Master in High Performance Computing

The Abdus Salam International Centre for Theoretical Physics

Luca Heltai
Director of the Joint SISSA-ICTP Master in High Performance Computing
Assistant Professor @ SISSA mathLab

Floating-Point Math

SISSA
International School for Advanced Studies

Exponential Notation

- The following are equivalent representations of 1,234

$123{,}400.0 \times 10^{-2}$

$12{,}340.0 \times 10^{-1}$

$\mathbf{1{,}234.0 \times 10^{0}}$

123.4×10^{1}

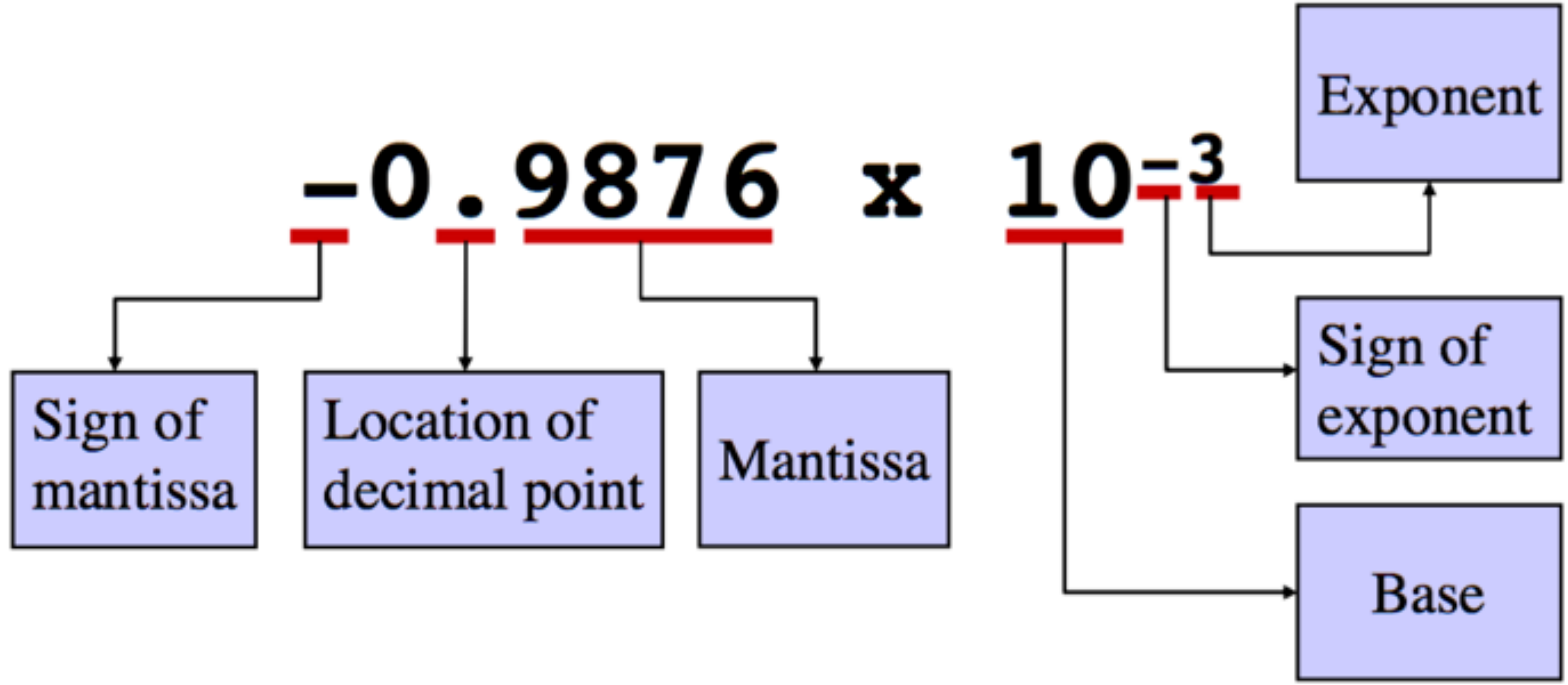
12.34×10^{2}

1.234×10^{3}

0.1234×10^{4}

The representations differ in that the decimal place – the “point” -- “floats” to the left or right (with the appropriate adjustment in the exponent).

Floating-Point Representation (I)



-0.9876 x 10^{-3}

Sign of mantissa

Location of decimal point

Mantissa

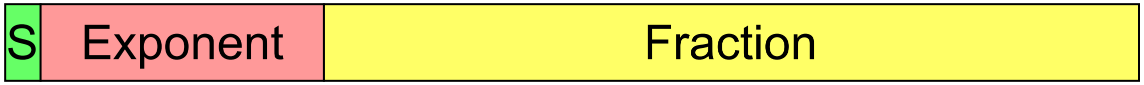
Base

Sign of exponent

Exponent

Floating-Point Representation (II)

