DATA REPRESENTATION PART 2: REAL NUMBERS

CCICOMP

OVERVIEW

- Perform conversion between binary and decimal fractional numbers
- Represent real numbers using floating point notation
- Explain the concepts of overflow, truncation and precision in relation to real numbers

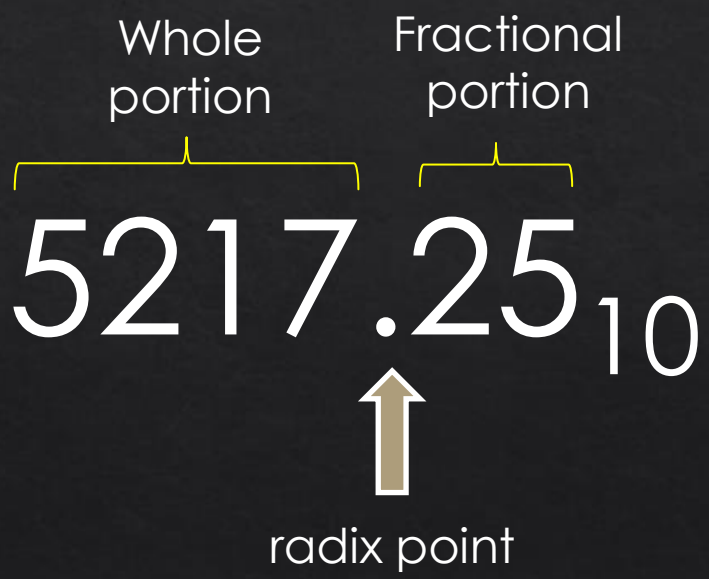
RECALL: CPU DATA TYPES

- The CPUs of most modern computers can represent and process at least the following primitive data types:

- The arrangement and interpretation of bits are usually different for each data type.
- Format for each data type balances compactness, range, accuracy, ease of manipulation, and standardization.

REAL NUMBERS

- Real numbers contain both whole and fractional components, with the fractional portion represented by digits to the right of the radix point.



REAL NUMBERS

- When working with a positional number system, the fractional digits are weighted by negative powers of the base

