School of Engineering and Computer Science

Numerical Applied Mathematics

(Floating Point Numbers)

Kamel ATTAR
attar.kamel@gmail.com

Week **#12** ✦ 12/JAN/2024 ✦

1 Fraction Number
- Fractional Decimal Number
- Fractional Binary Number
- Converting Binary Fraction to a Decimal
- Converting Decimal to a Binary Fraction

2 Floating point representation
- Normalized binary scientific notation
- IEEE Floating Point Standard
  - IEEE single precision
  - IEEE doule precision

3 Conversions
- Decimal to Floating-Point Conversions
- Float to Decimal Conversion

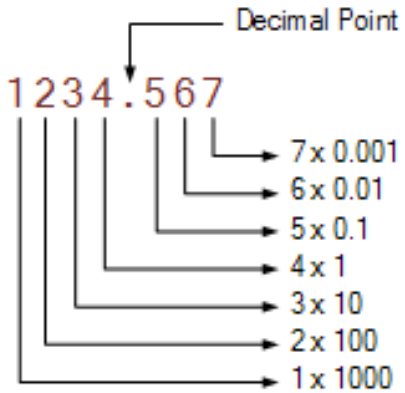
4 Special Values in IEEE 754-1985 Standard (32 Bits)
- Representation of Zero
- Representation of Infinity
- Representation of Non Numbers

Largest and Smallest Positive Floating Point Numbers: Subnormal Numbers

1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

- The decimal numbering system uses the concept of positional or relative weighting values producing a positional notation, where each digit represents a different weighted value depending on the position occupied either side of the decimal point.
- Then the value of any decimal number will be equal to the sum of its digits multiplied by their respective weights.
- Here in this decimal number example, the digit immediately to the right of the decimal point (number **5**) is worth one tenth (**1**/**10** or **0.1**) of the digit immediately to the left of the decimal point (number **4**) which as a multiplication value of one (**1**).
- Thus as we move through the number from left-to-right, each subsequent digit will be one tenth the value of the digit immediately to its left position, and so on.



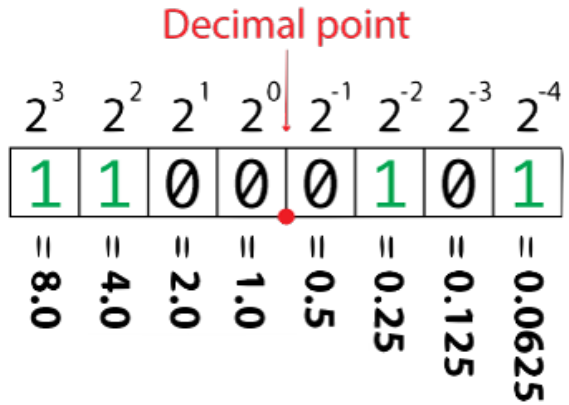
- For our example: $N = 1234.567_{10}$ in the weighted decimal format this will be equal to:

$$\begin{aligned} 1234.567_{10} &= (1 \times 10^3) + (2 \times 10^2) + (3 \times 10^1) + (4 \times 10^0) + (5 \times 10^{-1}) + (6 \times 10^{-2}) + (7 \times 10^{-3}) \\ &= (1 \times 1000) + (2 \cdot 100) + (3 \cdot 10) + (4 \cdot 1) + (5 \cdot 0.1) + (6 \cdot 0.01) + (7 \times 0.001) \\ &= 1000 + 200 + 30 + 4 + 0.5 + 0.06 + 0.007\ . \end{aligned}$$

1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

- We can use the same idea of positional notation where each digit represents a different weighted value depending upon the position it occupies in the binary numbering system.
- The difference this time is that the binary number system is a positional system, where the different weighted positions of the digits are to the power of **2** (base-**2**) instead of **10**.

- Thus all the fractional digits to the right of the binary point have respective weightings which are negative powers of two, creating binary fractions.
- So for the fractional binary numbers to the right of the binary point, the weight of each digit becomes more negative giving: $\mathbf{2^{-1}}$, $\mathbf{2^{-2}}$, $\mathbf{2^{-3}}$, $\mathbf{2^{-4}}$, and so on as shown.