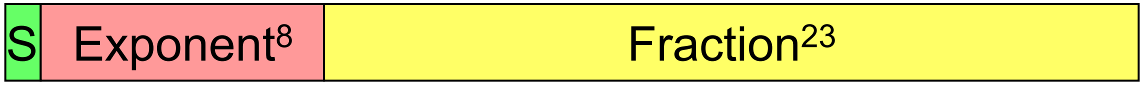
- A floating-point number is represented by the triple
  - *S* is the Sign bit (0 is positive and 1 is negative)
    - Representation is called sign and magnitude
  - *E* is the Exponent field (signed)
    - Very large numbers have large positive exponents
    - Very small close-to-zero numbers have negative exponents
    - More bits in exponent field increases range of values
  - *F* is the Fraction field (fraction after binary point)
    - More bits in fraction field improves the precision of FP numbers



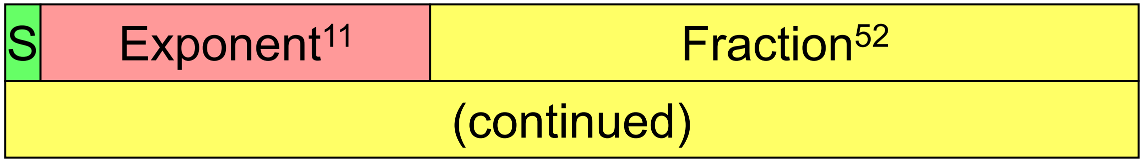
Value of a floating-point number = $(-1)^{S} \times \mathrm{val}(F) \times 2^{\mathrm{val}(E)}$

IEEE 754 Floating-Point Standard

- Found in virtually every computer invented since 1980
  - Simplified porting of floating-point numbers
  - Unified the development of floating-point algorithms
  - Increased the accuracy of floating-point numbers
- Single Precision Floating Point Numbers (32 bits)
  - 1-bit sign + 8-bit exponent + 23-bit fraction



- Double Precision Floating Point Numbers (64 bits)
  - 1-bit sign + 11-bit exponent + 52-bit fraction



Number limits

- Single precision: $\sim\pm1.2*10^{-38} < x < \sim\pm3.4*10^{38}$
- actual precision: ~7 decimal digits
- In comparison: signed 32-bit integer numbers range only from -214783648 to 214783647 and the smallest positive number is 1

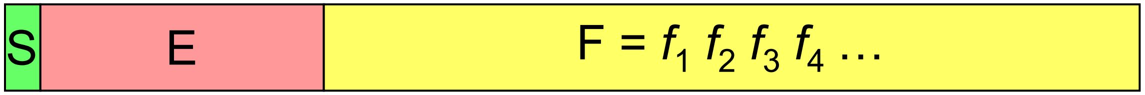
- Double precision: $\sim\pm2.2*10^{-308} < x < \sim\pm1.8*10^{308}$
- actual precision: ~15 decimal digits

Floating-point Math Pitfalls

- Floating point math is commutative, but not associative! Example (single precision):
- $1.0 + (1.5*10^{38} + (- 1.5*10^{38})) = 1.0$
- $(1.0 + 1.5*10^{38}) + (- 1.5*10^{38}) = 0.0$
  - the result of a summation depends on the order of how the numbers are summed up
  - results may change significantly, if a compiler changes the order of operations for optimisation
  - prefer adding numbers of same magnitude
  - avoid subtracting very similar numbers

Normalized Floating Point

- For a normalized floating point number (S, E, F)



- Significand is equal to $(1.F)_2 = (1.f_1f_2f_3f_4\ldots)_2$
  - IEEE 754 assumes hidden 1. (not stored) for normalized numbers
  - Significand is 1 bit longer than fraction
- Value of a Normalized Floating Point Number is

$$(-1)^S \times (1.F)_2 \times 2^{\mathrm{val}(E)}$$

$$(-1)^S \times (1.f_1f_2f_3f_4\ldots)_2 \times 2^{\mathrm{val}(E)}$$

$$(-1)^S \times (1 + f_1\times2^{-1} + f_2\times2^{-2} + f_3\times2^{-3} + f_4\times2^{-4}\ldots)_2 \times 2^{\mathrm{val}(E)}$$

Biased Exponent Representation

- How to represent a signed exponent? Choices are …
  - Sign + magnitude representation for the exponent
  - Two's complement representation
  - Biased representation
- IEEE 754 uses biased representation for the exponent
  - Value of exponent = val(E) = E – Bias (Bias is a constant)
- Recall that exponent field is 8 bits for single precision
  - E can be in the range 0 to 255
  - E = 0 and E = 255 are reserved for special use (discussed later)
  - E = 1 to 254 are used for normalized floating point numbers
  - Bias = 127 (half of 254), val(E) = E – 127
  - val(E=1) = –126, val(E=127) = 0, val(E=254) = 127

Biased Exponent – Cont'd

- For double precision, exponent field is 11 bits
  - E can be in the range 0 to 2047
  - $E = 0$ and $E = 2047$ are reserved for special use
  - $E = 1$ to 2046 are used for normalized floating point numbers
  - Bias = 1023 (half of 2046), val(E) = E – 1023
  - val(E=1) = –1022, val(E=1023) = 0, val(E=2046) = 1023
- Value of a Normalized Floating Point Number is

$$(-1)^S \times (1.F)_2 \times 2^{E - \text{Bias}}$$

$$(-1)^S \times (1.f_1 f_2 f_3 f_4 \ldots)_2 \times 2^{E - \text{Bias}}$$

$$(-1)^S \times (1 + f_1 \times 2^{-1} + f_2 \times 2^{-2} + f_3 \times 2^{-3} + f_4 \times 2^{-4} \ldots)_2 \times 2^{E - \text{Bias}}$$

Examples of Single Precision

- What is the decimal value of this Single Precision float?