5217.25_{10}

$5 \times 10^{-2} = 5 \times 1 / 10 / 10 = .05$

$2 \times 10^{-1} = 2 \times 1 / 10 = .2$

$7 \times 10^{0} = 7 \times 1 = 7$

$1 \times 10^{1} = 1 \times 1 \times 10 = 10$

$2 \times 10^{2} = 2 \times 1 \times 10 \times 10 = 200$

$5 \times 10^{3} = 5 \times 1 \times 10 \times 10 \times 10 = 5000$

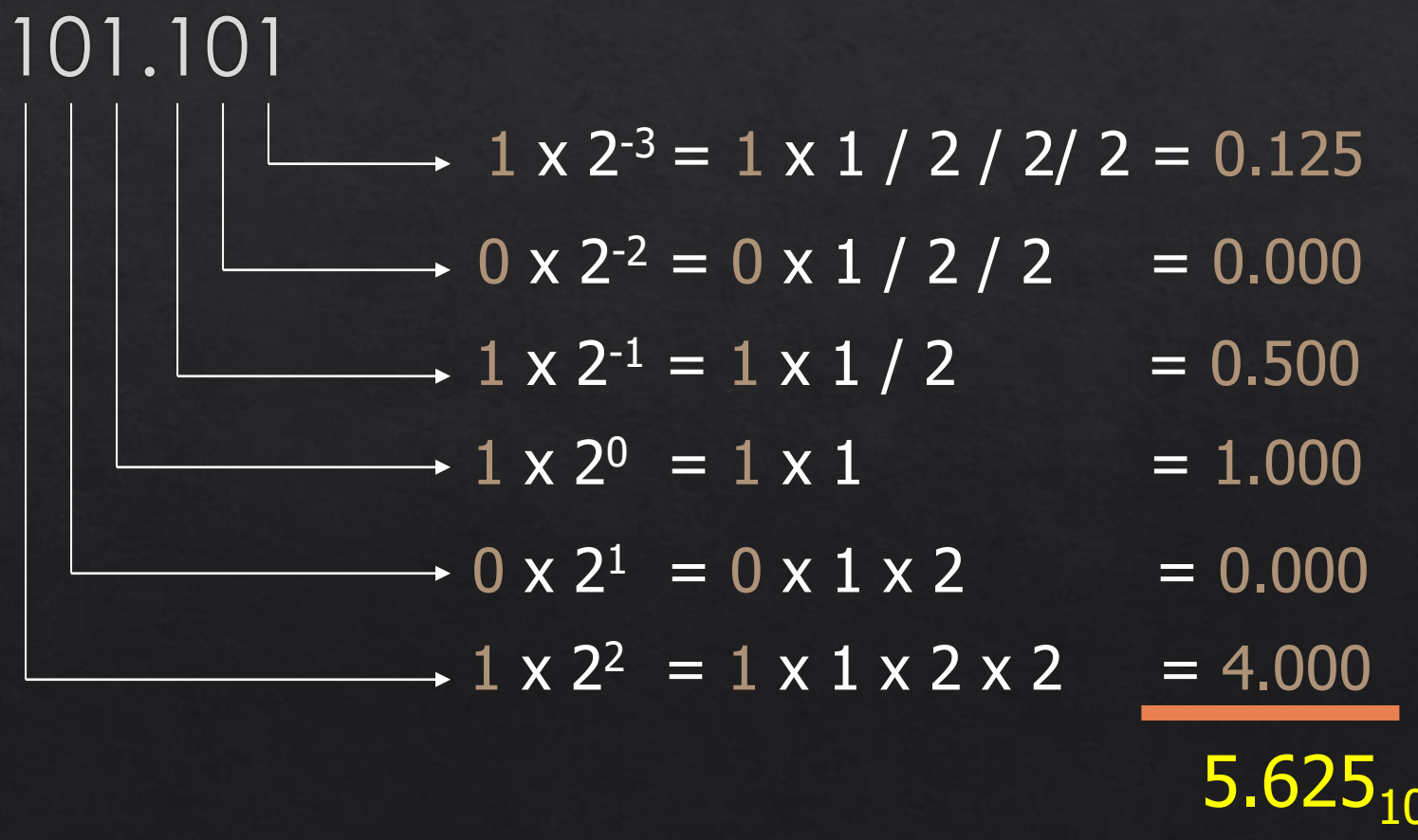
5217.25_{10}

BINARY TO DECIMAL CONVERSION (WITH FRACTIONS)

- Determine each position weight by raising 2 to number of positions left (+) or right (-) of decimal point, multiply each binary digit by its weight, then add all the products

EXAMPLE 1:

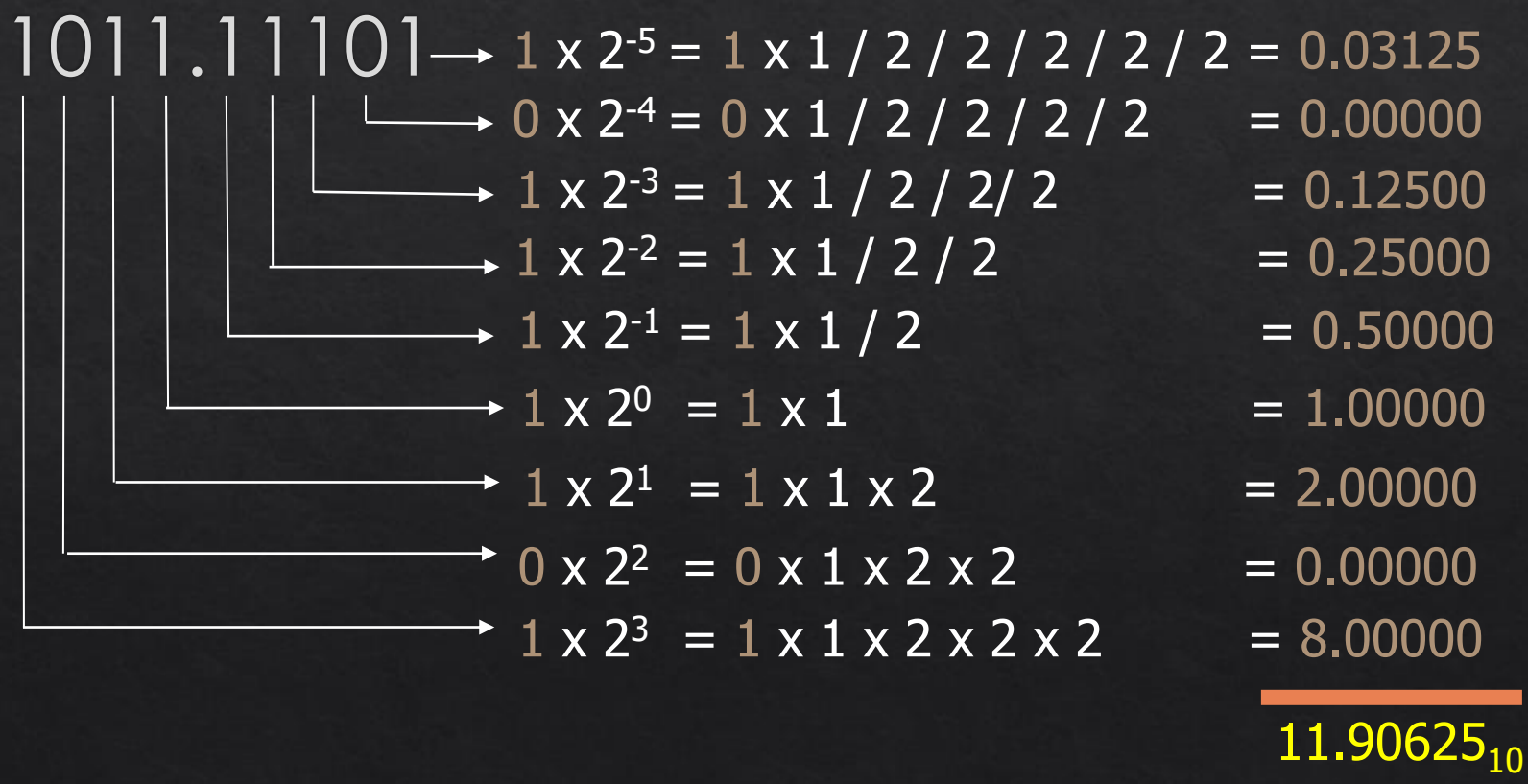
$101.101_2 = ?_{10}$



BINARY TO DECIMAL CONVERSION (WITH FRACTIONS)

EXAMPLE 2:

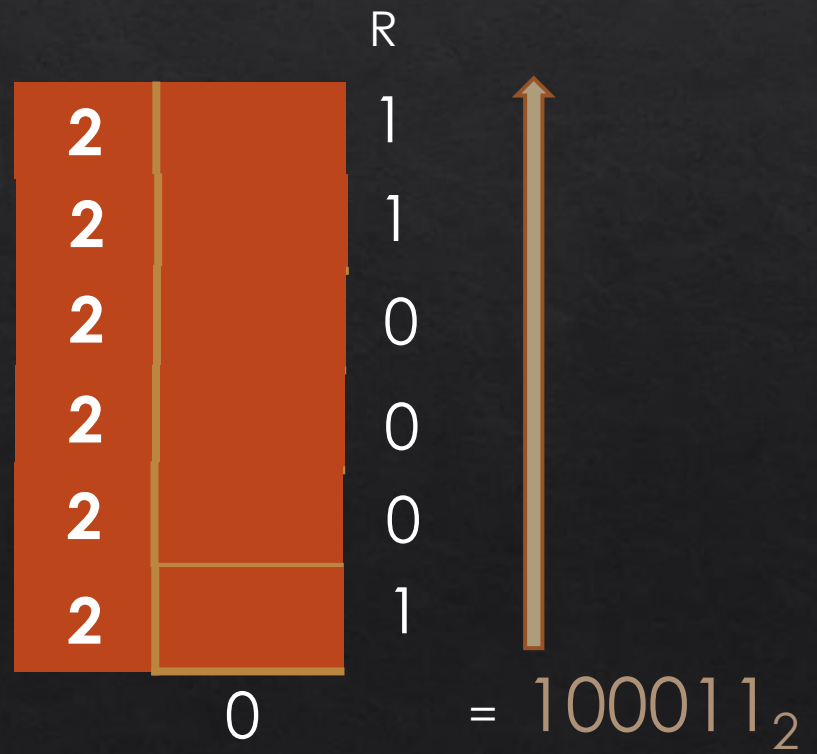
$1011.01101_2 = ?_{10}$



DECIMAL TO BINARY CONVERSION (INTEGER PART)

- Continuously divide the integer portion of the number by 2.
- For each division note the remainder (which is 1 or 0) and continue the division until only 0 is left.
- read binary numbers starting from the last remainder to the first as the left to right bits.

EXAMPLE : $35_{10} = ?_2$



DECIMAL TO BINARY CONVERSION (FRACTIONAL PART)

- Continuously multiply only the fractional portion of the number by 2 until the desired number of bits is met or a 0 fractional part is left. For each multiplication, note the resulting integer portion (1 or 0).
- Take binary numbers starting from the first noted integer to the last as the left to right bits after the binary point.

EXAMPLE 1:

$0.875_{10} = (?)_2$

$2 \times 0.875 = 1.75$

$2 \times 0.75 = 1.5$

$2 \times 0.5 = 1.0$

$= 0.111_2$

EXAMPLE 2:

$0.90625_{10} = (?)_2$

$2 \times 0.90625 = 1.8125$

$2 \times 0.8125 = 1.625$

$2 \times 0.625 = 1.25$

$2 \times 0.25 = 0.5$

$2 \times 0.5 = 1.0$

$= 0.11101_2$

FRACTIONAL EXTENSION

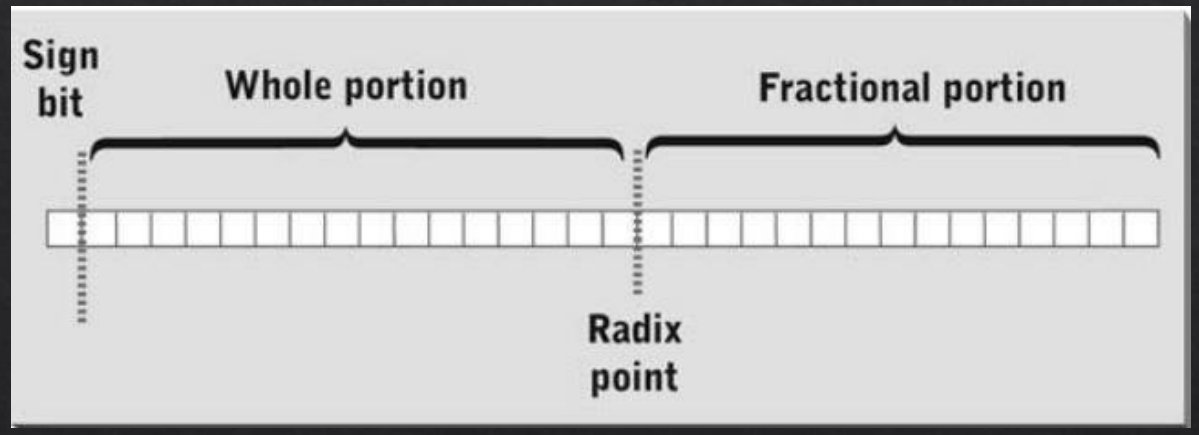
- The fractional portion of a real number can be extended by any number 0's to the right without changing its original value
- Examples:

$5217.25_{10} = 5217.2500_{10} = 5217.2500000000_{10}$

$101.101_2 = 101.10100_2 = 101.10100000000000_2$

FIXED POINT NOTATION

- Simple way to represent real numbers using a given data size is to define a format in which a fixed-length portion of the bit string holds the whole and fractional portions



- Fast and easy to convert / interpret but …
  - We lack range in representing very large numbers because of limited whole portion bits
  - We lack precision in representing very small values because of limited fractional portion bits

How do we work around this limitation?