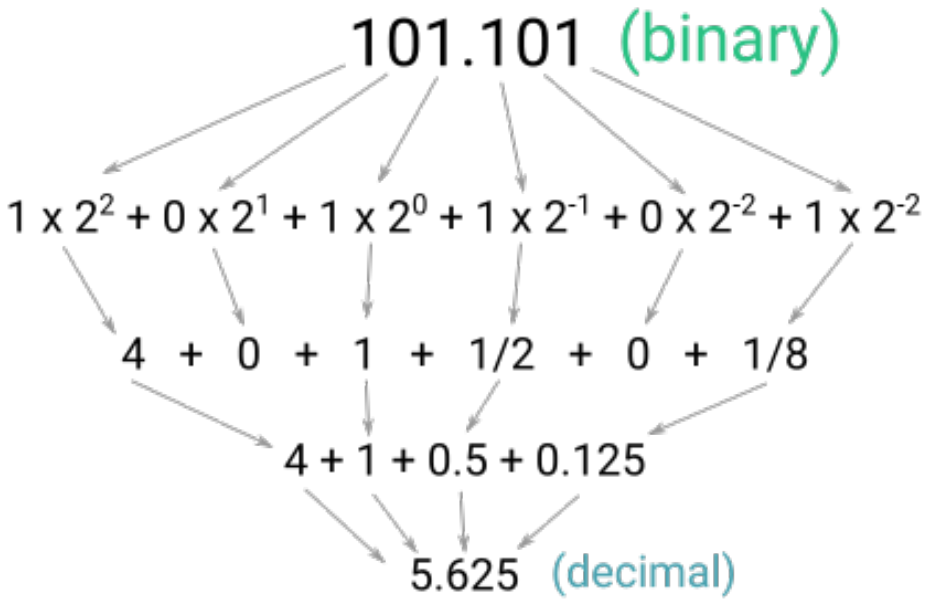


1 Fraction Number
- Fractional Decimal Number
- Fractional Binary Number
- Converting Binary Fraction to a Decimal
- Converting Decimal to a Binary Fraction

2 Floating point representation
- Normalized binary scientific notation
- IEEE Floating Point Standard
  - IEEE single precision
  - IEEE doule precision

3 Conversions
- Decimal to Floating-Point Conversions
- Float to Decimal Conversion

4 Special Values in IEEE 754-1985 Standard (32 Bits)
- Representation of Zero
- Representation of Infinity
- Representation of Non Numbers
- Largest and Smallest Positive Floating Point Numbers:
- Subnormal Numbers



✎ **Exercise 1.** Convert the following binary number to decimal number

a) **1101.0111** b) **0.11** c) **11.001** d) **1011.111**

1 Fraction Number
- Fractional Decimal Number
- Fractional Binary Number
- Converting Binary Fraction to a Decimal
- Converting Decimal to a Binary Fraction

2 Floating point representation
- Normalized binary scientific notation
- IEEE Floating Point Standard
  - IEEE single precision
  - IEEE doule precision

3 Conversions
- Decimal to Floating-Point Conversions
- Float to Decimal Conversion

4 Special Values in IEEE 754-1985 Standard (32 Bits)
- Representation of Zero
- Representation of Infinity
- Representation of Non Numbers
- Largest and Smallest Positive Floating Point Numbers:
- Subnormal Numbers

- The conversion of decimal fractions to binary fractions is achieved using a method similar to that we used for integers. However, this time multiplication is used instead of division with the integers instead of remainders used with the carry digit being the binary equivalent of the fractional part of the decimal number.
- When converting from decimal to binary, the integer (positive sequence right-to-left) part and the fractional (negative sequence from left-to-right) part of the decimal number are calculated separately.
- For the integer part of the number, the binary equivalent is found by successively dividing (known as successive division) the integer part of the decimal number repeatedly by **2** (÷**2**),

noting the remainders in reverse order from the least significant bit (**LSB**) to the most significant bit (**MSB**), until the value becomes "**0**" producing the binary equivalent.

The binary equivalent of $\mathbf{118_{10}}$ is $\mathbf{1110110_2}$

118 (divide by 2)	=	59	plus remainder	0 (LSB)
59 (divide by 2)	=	29	plus remainder	1 (↑)
29 (divide by 2)	=	14	plus remainder	1 (↑)
14 (divide by 2)	=	7	plus remainder	0 (↑)
7 (divide by 2)	=	3	plus remainder	1 (↑)
3 (divide by 2)	=	1	plus remainder	1 (↑)
1 (divide by 2)	=	0	plus remainder	1 (MSB)

- The fractional part of the number is found by successively multiplying (known as successive multiplication) the given fractional part of the decimal number repeatedly by **2** ($\times$**2**), noting the carries in forward order, until the value becomes "**0**" producing the binary equivalent.
- So if the multiplication process produces a product greater than **1**, the carry is a "**1**" and if the multiplication process produces a product less than "**1**", the carry is a "**0**".

