.g. both exponent and mantissa are zero's
- Accuracy
  - Not represent more individual values
    - Extend the range
  - Not space evenly

Expressible Numbers

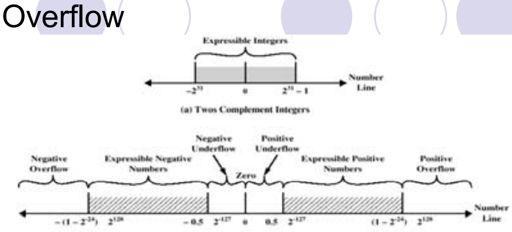


- Absolute value
  - Largest → largest significand (1.1…1) + largest exponent (11…1) = $(2 – 2^{-23}) \times 2^{128}$
  - Smallest → smallest significand (1.0…0) + smallest exponent (00…0) = 1×2^{-127}
- Negative number range: $[-(2 – 2^{-24}) \times 2^{128}, -2^{-127}]$
- Positive number range: $[2^{-127}, (2-2^{-23}) \times 2^{128}]$

Several Issues

- Expressible numbers (for a 32 bit number)
- Overflow/underflow
  - Negative/Positive overflow/underflow
- Representation of zero?
  - Special pattern, e.g. both exponent and mantissa are zero's
- Accuracy
  - Not represent more individual values
    - Extend the range
  - Not space evenly

Overflow

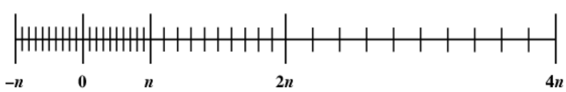


- When an arithmetic operation leads to a result that is out of the expressible regions
  - Negative/Positive overflow/underflow

Several Issues

- Expressible numbers (for a 32 bit number)
- Overflow/underflow
  - Negative/Positive overflow/underflow
- Representation of zero?
  - Special pattern, e.g. both exponent and mantissa are zero's
- Accuracy
  - Not represent more individual values
    - Extend the range
  - Not space evenly

Density of Floating Point Numbers



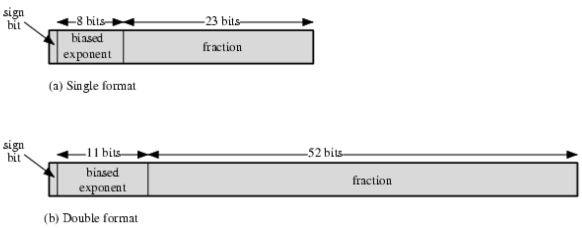
Increase accuracy?

- Use more bits, e.g. double

IEEE Standard

- Standard for floating point storage
- 32 and 64 bit standards
- 8 and 11 bit exponent respectively
- Extended formats (both mantissa and exponent) for intermediate results

IEEE 754 Formats



Example

- IEEE 754 binary single/double representation of -0.75_{10}

$-0.75_{10} = -0.11_2$
$= -1.1 \times 2^{-1}$ (normalized scientific notation)

For single precision
sign = 1 (negative)
biased value = -1 + 127 = 126_{10} = 01111110_2 (unsigned)
exponent = 01111110
significand (23bits): 1000 0000 0000 0000 0000 000

For double precision
sign = 1 (negative)
biased value = -1 + 1023 = 1022_{10} = 01111111110_2 (unsigned)
exponent : 01111111110
significand (52-bits): 1000 0000 0000 0000 0000 000 … 0

Example

- Decimal value for IEEE 754 binary single representation
  - Sign 1
  - Exponent 10000001 (8bits)
  - Significand 010000…0 (23bits)

- Significand = 1.01_2 = 1.25
- Biased value = 10000001_2 (unsigned) = 129
- Actual exponent value = 129 – 127 = 2
- Sign = 1 (negative)
- So the decimal value = - 1.25 x 2^2 = -5.0

Exponent Representation Summary

- Biased representation
  - Bias
    - Constant ($2^{(k-1)}-1$)
    - k=8, bias = 127; k=11, bias = 1023
- Actual value to machine representation
  - biased value = actual value + bias
  - Convert the biased value to unsigned integer
- Machine representation to actual value
  - Convert the biased value (unsigned integer) to decimal value
  - actual value = biased value (decimal) - bias

FP Arithmetic +/-

- Check for zeros
- Align significands (adjusting exponents)
- Add or subtract significands
- Normalize result

FP Arithmetic Examples

- **Decimal**
  - **$1.03 \times 10^0 - 4.56 \times 10^{-2}$**
    $= 1.03 \times 10^0 - 0.0456 \times 10^0$
    $= 0.9844 \times 10^0 = 9.844 \times 10^{-1}$
- **Binary**
  - **$1.101 \times 2^{-1010} + 1.011 \times 2^{-1011}$**
    $= 1.101 \times 2^{-1010} + 0.1011 \times 2^{-1010}$
    $= 10.0101 \times 2^{-1010}$
    $= 1.00101 \times 2^{-1001}$

Summary

- Integer representation/range of numbers
  - Unsigned
  - Signed magnitute
  - 2's complement
- 2's complement addition/subtraction
  - how
  - overflow
- Floating point representation
  - sign/exponent/significand
  - biased representation
  - data range
  - over/under flow
- Floating point addition/subtraction