Floating Point Numbers

Real Numbers: pi = `3.14159265...` e = `2.71828...`

Scientific Notation: has a single digit to the left of the decimal point.

A number in Scientific Notation with no leading 0s is called a **Normalised Number:** 1.0×10^{-8}

Not in **normalised** form: 0.1×10^{-7} or 10.0×10^{-9}

Can also represent **binary** numbers in scientific notation: 1.0×2^{-3}

Computer arithmetic that supports such numbers is called **Floating Point**.

The form is $1.xxxx\ldots \times 2^{yy\ldots}$

Using **normalised scientific notation**

1. Simplifies the exchange of data that includes floating-point numbers
2. Simplifies the arithmetic algorithms to know that the numbers will always be in this form
3. Increases the accuracy of the numbers that can be stored in a word, since each unnecessary leading 0 is replaced by another significant digit to the right of the decimal point

---

Representation of Floating-Point numbers

$$-1^{S} \times M \times 2^{E}$$

Bit No	Size	Field Name
31	1 bit	Sign (S)
23-30	8 bits	Exponent (E)
0-22	23 bits	Mantissa (M)

A **Single-Precision** floating-point number occupies 32-bits, so there is a compromise between the size of the mantissa and the size of the exponent.

These chosen sizes provide a range of approx:

$$\pm\ 10^{-38}\ \ldots\ 10^{38}$$

- **Overflow**

  The exponent is too *large* to be represented in the Exponent field

- **Underflow**

  The number is too *small* to be represented in the Exponent field

To reduce the chances of underflow/overflow, can use 64-bit **Double-Precision** arithmetic

Bit No	Size	Field Name

63	1 bit	Sign (S)
52-62	11 bits	Exponent (E)
0-51	52 bits	Mantissa (M)

providing a range of approx

$$\pm 10^{-308} \ldots 10^{308}$$

These formats are called ...

IEEE 754 Floating-Point Standard

Since the mantissa is always `1.xxxxxxxxx` in the normalised form, no need to represent the leading `1`. So, effectively:

- **Single Precision:** mantissa `===>` `1` bit + `23` bits
- **Double Precision:** mantissa `===>` `1` bit + `52` bits

Since zero (0.0) has no leading 1, to distinguish it from others, it is given the reserved bitpattern all 0s for the exponent so that hardware won't attach a leading 1 to it. Thus:

- Zero (0.0) = `0000...0000`
- Other numbers = $-1^S \times (1 + \text{Mantissa}) \times 2^E$

If we number the mantissa bits from left to right m1, m2, m3, ...

$$\text{mantissa} = m1 \times 2^{-1} + m2 \times 2^{-2} + m3 \times 2^{-3} + \ldots.$$

Negative exponents *could* pose a problem in comparisons.

For example (with two's complement):

	Sign	Exponent	Mantissa
1.0×2^{-1}	0	11111111	0000000 00000000 00000000
$1.0 \times 2^{+1}$	0	00000001	0000000 00000000 00000000

With this representation, the first exponent shows a "larger" binary number, making direct comparison more difficult.

To avoid this, **Biased Notation** is used for exponents.

If the real exponent of a number is X then it is represented as (X + bias)

IEEE single-precision uses a bias of `127`. Therefore, an exponent of

-1 is represented as $-1 + 127 = 126 = 01111110_2$

0 is represented as $0 + 127 = 127 = 01111111_2$

$+1$ is represented as $+1 + 127 = 128 = 10000000_2$

$+5$ is represented as $+5 + 127 = 132 = 10000100_2$

So the actual exponent is found by subtracting the bias from the stored exponent. Therefore, given S, E, and M fields, an IEEE floating-point number has the value:

$$\boxed{-1^{S} \times (1.0 + 0.M) \times 2^{E-bias}}$$

(Remember: it is (1.0 + 0.M) because, with normalised form, only the *fractional* part of the mantissa needs to be stored)

Floating Point Addition

Add the following two decimal numbers in scientific notation:

8.70×10^{-1} with 9.95×10^{1}

1. Rewrite the smaller number such that its exponent matches with the exponent of the larger number.

   $8.70 \times 10^{-1} = 0.087 \times 10^{1}$

2. Add the mantissas

   $9.95 + 0.087 = 10.037$ and write the sum 10.037×10^{1}

3. Put the result in Normalised Form

   $10.037 \times 10^{1} = 1.0037 \times 10^{2}$ (shift mantissa, adjust exponent)

   check for overflow/underflow of the exponent after normalisation

4. Round the result

   If the mantissa does not fit in the space reserved for it, it has to be rounded off.

   For Example: If only 4 digits are allowed for mantissa

   $1.0037 \times 10^{2} \implies 1.004 \times 10^{2}$

   (only have a *hidden* bit with *binary* floating point numbers)

Example addition in binary

Perform $0.5 + (-0.4375)$

$0.5 = 0.1 \times 2^{0} = 1.000 \times 2^{-1}$ (normalised)

$-0.4375 = -0.0111 \times 2^{0} = -1.110 \times 2^{-2}$ (normalised)

1. Rewrite the smaller number such that its exponent matches with the exponent of the larger number.

   $-1.110 \times 2^{-2} = -0.1110 \times 2^{-1}$

2. Add the mantissas:

$$1.000 \times 2^{-1} + -0.1110 \times 2^{-1} = 0.001 \times 2^{-1}$$

3. Normalise the sum, checking for overflow/underflow:

$$0.001 \times 2^{-1} = 1.000 \times 2^{-4}$$

`-126 <= -4 <= 127 ===>` No overflow or underflow

4. Round the sum:

The sum fits in 4 bits so rounding is not required

Check: $1.000 \times 2^{-4} = 0.0625$ which is equal to $0.5 - 0.4375$

Correct!

Floating Point Multiplication

Multiply the following two numbers in scientific notation by hand:

$$1.110 \times 10^{10} \times 9.200 \times 10^{-5}$$

1. Add the exponents to find

New Exponent = $10 + (-5) = 5$

If we add *biased* exponents, bias will be added twice. Therefore we need to subtract it once to compensate:

$$(10 + 127) + (-5 + 127) = 259$$

$259 - 127 = 132$ which is $(5 + 127) =$ biased new exponent

2. Multiply the mantissas

$$1.110 \times 9.200 = 10.212000$$

Can only keep three digits to the right of the decimal point, so the result is

$$10.212 \times 10^{5}$$

3. Normalise the result

$$1.0212 \times 10^{6}$$

4. Round it

$$1.021 \times 10^{6}$$

Example multiplication in binary:

$$1.000 \times 2^{-1} \times -1.110 \times 2^{-2}$$

1. Add the biased exponents

```
(-1 + 127) + (-2 + 127) - 127 = 124 ===> (-3 + 127)
```

2. Multiply the mantissas

```
   1.000
×  1.110
-----------
       0000
      1000
     1000
+   1000
-----------
    1110000  ===> 1.110000
```

The product is 1.110000×2^{-3}

Need to keep it to 4 bits 1.110×2^{-3}

3. Normalise (already normalised)

   At this step check for overflow/underflow by making sure that

   -126 <= Exponent <= 127

   1 <= Biased Exponent <= 254

4. Round the result (no change)

5. Adjust the sign.

   Since the original signs are different, the result will be negative

   -1.110×2^{-3}

Further Reading

IEEE-754 References and Conversion and another Converter

[Index]

---

last updated: 2-Dec-04 Ian Harries <ih@doc.ic.ac.uk>