The binary fraction equivalent of the decimal fraction: $\mathbf{0.8125_{10}}$ is $\mathbf{0.1101_2}$

0.8125 (multiply by 2)	=	1.625	=	0.625	carry 1 (MSB)
0.625 (multiply by 2)	=	1.25	=	0.25	carry 1 ($\downarrow$)
0.25 (multiply by 2)	=	0.50	=	0.5	carry 0 ($\downarrow$)
0.5 (multiply by 2)	=	1.00	=	0.0	carry 1 (LSB)

- Note also that if the successive multiplication processes does not seem to be heading towards a final zero, the fractional number will have an infinite length or until the equivalent number of bits have been obtained, for example **8**-bits. or **16**-bits, etc. depending on the degree of accuracy required.
- So the binary fraction equivalent of the decimal number $\mathbf{118.8125_{10}}$ si $\mathbf{1110110.1101_2}$

✎ **Exercise 2.** Convert the following decimal numbers to the indicated bases:
a) **7562, 45** to octal b) **1938, 257** to hexadecimal c) **175, 175** to binary

✎ **Exercise 3.** Fill in the missing information in the following table:

Fractional value	Binary representation	Decimal representation
$\frac{1}{8}$	**0, 001**	**0, 125**
$\frac{3}{4}$		
$\frac{25}{16}$		
	10, 1011	
	1, 001	
		5, 875
		3, 1875

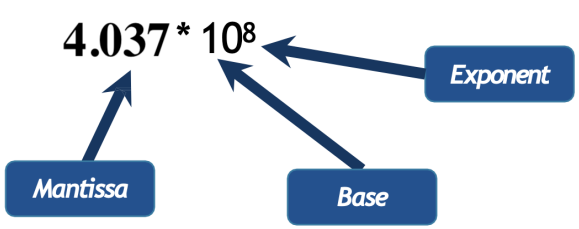
1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

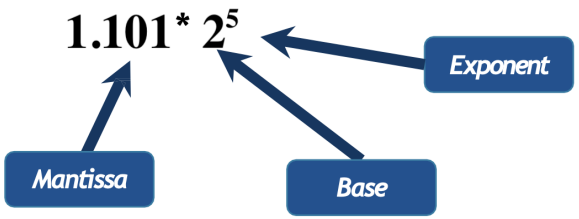
1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

- So far we know how to store **integers whole numbers**. But what if we want to store **real numbers** with **decimal fractions**. Even **27.5** needs another way to represent it.

This method is called floating point representation

- Floating point representation is used when the number contains a decimal fraction.
- Normalized decimal scientific notation has single non-zero digit to the left of the decimal (binary) point.
- Normalized binary scientific notation has single digit to the left of the radix point and there are no leading zeroes. The structure is as follows:





Binary Value	Normalized	Exp
1101.101	1.101101	3
.00101	1.01	-3
1.0001	1.0001	0
10000011.0	1.0000011	7

- Only the mantissa and the exponent are stored. The base is implied (known already) as it is not stored this will save memory capacity

Example

To represent **6.75** into binary floating-point, first we convert the decimal value to binary numbers:

$$\mathbf{6 = 110} \quad \text{and} \quad \underbrace{\begin{array}{lll} \mathbf{0.75 \times 2} & \mathbf{= 1.50} & \mathbf{= 1} \\ \mathbf{0.50 \times 2} & \mathbf{= 1.00} & \mathbf{= 1} \\ \mathbf{0.00 \times 2} & \mathbf{= 0} & \mathbf{= 0} \end{array}}_{\mathbf{0.75=11}} \Longrightarrow \mathbf{6.75 = 110.11}$$

Then the normalized floating point value of **6.75** is $\mathbf{110.11 = +1.1011 \times 2^2}$

Example

To represent **6.75** into binary floating-point, first we convert the decimal value to binary numbers:

$$\mathbf{6 = 110} \quad \text{and} \quad \underbrace{\begin{array}{lll} \mathbf{0.75 \times 2} & \mathbf{= 1.50} & \mathbf{= 1} \\ \mathbf{0.50 \times 2} & \mathbf{= 1.00} & \mathbf{= 1} \\ \mathbf{0.00 \times 2} & \mathbf{= 0} & \mathbf{= 0} \end{array}}_{\mathbf{0.75=11}} \Longrightarrow \mathbf{6.75 = 110.11}$$

Then the normalized floating point value of **6.75** is $\mathbf{110.11 = +1.1011 \times 2^2}$

Example

Find the normalized value of **9.5**.