RECALL: SCIENTIFIC NOTATION

- To express very large or very small numerical values in a more compact form, scientists and engineers use the scientific notation

$$M \times 10^{E}$$

Where

- M is the mantissa
- E is the exponent
- 10 is the base of the number system (decimal in this case)

10^{E} is also known as scaling factor

Value	Scientific Notation
724.1301	7.241301×10^{2}
.000018132	1.8132×10^{-5}
26310.3522	**2.63103522×10^{4}**
$.573628$	**5.73628×10^{-1}**

FLOATING POINT NOTATION

- Applying this technique to real number representation, we deal with the tradeoff between range and precision associated with limited space using a ‘floating’ radix point instead of a fixed point
  - To represent extremely small values, move the radix point far to the left (high fractional precision).
    Ex. 0.0000000013526473
  - To represent very large values, move the radix point far to the right (high range).
    Ex. 13526473000000000.0
- Values can be very large or very small (precise), but not both at the same time.

FLOATING POINT NOTATION

- Floating point notation for binary real numbers is based on scientific notation using base 2

  $$\text{Value} = \text{mantissa} \times 2^{\text{exponent}}$$

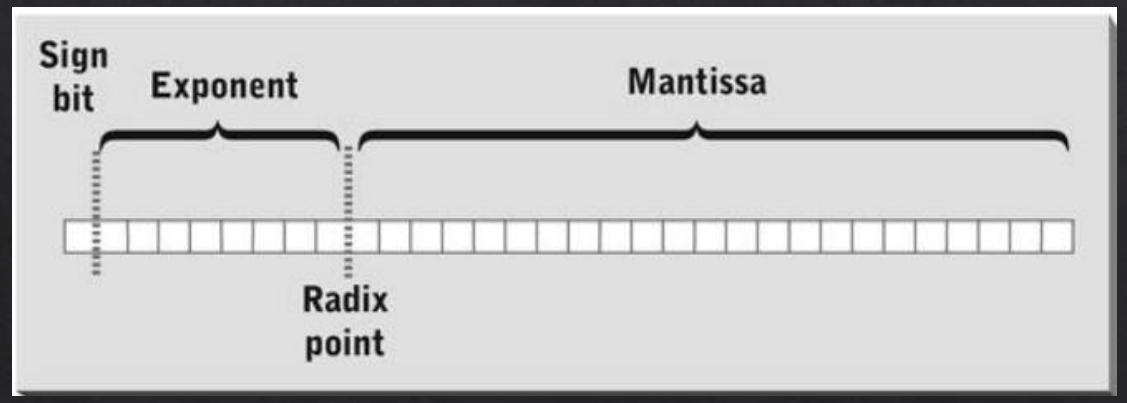
  - Mantissa holds the bits that are interpreted to derive the real number's digits.
  - Exponent value indicates the binary point's position
- Current format used by processors is defined by the Institute of Electrical and Electronics Engineers and is defined as the IEEE 745 standard
  - *binary32*—32-bit format for base 2 values (aka *IEEE single precision)*
  - *binary64*—64-bit format for base 2 values (aka *IEEE double precision)*
  - *binary128*—128-bit format for base 2 values (aka *IEEE quad precision)*

IEEE FLOATING POINT NOTATION (BINARY32)

$$(-1)^{S} \times 1.M \times 2^{E'-127}$$

- Sign bit applies to the mantissa (e.g. 1 if mantissa is negative, 0 if mantissa is positive)
- 8-bit exponent is coded using bias 127 (i.e. 127 + value of exponent)
- The fractional portion of the mantissa is coded as a 23-bit sequence. It is assumed to be preceded by a binary 1 and the binary point

DECIMAL TO BINARY FLOAT – EXAMPLE 1



What is -12.125 in IEE single precision format?

