

FLOATING POINT NUMBERS

IEEE floating point standard

Richard Hayden (with thanks to **Naranker Dulay**)
rh@doc.ic.ac.uk

<http://www.doc.ic.ac.uk/~rh/teaching>

or

<https://www.doc.ic.ac.uk/~wl/teachlocal/arch1>

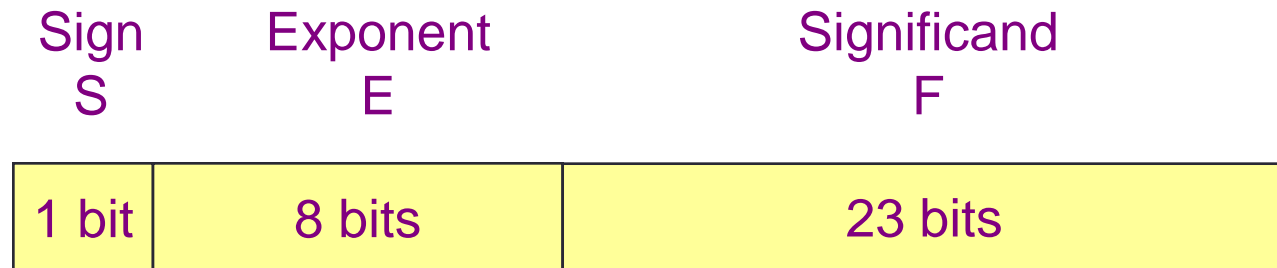
or

CATE

IEEE floating point standard

- **IEEE: institute of electrical and electronic engineers (USA)**
- **Comprehensive standard** for binary floating point arithmetic
- Widely adopted → **predictable results** independent of architecture
- Standard defines:
 - **Format** of binary floating point numbers, i.e. how the fields are stored in memory
 - **Semantics** of arithmetic operations
 - Rules for **error conditions**

Single precision format (32-bit)



- Mantissa is called the **significand** in the IEEE standard
- Value represented is $\pm 1.F \times 2^{E-127}$
- The **normal bit** (the 1.) is omitted from the significand field → a **hidden bit**
- Single precision yields **24 bits** (approx. **7 decimal digits** of precision)
- **Normalised ranges** in decimal are approximately:
 -10^{38} to -10^{-38} , **0**, 10^{38} to 10^{-38}

Exponent field

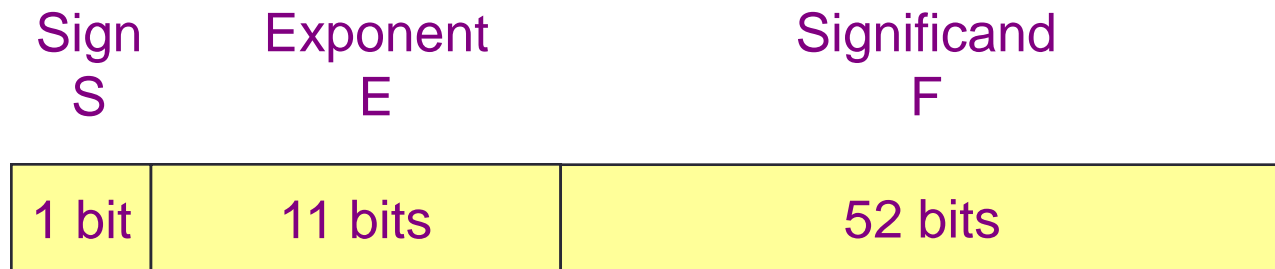
- In the IEEE standard, exponents are stored as **excess (bias) values**, not as 2's complement

- Example: **In 8-bit excess-127**

-127	would be held as	0000 0000
...		...
0		0111 1111
1		1000 0000
...		...
128		1111 1111

- Allows non-negative floating point numbers to be compared using simple integer comparisons

Double precision format (64-bit)



- Value represented is $\pm 1.F \times 2^{E-1023}$
- Double precision yields **53 bits** (approx. **16 decimal digits** of precision)
- **Normalised ranges** in decimal are approximately:
 -10^{308} to -10^{-308} , **0**, 10^{308} to 10^{-308}
- Single precision generally reserved for when memory is scarce or for debugging numerical calculations since rounding errors show up more quickly

Example: conversion to IEEE format

What is 42.6875 in IEEE single precision format?

1. Convert to **binary number**: $42.6875 = 10\ 1010.1011$

Example: conversion to IEEE format

What is 42.6875 in IEEE single precision format?