Example

To represent **6.75** into binary floating-point, first we convert the decimal value to binary numbers:

$$\mathbf{6 = 110} \quad \text{and} \quad \underbrace{\begin{array}{lll} \mathbf{0.75 \times 2} & \mathbf{= 1.50} & \mathbf{= 1} \\ \mathbf{0.50 \times 2} & \mathbf{= 1.00} & \mathbf{= 1} \\ \mathbf{0.00 \times 2} & \mathbf{= 0} & \mathbf{= 0} \end{array}}_{\mathbf{0.75=11}} \Longrightarrow \mathbf{6.75 = 110.11}$$

Then the normalized floating point value of **6.75** is $\mathbf{110.11 = +1.1011 \times 2^2}$

Example

Find the normalized value of **9.5**.

Solution

The normalized floating point value of **9.5** is $\mathbf{1001.1 = +1.0011 \times 2^3}$

1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

- The IEEE (Institute of Electrical and Electronic Engineers) is an international organization that has designed specific binary formats for storing floating point numbers.
- The IEEE defines two different formats with different precisions: single and double precision. Single precision is used by float variables in C and double precision is used by double variables.
- Intel's math coprocessor also uses a third, higher precision called extended precision. In fact, all data in the coprocessor itself is in this precision. When it is stored in memory from the coprocessor it is converted to either single or double precision automatically.
- There are **4** main different sizes of floating point numbers **16**, **32**, **64** and **128** bit. Sometimes referred to as Half, Single, Double and Quadruple precision

Floating point format	Total bits	Sign bits	Biased Exponent bits	Fraction bits
Half precision	**16**	**1**	**5**	**10**
Single precision	**32**	**1**	**8**	**23**
Double precision	**64**	**1**	**11**	**52**

Single precision		
Sign	Biased Exponent	Mantissa
S	E	F
1 bit	8 bits	23 bits

$$(-1)^S \times (1 + 0.F) \times 2^{(E-127)}$$

S: The sign of a binary floating-point number is represented by a single bit. A **1** bit indicates a negative number, and a **0** bit indicates a positive number.

E: The binary exponent is not stored directly. Instead, the sum of the exponent and **7*F*** is stored as a **8**-bits. Notice that the binary exponent is unsigned, so it cannot be negative.

We will pad the left of the exponent with 0's up to 8 bits

exp	E=e + 127	Binary Rep.
5	+132	10000100
-10	+117	01110101
128	+255	11111111
-1	+126	01111110

The largest possible exponent is **128**, because when added to **127**, produces **255**, the largest unsigned value represented by **8** bits. The approximate range exponents is

from $-$**127** to **128**.

The main advantage of this format is a unique representation for exponent zero.

Single precision		
Sign	Biased Exponent	Mantissa
S	E	F
1 bit	8 bits	23 bits

$$(-1)^S \times (1 + 0.F) \times 2^{(E-127)}$$

F: It's the first **23**-bits after the **1.** in the significant. The fraction part assumes a normalized significand (in the form $\mathbf{1.}b_{10}b_{11}\ldots b_{32}$). Since the first bit is always an one, the leading one is not stored! This allows the storage of an additional bit at the end and so increases the precision slightly. This idea is know as the hidden one representation.

We will pad the right of the mantissa with 0's up to 23 bits.

Example

Consider the number $x = -1.11011 \times 2^{-48}$. Determine the floating point representation in IEEE single precision (**32** bits).

Solution

1. We immediately see that x is a negative number and so the sign is $\sigma = +$ Therefore the first bit in our floating point representation of this number will be $b_1 = S = 1$
2. Now we also see that the exponent $e = -48$ IEEE floating point single precision (32 bits) stores the number $E = e + 127$ instead though, and hence $E = -48 + 127 = 79 = (01001111)_2$. Therefore $b2b3 \ldots b9 = 01001111$.
3. lastly we will determine the last twenty-three digits which represent the fractional part. We note that $F = 0.11011$.

Sign	Exponent	Mantissa
1	01001111	11011000000000000000000

Thus the IEEE single precision representation of -1.11011×2^{-48} is $10100111111011000000000000000000_2$

✎ **Exercise 4.** Represent the following binary numbers as single precision float. Show:

- a scientific notation representation
- the significand, exponent, and sign bits (**32** bits total)
- a hexadecimal representation of these **32** bits

a) -1.11 b) $+1101.101$ c) -0.00101 d) $+100111.0$ e) $+0.0000001101011$

Double precision		
Sign	Exponent	Mantissa
S	E	F
1 bit	11 bits	52 bits

$$(-1)^{S} \times (1 + 0.F) \times 2^{(E-1023)}$$

- The largest possible exponent is **1024**, because when added to **1023**, produces **2047**, the largest unsigned value represented by **11** bits.
- The approximate range is from -1023 to **1024**.