11000000101000010000000000000000

In hexadecimal format : **C1420000h**

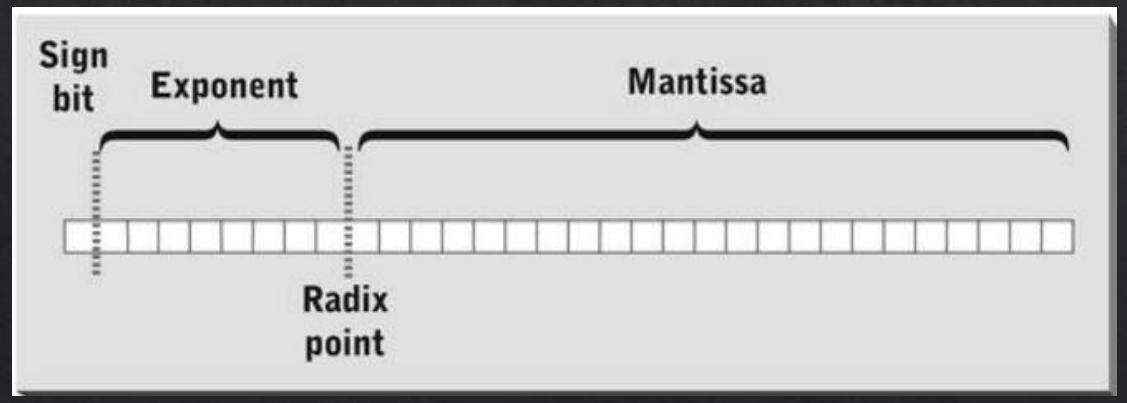
- Number is negative □□ sign bit = 1
- Whole number 12 = 1100
- Fraction 0.125 = .001
- Combine whole and fractional parts:

  1100.001
- Express value in binary scientific notation, take the fractional portion and express as a 23-bit binary value

  1100.001 □□ 1.100001×2^3
- Normalize exponent to 127 and convert to 8-bit binary

  127 + 3 = 130 □□ 1000 0010

DECIMAL TO BINARY FLOAT – EXAMPLE 2



What is 0.15625 in IEE single precision format?

00111111001000000000000000000000

In hexadecimal format : **3E200000h**

- Number is positive □□ sign bit = 0
- Whole number 0 = 0
- Fraction 0.15625 = .00101
- Combine whole and fractional parts:

  0.00101

- Express value in binary scientific notation, take the fractional portion and express as a 23-bit binary value

  0.00101 □□ 1.**01** x 2^{-3}

- Normalize exponent to 127 and convert to 8-bit binary

  127 - 3 = 124 □□ 0111 1100

BINARY FLOAT TO DECIMAL - EXAMPLE 1

What is C6812AC0h (IEE single precision) in decimal?

1 10001101 00000010010101011000000

Mantissa: (Append to 1.xxxx…. then scale by exponent)

$1.00000010010101011000000 \times 2^{14}$

$10000010010101.011000000 = 1x2^{14} + 1x2^{7} + 1x2^{4} + 1x2^{2} + 1x2^{0} + 1x2^{-2} + 1x2^{-3}$

$= 16384 + 128 + 16 + 4 + 1 + 0.25 + 0.125$

$= 16533.375$

Exponent: 10001101

$= 1x2^{7} + 1x2^{3} + 1x2^{2} + 1x2^{0}$

$= 128 + 8 + 4 + 1 = 141$

Denormalize 141-127 = 14

Sign bit : 1

Number is negative

Final value (including the sign)

-16533.375

BINARY FLOAT TO DECIMAL - EXAMPLE 2

What is 3EE80000h (IEE single precision) in decimal?