1. Convert to **binary number**: $42.6875 = 10\ 1010.1011$
2. **Normalise**: $1.0101\ 0101\ 1 \times 2^5$

Example: conversion to IEEE format

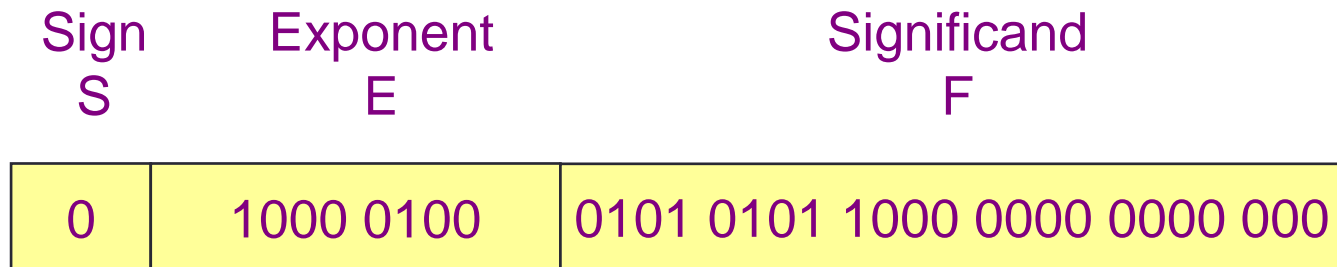
What is 42.6875 in IEEE single precision format?

1. Convert to **binary number**: $42.6875 = 10\ 1010.1011$
2. **Normalise**: $1.0101\ 0101\ 1 \times 2^5$
3. **Significand field** is thus:
 $0101\ 0101\ 1000\ 0000\ 0000\ 000$

Example: conversion to IEEE format

What is 42.6875 in IEEE single precision format?

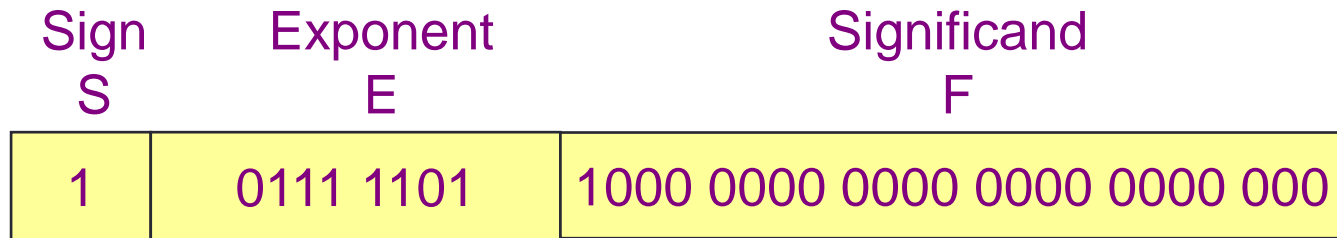
1. Convert to **binary number**: $42.6875 = 10\ 1010.1011$
2. **Normalise**: $1.0101\ 0101\ 1 \times 2^5$
3. **Significand field** is thus:
 $0101\ 0101\ 1000\ 0000\ 0000\ 0000$
4. **Exponent field** is ($5 + 127 = 132$): $1000\ 0100$



Hex: 422A C000

Example: conversion from IEEE format

What is the IEEE single precision value represented by **BEC0 0000** in decimal?



1. **Exponent field:**

$$0111\ 1101 = 125$$

Example: conversion from IEEE format

What is the IEEE single precision value represented by **BEC0 0000** in decimal?

Sign S	Exponent E	Significand F
1	0111 1101	1000 0000 0000 0000 0000 000

1. **Exponent field:** $0111\ 1101 = 125$
2. **True binary exponent:** $125 - 127 = -2$

Example: conversion from IEEE format

What is the IEEE single precision value represented by **BEC0 0000** in decimal?

Sign S	Exponent E	Significand F
1	0111 1101	1000 0000 0000 0000 0000 000

1. **Exponent field:** $0111\ 1101 = 125$
2. **True binary exponent:** $125 - 127 = -2$
3. **Significand field + hidden bit:**
 $1.1000\ 0000\ 0000\ 0000\ 0000\ 000$

Example: conversion from IEEE format

What is the IEEE single precision value represented by **BEC0 0000** in decimal?

Sign S	Exponent E	Significand F
1	0111 1101	1000 0000 0000 0000 0000 000

1. **Exponent field:** $0111\ 1101 = 125$
2. **True binary exponent:** $125 - 127 = -2$
3. **Significand field + hidden bit:**
 $1.1000\ 0000\ 0000\ 0000\ 0000\ 000$
4. **So unsigned value is** $1.1 \times 2^{-2} = 0.011$ (binary)
 $= 0.25 + 0.125 = 0.375$ (decimal)

Example: conversion from IEEE format

What is the IEEE single precision value represented by **BEC0 0000** in decimal?