1	0	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Solution:
  - Sign = 1 is negative
  - Exponent = $(01111100)_2 = 124$, E – bias = 124 – 127 = –3
  - Significand = $(1.0100 \ldots 0)_2 = 1 + 2^{-2} = 1.25$ (1. is implicit)
  - Value in decimal = $-1.25 \times 2^{-3} = -0.15625$
- What is the decimal value of?

0	1	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

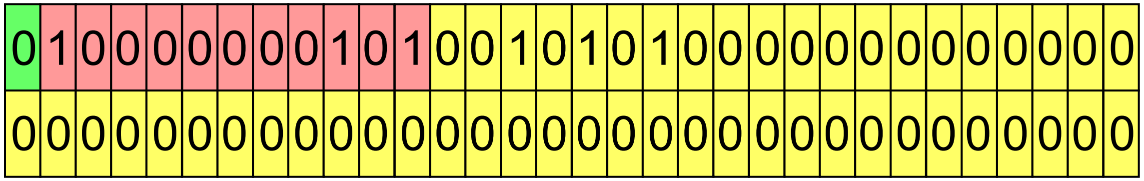
- Solution:

implicit

  - Value in decimal = $+(1.01001100 \ldots 0)_2 \times 2^{130-127} =$

Examples of Double Precision

- What is the decimal value of this Double Precision float ?

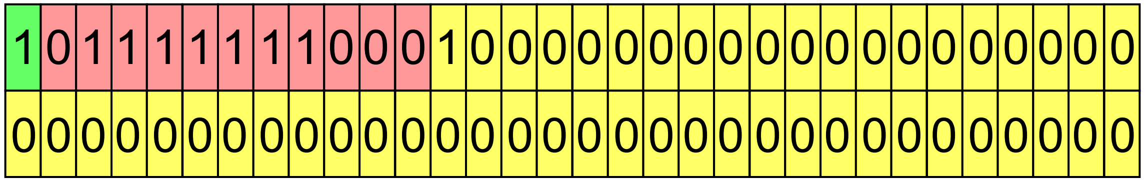


0	1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Solution:
  - Value of exponent = $(10000000101)_2$ – Bias = 1029 – 1023 = 6
  - Value of double float = $(1.00101010 \ldots 0)_2 \times 2^6$ (1. is implicit) =

    $(1001010.10 \ldots 0)_2$ = 74.5

- What is the decimal value of ?



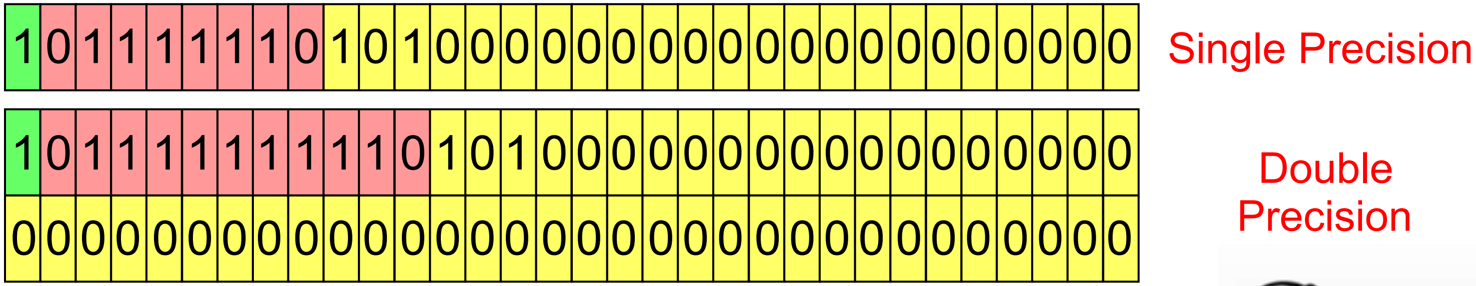
1	0	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- Do it yourself! (answer should be $-1.5 \times 2^{-7} = -0.01171875$)

Converting FP Decimal to Binary

- Convert –0.8125 to binary in single and double precision
- Solution:
  - Fraction bits can be obtained using multiplication by 2
    - 0.8125 × 2 = 1.625
    - 0.625 × 2 = 1.25
    - 0.25 × 2 = 0.5
    - 0.5 × 2 = 1.0
    - Stop when fractional part is 0
  - Fraction = $(0.1101)_2 = (1.101)_2 \times 2^{-1}$ (Normalized)
  - Exponent = –1 + Bias = 126 (single precision) and 1022 (double)