0 01111101 11010000000000000000000

Mantissa: (Append to 1.xxxx…. then scale by exponent)

$01.11010000000000000000000 \times 2^{-2}$

$0.01110100000000000000000 = 1x2^{-2} + 1x2^{-3} + 1x2^{-4} + 1x2^{-6}$

$= 0.25 + 0.125 + 0.0625 + 0.015625$

$= 0.453125$

Exponent: 01111101

$= 1x2^6+1x2^5+1x2^4 + 1x2^3+ 1x2^2 + 1x2^0$

$= 64 + 32+ 16 + 8 + 4 + 1 = 125$

Denormalize 125-127 = -2

Sign bit : 0

Number is positive

Final value (including the sign)

0.453125

SPECIAL FLOAT VALUES (SINGLE PRECISION)

Sign Bit	E’	Mantissa	Value
0	0000 0000	000 0000 0000 0000 0000 0000	+0 (Positive Zero)
1	0000 0000	000 0000 0000 0000 0000 0000	-0 (Negative Zero)
0/1	0000 0000	<>0	Denormalized (mantissa treated as 0.xxxxxx)
0	1111 1111	000 0000 0000 0000 0000 0000	+ Infinity
1	1111 1111	000 0000 0000 0000 0000 0000	- Infinity
x	1111 1111	0xx xxxx xxxx xxxx xxxx xxxx	sNaN
x	1111 1111	1xx xxxx xxxx xxxx xxxx xxxx	qNaN

RANGE, OVERFLOW AND UNDERFLOW

- Number of bits in a floating-point string and the formats of the mantissa and exponent impose limits on the range of values that can be represented.
  - The number of mantissa digits determines the number of significant (nonzero) digits in the largest and smallest values that can be represented.
  - The number of exponent digits in the exponent determines the number of possible bit positions to the right or left of the binary point.
- Floating point range issues manifest in the exponent
  - Overflow occurs when large numbers need exponents that occupy more bits than the exponent allows after normalization (e.g. $> 2^{127}$ for single precision)
  - Underflow occurs when very small numbers need exponents that result to less than 1 after normalization (e.g. $< 2^{-126}$ for single precision)