Floating point representation

Comparison Chart: Single Precision vs. Double Precision

	Single Precision	**Double Precision**
Overview	Uses **32** bits of memory to represent a numerical value, with one of the bits representing the sign of mantissa	Uses **64** bits of memory to represent a numerical value, with one of the bits representing the sign of mantissa
Biased exponent	**8** bits used for exponent	**11** bits used for exponent
Mantissa	Uses **23** bits for mantissa (to represent fractional part)	Uses **52** bits for mantissa (to represent fractional part)
Real-world Application	Often used for games or any program that requires wider representation without a high level of precision	Often used for scientific calculations and complex programs that require a high level of precision

Floating point representation

Comparison Chart: Single Precision vs. Double Precision

Example

Take Euler's number (e), for example. Here are the first 50 decimal digits of e:

$$\mathbf{2.71828182845904523536028747135266249775724709369 99}$$

Here's Euler's number in binary, converted to single precision:

$$\mathbf{0}\ \underbrace{\mathbf{10000000}}_{\textbf{8 bits}}\ \underbrace{\mathbf{01011011111100001010100}}_{\textbf{23 bits}}$$

Here's Euler's number in binary, converted to double precision:

$$\mathbf{0}\ \underbrace{\mathbf{10000000000}}_{\textbf{11 bits}}\ \underbrace{\mathbf{0101101111110000101010001011000101000101011101101001}}_{\textbf{52 bits}}$$

- If you increase the amount of bits allocated to the Mantissa you increase the accuracy/precision
- If you increase the amount of bits allocated to the exponent you increase the range of the number

1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

The rules for converting a decimal number into floating point are as follows:

1. Convert the absolute value of the number to binary, perhaps with a fractional part after the binary point. This can be done by converting the integral and fractional parts separately. The integral part is converted by using succesive division. The fractional part can be converted by multiplication.
2. Append $\times \mathbf{2^0}$ to the end of the binary number (which does not change its value).
3. Normalize the number. Move the binary point so that it is one bit from the left. Adjust the exponent of two so that the value does not change.
4. Place the mantissa into the mantissa field of the number. Omit the leading one, and fill with zeros on the right.
5. Add the bias to the exponent of two, and place it in the exponent field. The bias is $\mathbf{2^{k-1} - 1}$, where $\boldsymbol{k}$ is the number of bits in the exponent field. For the eight-bit format, $\boldsymbol{k} = \mathbf{3}$, so the bias is $\mathbf{2^{3-1} - 1 = 3}$. For IEEE **32**-bit, $\boldsymbol{k} = \mathbf{8}$, so the bias is $\mathbf{2^{8-1} - 1 = 127}$.
6. Set the sign bit, **1** for negative, **0** for positive, according to the sign of the original number.

Example (How would **23.85** be stored)

1. We convert the integer part into binary $\mathbf{23 = 10111}$
2. We convert the decimal part into binary floating point

$$\underbrace{\begin{array}{lll} \mathbf{0.85 \times 2} & \mathbf{= 1.7} & \mathbf{= 1} \\ \mathbf{0.7 \times 2} & \mathbf{= 1.4} & \mathbf{= 1} \\ \mathbf{0.4 \times 2} & \mathbf{= 0.8} & \mathbf{= 0} \\ \mathbf{0.8 \times 2} & \mathbf{= 1.6} & \mathbf{= 1} \\ \mathbf{0.6 \times 2} & \mathbf{= 1.2} & \mathbf{= 1} \\ \mathbf{0.2 \times 2} & \mathbf{= 0.4} & \mathbf{= 0} \\ \mathbf{0.4 \times 2} & \mathbf{= 0.8} & \mathbf{= 0} \\ \mathbf{0.8 \times 2} & \mathbf{= 1.6} & \mathbf{= 1} \end{array}}_{\mathbf{0.85 = 0.11\overline{0110}}} \Longrightarrow \mathbf{23.85 = 10111.11\ 0110\ 0110\ 0110 \cdots}$$

3. We normalize the binary floating number $\mathbf{23.85 = 1.011111\ 0110\ 0110\ 0110 \times 2^4}$

Example

4. IEEE representation:
   - The sign bit is **0**, because it is positive.
   - The true exponent is **4**, so the biased exponent is **4 + 127 = 131 = 1000 0011**
   - The fraction is **011111 0110 0110 0110 0110 0**