Sign S	Exponent E	Significand F
1	0111 1101	1000 0000 0000 0000 0000 000

1. **Exponent field:** $0111\ 1101 = 125$
2. **True binary exponent:** $125 - 127 = -2$
3. **Significand field + hidden bit:**
 $1.1000\ 0000\ 0000\ 0000\ 0000\ 000$
4. **So unsigned value** is $1.1 \times 2^{-2} = 0.011$ (binary)
 $= 0.25 + 0.125 = 0.375$ (decimal)
5. Adding **sign bit** gives finally -0.375

Example: addition

Carry out the addition $42.6875 + 0.375$ in IEEE single precision arithmetic

Number	Sign	Exponent	Significand
42.6875	0	1000 0100	0101 0101 1000 0000 0000 000
0.375	0	0111 1101	1000 0000 0000 0000 0000 000

- To add these numbers, exponents must be the same → make the smaller exponent equal to the larger by shifting significand accordingly
- **Note:** must restore **hidden bit** when carrying out floating operations

Example: addition (cont.)

- **Significand** of larger no.: 1.0101 0101 1000 0000 0000 000
- **Significand** of smaller no.: 1.1000 0000 0000 0000 0000 000

Example: addition (cont.)

- **Significand** of larger no.: 1.0101 0101 1000 0000 0000 000
- **Significand** of smaller no.: 1.1000 0000 0000 0000 0000 000
- Exponents differ by $(1000\ 0100 - 0111\ 1101 = 7)$ so shift binary point of smaller no. 7 places to the left:

Example: addition (cont.)

- **Significand** of larger no.: 1.0101 0101 1000 0000 0000 000
- **Significand** of smaller no.: 1.1000 0000 0000 0000 0000 000

- Exponents differ by $(1000\ 0100 - 0111\ 1101 = 7)$ so shift binary point of smaller no. 7 places to the left:

- **Significand** of smaller no.: 0.0000 0011 0000 0000 0000 000
- **Significand** of larger no.: 1.0101 0101 1000 0000 0000 000
- **Significand** of **sum**: 1.0101 1000 1000 0000 0000 000

Example: addition (cont.)

- **Significand** of larger no.: 1.0101 0101 1000 0000 0000 000
- **Significand** of smaller no.: 1.1000 0000 0000 0000 0000 000
- Exponents differ by $(1000\ 0100 - 0111\ 1101 = 7)$ so shift binary point of smaller no. 7 places to the left:
- **Significand** of smaller no.: 0.0000 0011 0000 0000 0000 000
- **Significand** of larger no.: 1.0101 0101 1000 0000 0000 000
- **Significand** of **sum**: 1.0101 1000 1000 0000 0000 000
- So **sum** is $1.0101\ 1000\ 1 \times 2^5 = 10\ 1011.0001 = 43.0625$

Sign S	Exponent E	Significand F
0	1000 0100	0101 1000 1000 0000 0000 000

Special values

- IEEE formats can encode five kinds of values: **zero**, **normalised numbers**, **denormalised numbers**, **infinity** and **not-a-number (NaNs)**

Single precision representations:

IEEE value	Sign field	Exponent field	Significand field	True exponent
± 0	0 or 1	0	0 (all zeros)	
\pm denormalised no.	0 or 1	0	Any non-zero bit pattern	-126
\pm normalised no.	0 or 1	1 ... 254	Any bit pattern	-126 ... 127
$\pm\infty$	0 or 1	255	0 (all zeros)	
Not-a-number	0 or 1	255	Any non-zero bit pattern	

Denormalised numbers

- An **all zero exponent** is used to represent both **zero** and **denormalised numbers**
- An **all one exponent** is used to represent **infinities** and **not-a-numbers**
- Means **range for normalised numbers is reduced**, for single precision the exponent range is $-126 \dots 127$ rather than $-127 \dots 128$
- **Denormalised numbers** represent values between the underflow limits and zero, i.e. for single precision we have $\pm 0.F \times 2^{-126}$
- Allows a more **gradual shift to zero** – useful in some numerical applications

Infinites and NaNs

- Infinites represent values **exceeding the overflow limits** and for divisions of non-zero quantities by zero
- You can do basic `arithmetic' with them, e.g.:

$$\infty + 5 = \infty, \quad -\infty + -\infty = -\infty$$

- NaNs represent the result of operations which have **no (real) mathematical interpretation**, e.g.

$$\frac{0}{0}, \quad +\infty + -\infty, \quad 0 \times \infty, \quad \text{square root of a negative number}$$

- Operations resulting in NaNs can either yield a NaN result (**quiet NaN**) or an exception (**signalling NaN**)