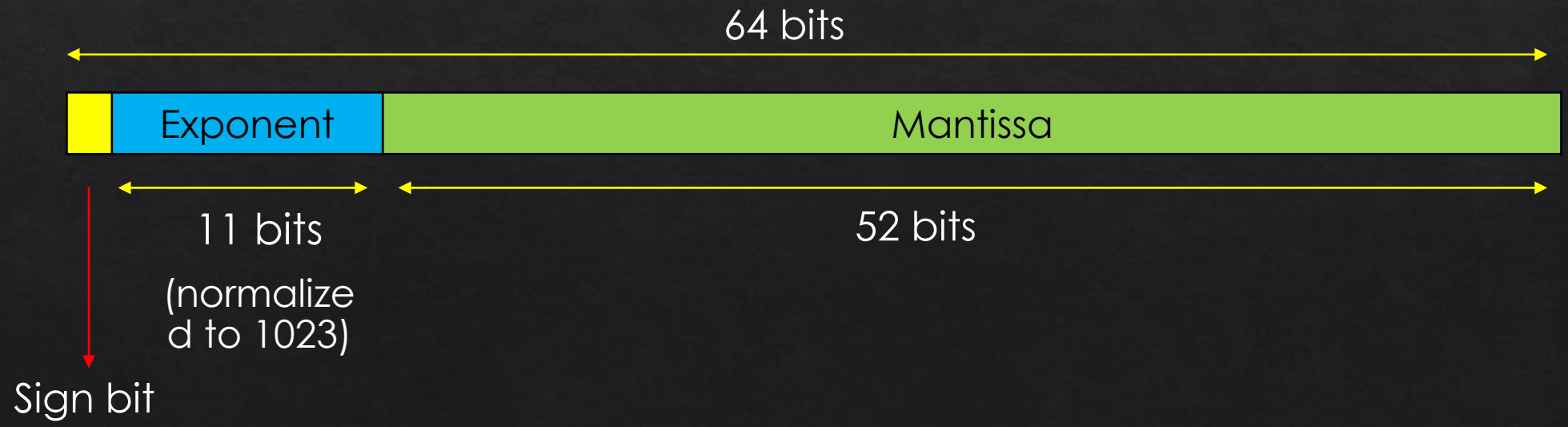
TRUNCATION

- Floating-point notation trades off accuracy for range
- For decimal values that contain more mantissa digits that can fit in the encoding format, only an approximate value is stored
- Numeric value is stored in the mantissa, starting with its most significant bit, until all available bits are used. The remaining bits are discarded (truncation)
- Ex: Single precision format for the value of 1/3 (0.333333333…….):
  - 0.3333333…. = 0.01010101010101010101010101010101….
  - Stored as: 0 01111101 01010101010101010101011 = 0.3333334

Not the same value, but close enough

WHAT IF WE NEED MORE ACCURACY?

- Increasing data width will allow more range and precision leading to better accuracy
- E.g. We can use IEEE double precision float for even larger / precise numbers



DECIMAL TO BINARY DOUBLE FLOAT EXAMPLE

What is -12.125 in IEE double precision format? **C058000000000000h**

1 10000000010 1000010000000000000000000000000000000000000000000000

- Number is negative □□ sign bit = 1
- Whole number 12 = 1100
- Fraction 0.125 = .001
- Combine whole and fractional parts: 1100.001
- Express value in binary scientific notation, take the fractional portion and express as a 52-bit binary value

  1100.001 □□ 1.100001×2^3
- Normalize exponent to 1023 and convert to 11-bit binary

  1023 + 3 = 1026 □□ 100 0000 0010

BINARY DOUBLE FLOAT TO DECIMAL EXAMPLE

What is 3FBD000000000000h (IEE double precision) in decimal?

0 01111111011 1101000000000000000000000000000000000000000000000000

Mantissa: (Append to 1.xxxx…. then scale by exponent)

$0001.1101000000000000000000 \times 2^{-4}$

$0.0001101000000000000000000 = 1x2^{-4} + 1x2^{-5} + 1x2^{-6} + 1x2^{-8}$

$= 0.0625 + +0.03125 + 0.015625 + 0.00390625$

$= 0.11328125$

Exponent: 01111111011

$= 1x2^9+1x2^8+1x2^7+1x2^6+1x2^5+1x2^4+1x2^3+1x2^1+1x2^0$

$= 512 + 256 + 128 + 64 + 32 + 16 + 8 + 2 + 1 = 1019$

Denormalize 1019 – 1023 = -4

Sign bit : 0

Number is positive

Final value (including the sign)

0.11328125

DOUBLE FLOAT ACCURACY

- Double precision format for the value of 1/3 (0.333333333……):
  - 0.3333333…. = 0.0101010101010101010101010101010101010101010….
  - Stored as:

    0 01111111101001010101010101010101010101010101010101010101010101

    = 0.3333333333333333
- For the same numerical value
  - In single precision, stored value is 0.3333334
  - In double precision, stored value is 0.3333333333333333 → Still not the same value, but closer to the actual number

SUMMARY

- Fractional binary values are converted to decimal by multiplying the binary digits to the right of the binary point by negative powers of 2
- To represent fractional decimal values to binary, perform continuous division on the fractional values and track the resulting whole number portions of the results
- Fixed point binary representation for real numbers is not practical because it lacks range for large numbers and precision for very small values
- Floating point binary representation is based on the scientific notation but uses powers of 2 as its scaling factor

SUMMARY

- In the IEEE 745 standard defines a floating point representation scheme where:

	Single Precision	Double Precision
Size	32 bits	64 bits
Sign	1 bit (0 = positive, 1 = negative)	
Exponent	8 bits normalized to 127	11 bits normalized to 1023
Mantissa	23 bits	52 bits

- Floating point representation schemes trade accuracy for range. A value stored is only a close approximate if the required number of bits to represent the exact mantissa is truncated due to exceeding the capacity of the data type
- A floating point overflow occurs when a value is too large to be represented by the largest exponent possible
- A floating point underflow occurs when a value is too small to be represented by the smallest exponent possible