Single precision		
Sign	Exponent	Mantissa
0	1000 0011	011111 0110 0110 0110 0110 0
1 bit	8 bits	23 bits

Thus the IEEE single precision representation of **23.85** is $\mathbf{01000001101111101100110011001100_2}$

✎ **Exercise 5.** Represent the following decimal numbers as single precision float. Show:
- a scientific notation representation
- the significand, exponent, and sign bits (**32** bits total)
- a hexadecimal representation of these **32** bits

a) **15.0625** b) **−35.6875** c) **86.5625**

1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

The rules for converting a floating point number into decimal are simply to reverse of the decimal to floating point conversion:

1. If the original number is in hex, convert it to binary.
2. Separate into the sign, exponent, and mantissa fields.
3. Extract the mantissa from the mantissa field, and restore the leading one. You may also omit the trailing zeros.
4. Extract the exponent from the exponent field, and subtract the bias to recover the actual exponent of two. As before, the bias is $\mathbf{2^{k-1} - 1}$, where $\boldsymbol{k}$ is the number of bits in the exponent field, giving **3** for the **8**-bit format, **127** for the **32**-bit and **1023** for the **64**-bit.
5. De-normalize the number: move the binary point so the exponent is **0**, and the value of the number remains unchanged.
6. Convert the binary value to decimal. This is done just as with binary integers, but the place values right of the binary point are fractions.
7. Set the sign of the decimal number according to the sign bit of the original floating point number: make it negative for **1**; leave positive for **0**.

If the binary exponent is very large or small, you can convert the mantissa directly to decimal without de-normalizing. Then use a calculator to raise two to the exponent, and perform the multiplication. This will give an approximate answer, but is sufficient in most cases.

Example

Convert the **32**-bit floating point number **44361000** (in hex) to decimal

Solution

1. Convert: $\mathbf{44361000_{16} = 0100\ 0100\ 0011\ 0110\ 0001\ 0000\ 0000\ 0000_2}$.
2. Seprate:

Single precision		
Sign	Exponent	Mantissa
0=+	10001000 = 136	01101100001000000000000

3.
   - Actual exponent of **2** is $\boldsymbol{e = E - 127 = 136 - 127 = 9}$
   - De-normalize: $\mathbf{1.01101100001_2 \times 2^9 = 1011011000.01_2}$
4. Convert $\mathbf{1011011000.01 = 512 + 128 + 64 + 16 + 8 + 0.25 = 728.25}$.
5. Sign: positive

Result $\mathbf{44361000_{16} = 728.25}$

Example

Consider the following floating point number presented in IEEE single precision (32 bits) as **01101011101101010000000000000** (Determine the sign σ, exponent e, and significand/mantissa F and determine the value of $x = \sigma\ 1 \cdot F \times 2^e$

Solution

1. We note that the first bit of the number given above is $S = b1 = 0$ It immediately follows that we have that the sign of x is $\sigma = (-1)^0 = +$
2. Now the next eight bits $b_2 b3 \ldots b$ care **11010111** and represent $E = e + 127$. We want to find what decimal number represents the binary number $E = (11010111)_2$. We have that:

$$E = 1 + 2 + 4 + 0 + 16 + 0 + 64 + 128 = 215 \implies e = E - 127 = 215 - 127 = 88$$

3. Lastly, recall that the twenty-three bits $b_{10} b_{11} \ldots b_{32}$ represent the fractional part, and that $F = 0 \cdot b_{10} b_{11} \ldots b_{32}$ and so:

$$F = 0.0110101$$

So the decimal representation of this number is $x = \sigma\ 1 \cdot F \times 2^e = +1.0110101 \times 2^{88}$

Example

Consider the following number presented in IEEE single precision **32** bits **11001100101111100010000000000** Determine the sign $\sigma = (-1)^S$, exponent e, and significand/mantissa F and determine the value of $x = \sigma\, 1 \cdot F \times 2^e$

Solution

1. Once again we immediately have that since $S = b1 = 1$ then the sign of x is $\sigma = (-1)^1 = -$
2. Now next eight bits are **10011001** These bits represent $E = e + 127$. Thus we have that:

   $$E = (10011001)_2 = 1+0+0+8+16+0+0+128 = 153 \implies e = E-127 = 153-127 = 26$$

3. Lastly find the mantissa by using the last twenty-three bits of the given number. We have that:

   $$F = 0.0111110001$$

   So the decimal representation of this number is $x = \sigma\, 1 \cdot F \times 2^e = -1.0111110001 \times 2^{26}$

✎ **Exercise 6.** Convert the following n-bit floating point number to decimal.

a) $D3_{16}$ with $n = 8$ b) $BE580000_{16}$ with $n = 32$

c) $A3358000_{16}$ with $n = 32$ d) 76650000_{16} with $n = 32$

✎ **Exercise 7.** Convert the decimal number **0.125** into IEEE single precision floating point. Give your answer in hexadecimal.