$0.8125 = (0.1101)_2 = ½ + ¼ + 1/16 = 13/16$



1 0 1 1 1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Single Precision

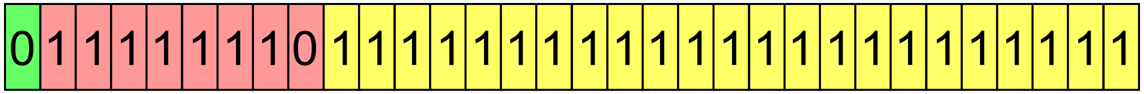
1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Double Precision

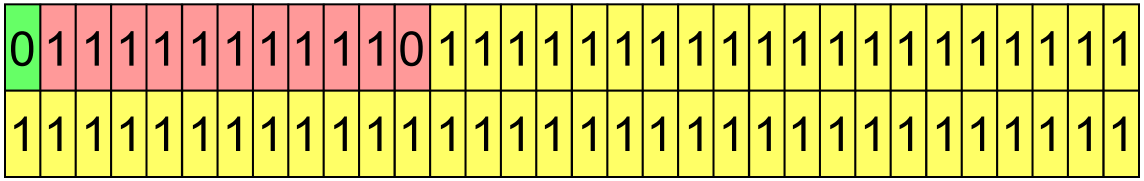
Largest Normalized Float

- What is the Largest normalized float?
- Solution for Single Precision:



0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

  - Exponent – bias = 254 – 127 = 127 (largest exponent for SP)
  - Significand = $(1.111 \ldots 1)_2$ = almost 2
  - Value in decimal $\approx 2 \times 2^{127} \approx 2^{128} \approx 3.4028 \ldots \times 10^{38}$
- Solution for Double Precision:



0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

  - Value in decimal $\approx 2 \times 2^{1023} \approx 2^{1024} \approx 1.79769 \ldots \times 10^{308}$
- Overflow: exponent is too large to fit in the exponent field

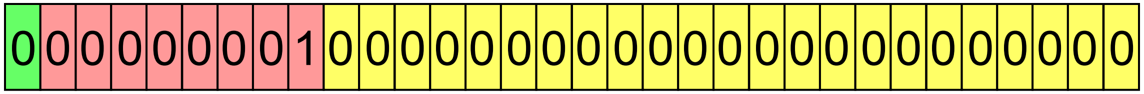
Density of Floating-point Numbers

- How can we represent so many more numbers in floating point than in integer? We don't!
  - The number of unique bit patterns has to be the same as with integers of the same "bitness"
  - There are 8,388,607 single precision numbers in 1.0< x <2.0, but only
  - 8191 in 1023.0< x <1024.0

- absolute precision depends on the magnitude
- some numbers have no exact representation
- approximated using rounding mode (nearest)

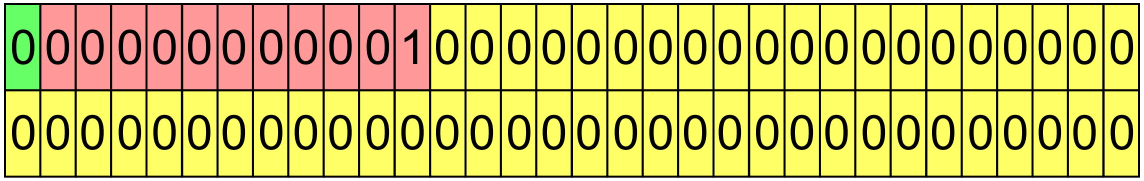
Smallest Normalized Float

- What is the smallest (in absolute value) normalized float?
- Solution for Single Precision:



00000000100000000000000000000000

  - Exponent – bias = 1 – 127 = –126 (smallest exponent for SP)
  - Significand = $(1.000 \ldots 0)_2 = 1$
  - Value in decimal = $1 \times 2^{-126} = 1.17549 \ldots \times 10^{-38}$
- Solution for Double Precision:



00000000000010000000000000000000
00000000000000000000000000000000

  - Value in decimal = $1 \times 2^{-1022} = 2.22507 \ldots \times 10^{-308}$
- Underflow: exponent is too small to fit in exponent field

Zero, Infinity, and NaN

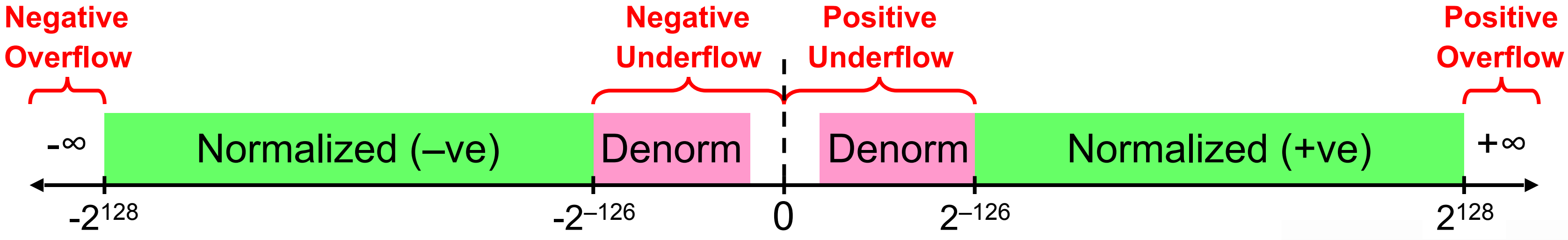
- Zero
  - Exponent field $E = 0$ and fraction $F = 0$
  - +0 and –0 are possible according to sign bit S
- Infinity
  - Infinity is a special value represented with maximum E and $F = 0$
    - For single precision with 8-bit exponent: maximum $E = 255$
    - For double precision with 11-bit exponent: maximum $E = 2047$
  - Infinity can result from overflow or division by zero
  - $+\infty$ and $-\infty$ are possible according to sign bit S
- NaN (Not a Number)
  - NaN is a special value represented with maximum E and $F \neq 0$
  - Result from exceptional situations, such as 0/0 or sqrt(negative)
  - Operation on a NaN results is NaN: Op(X, NaN) = NaN

Denormalized Numbers

- IEEE standard uses denormalized numbers to …
  - Fill the gap between 0 and the smallest normalized float
  - Provide gradual underflow to zero
- Denormalized: exponent field E is 0 and fraction $F \neq 0$
  - Implicit 1. before the fraction now becomes 0. (not normalized)
- Value of denormalized number (S, 0, F)

Single precision: $(-1)^{S} \times (0.F)_2 \times 2^{-126}$

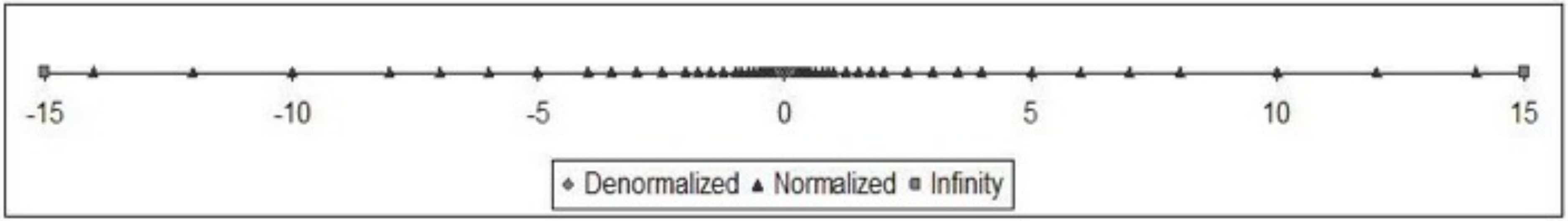
Double precision: $(-1)^{S} \times (0.F)_2 \times 2^{-1022}$

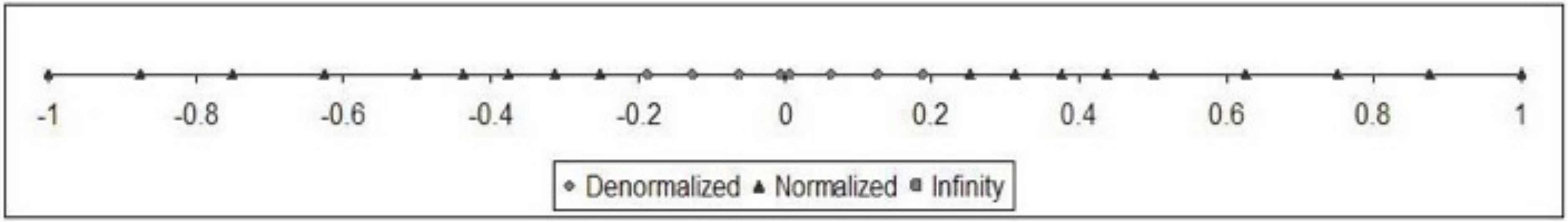


Filling the Gaps…

- hypothetical 6-bit floating point representation:

  E = 3, F = 2





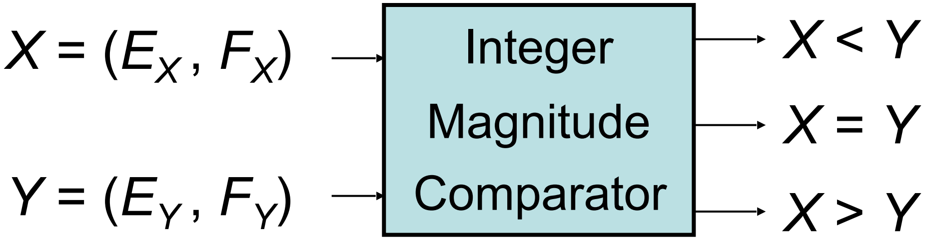
Summary of IEEE 754 Encoding

Single-Precision	Exponent = 8	Fraction = 23	Value
Normalized Number	1 to 254	Anything	$\pm (1.F)_2 \times 2^{E-127}$
Denormalized Number	0	nonzero	$\pm (0.F)_2 \times 2^{-126}$
Zero	0	0	± 0
Infinity	255	0	± ∞
NaN	255	nonzero	NaN