1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

- As the significand is assumed to have a hidden **1** as the most significant bit, all **0**s in the significand part of the number will be taken as $\mathbf{1,00\cdots0}$
- Thus zero is represented in the IEEE Standard by all **0**s for the exponent and all **0**s for the significand.
- All **0**s for the exponent is not allowed to be used for any other number.
- If the sign bit is **0** and all the other bits **0**, the number is $+\mathbf{0}$. If the sign bit is **1** and all the other bits **0**, it is $-\mathbf{0}$.
- Even though $+\mathbf{0}$ and $-\mathbf{0}$ have distinct representations they are assumed equal. Thus the representations of $+\mathbf{0}$ and $-\mathbf{0}$ are:

Representations of $+\mathbf{0}$		
Sign	Exponent	Mantissa
0	00000000	00000000000000000000000

Representations of $-\mathbf{0}$		
Sign	Exponent	Mantissa
1	00000000	00000000000000000000000

1 Fraction Number
- Fractional Decimal Number
- Fractional Binary Number
- Converting Binary Fraction to a Decimal
- Converting Decimal to a Binary Fraction

2 Floating point representation
- Normalized binary scientific notation
- IEEE Floating Point Standard
  - IEEE single precision
  - IEEE doule precision

3 Conversions
- Decimal to Floating-Point Conversions
- Float to Decimal Conversion

4 Special Values in IEEE 754-1985 Standard (32 Bits)
- Representation of Zero
- Representation of Infinity
- Representation of Non Numbers
- Largest and Smallest Positive Floating Point Numbers:
- Subnormal Numbers

- All **1**s in the exponent field is assumed to represent infinity **(∞)**
- A sign bit **0** represents $+\infty$ and a sign bit **1** represents $-\infty$. Thus the representations of $+\infty$ and $-\infty$ are:

Representations of $+\infty$		
Sign	Exponent	Mantissa
0	11111111	00000000000000000000000

Representations of $-\infty$		
Sign	Exponent	Mantissa
1	11111111	00000000000000000000000

1 Fraction Number
Fractional Decimal Number
Fractional Binary Number
Converting Binary Fraction to a Decimal
Converting Decimal to a Binary Fraction
2 Floating point representation
Normalized binary scientific notation
IEEE Floating Point Standard
IEEE single precision
IEEE doule precision
3 Conversions
Decimal to Floating-Point Conversions
Float to Decimal Conversion
4 Special Values in IEEE 754-1985 Standard (32 Bits)
Representation of Zero
Representation of Infinity
Representation of Non Numbers
Largest and Smallest Positive Floating Point Numbers:
Subnormal Numbers

- When an operation is performed by a computer on a pair of operands, the result may not be mathematically defined.
- the IEEE Standard defines two types of NaN.
  - When the result of an operation is not defined (i.e., indeterminate) it is called a **Quiet NaN** (QNaN). For example: $\frac{0}{0}$, $+\infty - \infty$, $\sqrt{-1}$
  - The other type of NaN is called a **Signalling NaN** (SNaN). This is used to give an error message. When an operation leads to a floating point underflow, i.e., the result of a computation is smaller than the smallest number that can be stored as a floating point number, or the result is an overflow, i.e., it is larger than the largest number that can be stored, SNaN is used.
-
  - QNaN is represented by **0** or **1** as the sign bit, all **1**s as exponent, and a **0** as the left-most bit of the significand and at least one **1** in the rest of the significand.
  - SNaN is represented by **0** or **1** as the sign bit, all **1**s as exponent, and a **1** as the left-most bit of the significand and any string of bits for the remaining **22** bits.
- We give below the representations of QNaN and SNaN.

QNaN		
Sign	Exponent	Mantissa
0 or 1	11111111	00010000000000000000000
The most significant bit of the significand is 0. There is at least one 1 in the rest of the significand.		

SNaN		
Sign	Exponent	Mantissa
0 or 1	11111111	10000000000010000000000
The most significant bit of the significand is 1. Any combination of bits is allowed for the other bits of the significand.		