Double-Precision	Exponent = 11	Fraction = 52	Value
Normalized Number	1 to 2046	Anything	$\pm (1.F)_2 \times 2^{E-1023}$
Denormalized Number	0	nonzero	$\pm (0.F)_2 \times 2^{-1022}$
Zero	0	0	± 0
Infinity	2047	0	± ∞
NaN	2047	nonzero	NaN

Floating-Point Comparison

- IEEE 754 floating point numbers are ordered
  - Because exponent uses a biased representation …
    - Exponent value and its binary representation have same ordering
  - Placing exponent before the fraction field orders the magnitude
    - Larger exponent ⇒ larger magnitude
    - For equal exponents, Larger fraction ⇒ larger magnitude
    - $0 < (0.F)_2 \times 2^{E_{min}} < (1.F)_2 \times 2^{E-Bias} < \infty \; (E_{min} = 1 - Bias)$
  - Because sign bit is most significant ⇒ quick test of signed <
- Integer comparator can compare magnitudes



$X = (E_X , F_X)$

$Y = (E_Y , F_Y)$

Integer Magnitude Comparator

$X < Y$

$X = Y$

$X > Y$

Floating Point Addition Example

- Consider Adding (Single-Precision Floating-Point):

$$+\ 1.11100100000000000000010_2 \times 2^{4}$$

$$+\ 1.10000000000000110000101_2 \times 2^{2}$$

- Cannot add significands … Why?
  - Because exponents are not equal
- How to make exponents equal?
  - Shift the significand of the lesser exponent right
  - Difference between the two exponents = 4 – 2 = 2
  - So, shift right second number by 2 bits and increment exponent

$$1.10000000000000110000101_2 \times 2^{2}$$

$$= 0.01100000000000001100001\ 01_2 \times 2^{4}$$

Floating-Point Addition – cont'd

- Now, ADD the Significands:

$+\ 1.11100100000000000000010 \quad \times 2^{4}$

$+\ 1.10000000000000110000101 \quad \times 2^{2}$

$+\ 1.11100100000000000000010 \quad \times 2^{4}$

$+\ 0.01100000000000001100001\ 01 \times 2^{4}$ (shift right)

$+10.01000100000000001100011\ 01 \times 2^{4}$ (result)

- Addition produces a carry bit, result is NOT normalized
- Normalize Result (shift right and increment exponent):

$+\ 10.01000100000000001100011 \quad 01 \times 2^{4}$

$=\ +\ 1.00100010000000000110001 \quad 101 \times 2^{5}$

Rounding

- Single-precision requires only 23 fraction bits
- However, Normalized result can contain additional bits

1.00100010000000000110001 | 1 01 × 2^5

Round Bit: **R = 1** *Sticky Bit:* **S = 1**

- Two extra bits are needed for rounding
  - Round bit: appears just after the normalized result
  - Sticky bit: appears after the round bit (OR of all additional bits)
- Since **RS = 11**, increment fraction to round to nearest

1.00100010000000000110001 × 2^5

+1

1.00100010000000000110010 × 2^5 (Rounded)

Rounding to Nearest Even

- Normalized result has the form: 1. $f_1\ f_2\ \ldots\ f_l$ **R S**
  - The round bit **R** appears after the last fraction bit f_l
  - The sticky bit **S** is the OR of all remaining additional bits
- Round to Nearest Even: default rounding mode
- Four cases for **RS**:
  - **RS = 00 →** Result is Exact, no need for rounding
  - **RS = 01 →** **Truncate** result by discarding **RS**
  - **RS = 11 →** **Increment** result: ADD 1 to last fraction bit
  - **RS = 10 →** Tie Case (either truncate or increment result)
    - **Check Last fraction bit f_l** (f_{23} for single-precision or f_{52} for double)
    - **If f_l is 0 then truncate result to keep fraction even**

Additional Rounding Modes

- IEEE 754 standard specifies four rounding modes:

1. Round to Nearest Even: described in previous slide
2. Round toward +Infinity: result is rounded up

   Increment result if sign is positive and **R** or **S** = **1**

3. Round toward -Infinity: result is rounded down

   Increment result if sign is negative and **R** or **S** = **1**

4. Round toward 0: always truncate result

- Rounding or Incrementing result might generate a carry
  - This occurs when all fraction bits are **1**
  - Re-Normalize after Rounding step is required only in this case

Example on Rounding

- Round following result using IEEE 754 rounding modes:

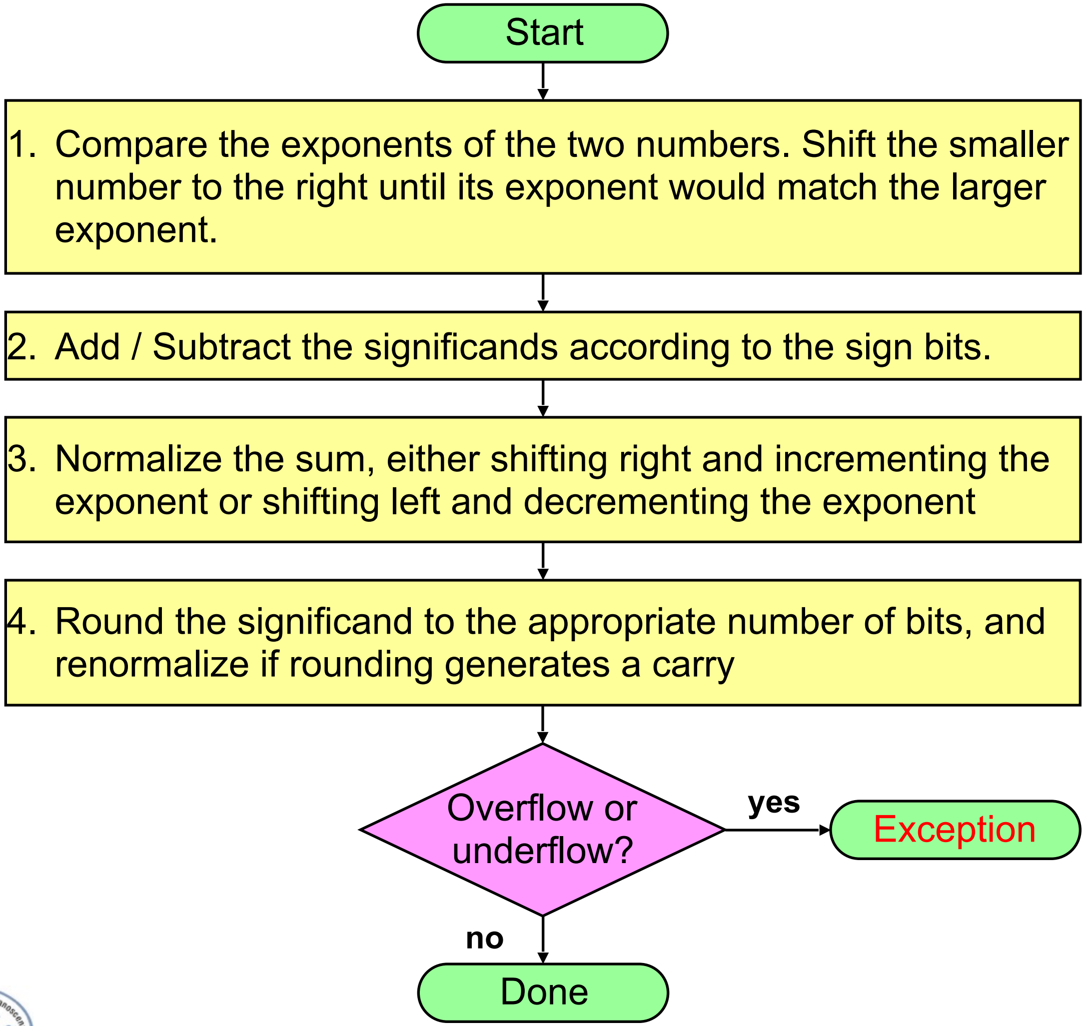
  $-1.11111111111111111111111 \; 1 \; 0 \times 2^{-7}$

  Round Bit *Sticky Bit*

- Round to Nearest Even:
  - Increment result since **RS = 10 and f_{23} = 1**
  - Incremented result: $-10.00000000000000000000000 \times 2^{-7}$
  - Renormalize and increment exponent (because of carry)
  - Final rounded result: $-1.00000000000000000000000 \times 2^{-6}$
- Round towards +∞: Truncate result since negative
  - Truncated Result: $-1.11111111111111111111111 \times 2^{-7}$
- Round towards –∞: Increment since negative and **R = 1**
  - Final rounded result: $-1.00000000000000000000000 \times 2^{-6}$

Truncate always

Floating Point Addition /



Start

1. Compare the exponents of the two numbers. Shift the smaller number to the right until its exponent would match the larger exponent.
2. Add / Subtract the significands according to the sign bits.
3. Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent
4. Round the significand to the appropriate number of bits, and renormalize if rounding generates a carry

Overflow or underflow?

Done

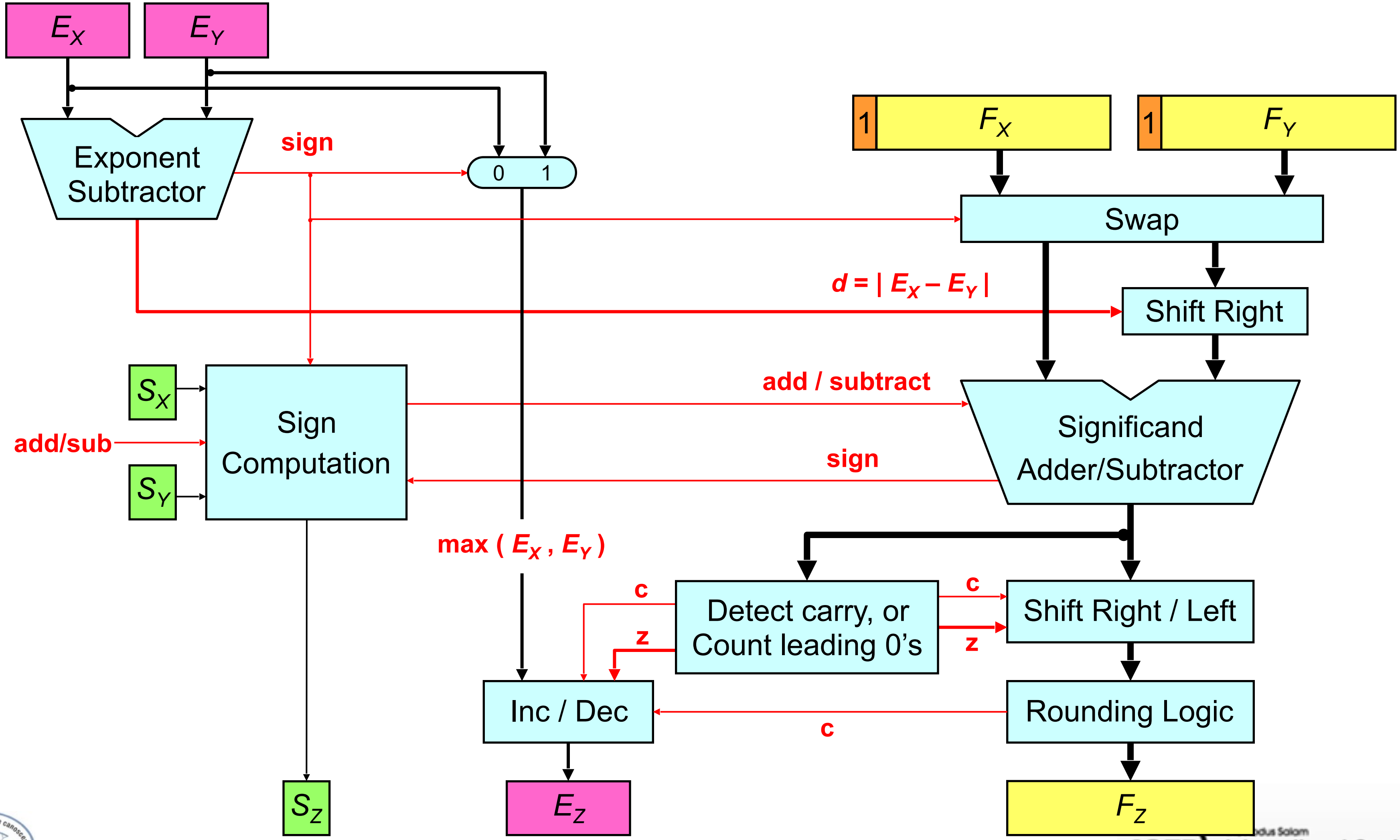
Shift significand right by
$d = | E_X - E_Y |$

Add significands when signs of X and Y are identical, Subtract when different
$X - Y$ becomes $X + (-Y)$

Normalization shifts right by 1 if there is a carry, or shifts left by the number of leading zeros in the case of subtraction

Rounding either truncates fraction, or adds a 1 to least significant fraction bit

Floating Point Adder Block



Advantages of IEEE 754 Standard

- Used predominantly by the industry
- Encoding of exponent and fraction simplifies comparison
  - Integer comparator used to compare magnitude of FP numbers
- Includes special exceptional values: NaN and ±∞
  - Special rules are used such as:
    - 0/0 is NaN, *sqrt*(–1) is NaN, 1/0 is ∞, and 1/∞ is 0
  - Computation may continue in the face of exceptional conditions
- Denormalized numbers to fill the gap
  - Between smallest normalized number $1.0 \times 2^{E_{min}}$ and zero
  - Denormalized numbers , values $0.F \times 2^{E_{min}}$, are closer to zero
  - Gradual underflow to zero

Floating Point Complexities

- Operations are somewhat more complicated
- In addition to overflow we can have underflow
- Accuracy can be a big problem
  - Extra bits to maintain precision: guard, round, and sticky
  - Four rounding modes
  - Division by zero yields Infinity
  - Zero divide by zero yields Not-a-Number
  - Other complexities
- Implementing the standard can be tricky
  - See text for description of 80x86 and Pentium bug!
- Not using the standard can be even worse

Accuracy can be a Big Problem

Value1	Value2	Value3	Value4	Sum
1.0E+30	-1.0E+30	9.5	-2.3	7.2
1.0E+30	9.5	-1.0E+30	-2.3	-2.3
1.0E+30	9.5	-2.3	-1.0E+30	0

- Adding double-precision floating-point numbers (Excel)
- Floating-Point addition is NOT associative
- Produces different sums for the same data values
- Rounding errors when the difference in exponent is large