1 Fraction Number
- Fractional Decimal Number
- Fractional Binary Number
- Converting Binary Fraction to a Decimal
- Converting Decimal to a Binary Fraction

2 Floating point representation
- Normalized binary scientific notation
- IEEE Floating Point Standard
  - IEEE single precision
  - IEEE doule precision

3 Conversions
- Decimal to Floating-Point Conversions
- Float to Decimal Conversion

4 Special Values in IEEE 754-1985 Standard (32 Bits)
- Representation of Zero
- Representation of Infinity
- Representation of Non Numbers
- Largest and Smallest Positive Floating Point Numbers:
- Subnormal Numbers

Largest Positive Number		
Sign	Exponent	Mantissa
0	11111110	11111111111111111111111

- Significand: $\mathbf{1,1111\cdots1 = 2^0 + 2^{-1} + 2^{-2} + \cdots + 2^{-23} = 1 + 1 - 2^{-23} = 2 - 2^{-23}}$.
- Exponent: $\mathbf{(254 - 127) = 127}$.
- Largest Number = $\mathbf{+1,1\cdots1 \times 2^{127} = \left(1 + 1 - 2^{-23}\right) \times 2^{127} = (2 - 2^{-23}) \times 2^{127} \approx 3.403 \times 10^{38}}$.

If the result of a computation exceeds the largest number that can be stored in the computer, then it is called an **overflow**.

Smallest Positive Number		
Sign	Exponent	Mantissa
0	00000001	00000000000000000000000

- Significand: $\mathbf{1,0\cdots0 = 1}$
- Exponent: $\mathbf{(1 - 127) = -126}$.
- The smallest normalized number is = $\mathbf{1 \times 2^{-126} \approx 1.1755 \times 10^{-38}}$.

If the result of a computation is less than the smallest number that can be stored in the computer, then it is called an **underflow**.

1 Fraction Number
- Fractional Decimal Number
- Fractional Binary Number
- Converting Binary Fraction to a Decimal
- Converting Decimal to a Binary Fraction

2 Floating point representation
- Normalized binary scientific notation
- IEEE Floating Point Standard
  - IEEE single precision
  - IEEE doule precision

3 Conversions
- Decimal to Floating-Point Conversions
- Float to Decimal Conversion

4 Special Values in IEEE 754-1985 Standard (32 Bits)
- Representation of Zero
- Representation of Infinity
- Representation of Non Numbers
- Largest and Smallest Positive Floating Point Numbers:
- Subnormal Numbers

★ Special Values in IEEE 754-1985 Standard (32 Bits) ★

Subnormal Numbers

- When all the exponent bits are **0** and the leading hidden bit of the siginificand is **0**, then the floating point number is called a **subnormal number**.
- Thus, one logical represen- tation of a subnormal number is

$$(-1)^{S} \times 0.F \times 2^{(-127)}$$

where ***F*** has at least one **1** (otherwise the number will be taken as **0**)

However, the standard uses –126, i.e., bias +1 for the exponent rather than –127 which is the bias for some not so obvious reason, possibly because by using –126 instead of –127, the gap between the largest subnormal number and the smallest normalized num- ber is smaller.

Largest subnormal number		
Sign	Exponent	Mantissa
0	00000000	11111111111111111111111
$(1 - 2^{-23})2^{-126} = 0.9999999988 \times 2^{-126} \approx 2^{-126}$		

Smallest Positive Subnormal Number		
Sign	Exponent	Mantissa
0	00000000	00000000000000000000001
$2^{-23} \times 2^{-126} = 2^{-149} \approx 2^{-126}$		

★ Special Values in IEEE 754-1985 Standard (32 Bits) ★

Summary

Value	Sign	Exponent (8 bits)	Significand (23 bits)
$+0$	**0**	**00000000**	**00 · · · 00**
-0	**1**	**00000000**	**00 · · · 00**
$+1.F \times 2^{E-127}$	**0**	**00000001** to **11111110**	***aa · · · aa*** (***a*** = **0** or **1**)
$-1.F \times 2^{E-127}$	**1**	**00000001** to **11111110**	***aa · · · aa*** (***a*** = **0** or **1**)
$+\infty$	**0**	**11111111**	**00 · · · 00**
$-\infty$	**1**	**11111111**	**00 · · · 00**
SNaN	**0** or **1**	**11111111**	**00 · · · 01** to **011 · · · 11** leading bit **0** at least one **1** in the rest
QNaN	**0** or **1**	**11111111**	**1000 · · · 10** leading bit **1**
L.P.F.P.N	**0**	**11111110**	**11111111111111111111111**
S.P.F.P.N	**0**	**00000001**	**00000000000000000000000**
Positive subnormal $0.F \times 2^{-126}$	**0**	**00000000**	**00000001** to **11111111** at least one **1** in the rest


Thank you! Questions?