Fractional Numbers

Numbers with a decimal point: Fixed-Point numbers; floating-point numbers.

Fixed-point number representation attempts to represent numbers with a fixed number of bits to represent the whole part of the number and a fixed number of bits to represent the fractional part of the number.

These numbers can, of course, be signed or unsigned

Let's use 8-bits for illustration.

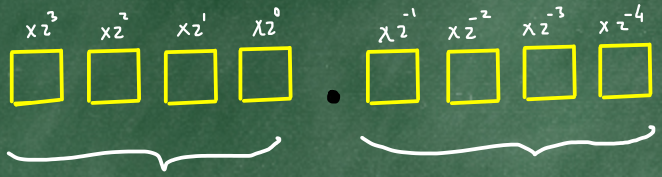
We know that, with 8-bits, we can represent whole numbers in the range: $0 \rightarrow 2^8 - 1$ unsigned,

or $-2^7 \rightarrow 2^7 - 1$ signed.

If we wish to represent fractional numbers with 8-bits, we assume a decimal point is located so that there is a fixed number of bits before the decimal point (more accurately, the binary point) and a fixed number of bits after.

Let's examine the case where we have 4 digits before and 4 digits after the decimal point.

Note, we do not physically represent the binary point in the bit sequence — we simply agree where it is. Once agreed, in a fixed-point representation, it cannot be moved.



Whole number Part

Fractional number Part

Range (unsigned)

$0 \rightarrow 2^4 - 1$ (15)

Range (Signed)

-2^3 (-8) $\rightarrow 2^3 - 1$ (7)

Range: $0 \rightarrow \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16}$

$= \frac{8+4+2+1}{16}$

$= \frac{15}{16} = 0.9375$

Total Range of number

Unsigned : $0 \rightarrow 15.9375$

Signed : $-8.0 \rightarrow 7.9375$

Examples :

Assuming 8-bit, fixed-point representation, Convert the following to decimal :

(1) Unsigned : 10010101

$1\times2^{3}+0\times2^{2}+0\times2^{1}+1\times2^{0}$ + $0\times2^{-1}+1\times2^{-2}+0\times2^{-3}+1\times2^{-4}$

$8+0+0+1$ + $0+0.25+0+0.0625$

Answer = 9.3125

Before we look at a signed fixed-point example,

Convert the following signed binary numbers to decimal:

(i) 0111

2^3 2^2 2^1 2^0

0 1 1 1 → 0 + 4 + 2 + 1 = 7

— Nothing new here

(ii) 1010

-2^3 2^2 2^1 2^0

1 0 1 0 → −8 + 0 + 2 + 0 = −6

Here, since the MSB = 1, we know the number is negative, so we use -2^3 instead of 2^3

(ii) Signed Fixed-point example

1111 1101

$= -1\times2^3 + 1\times2^2 + 1\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 1\times2^{-4}$

$= -8 + 4 + 2 + 1 + 0.5 + 0.25 + 0 + 0.0625$

$= -1 + 0.8125$

$= -0.1875$

0.5
0.25
0.0
0.0625
0.8125

Answer = −0.1875

Let's look at all the possible fractional values in tabular form:

.0000	.0	.1000	.5
.0001	.0625	.1001	.5625
.0010	.125	.1010	.625
.0011	.1875	.1011	.6875
.0100	.25	.1100	.75
.0101	.3125	.1101	.8125
.0110	.375	.1110	.875
.0111	.4375	.1111	.9375

We see that each successive fraction is 0.0625 more than the previous one.

$0.0625 \left(= \frac{1}{2^4} = \frac{1}{16}\right)$ is the smallest step between any two numbers. It is $1/2^4$, since we are using only 4 bits to represent our fractional component of our number. We call it the resolution of our number.

Example fixed-point numbers using 4 bits before, and 4 bits after, the binary point.

What is the 4-bit fixed-point representation of the following:

(i) 4.75 → 4 is 0100

.75 is 1100

∴ Answer = 0100 1100

(ii) 16.0 → no representation, too few bits before decimal point. 15 (=1111) = biggest unsigned whole number in this representation.

(iii) 4·09375 → no representation, too few bits after the binary point. Smallest fraction is 0·0625 (=.0001) in this representation.

If we were to use 5 bits to represent our fractional number, we would get a higher resolution of $\frac{1}{2^5} = \frac{1}{32} = 0.03125$.

(Note this is twice the resolution we get with 4 bits. i.e., 0·03125 is ½ x 0·0625)

This means that every successive fraction would differ from previous one by 0·03125.

Therefore, using 5 bits after the binary point we could represent 4·09375 as follows:

Assuming the representation is unsigned.

100.00011

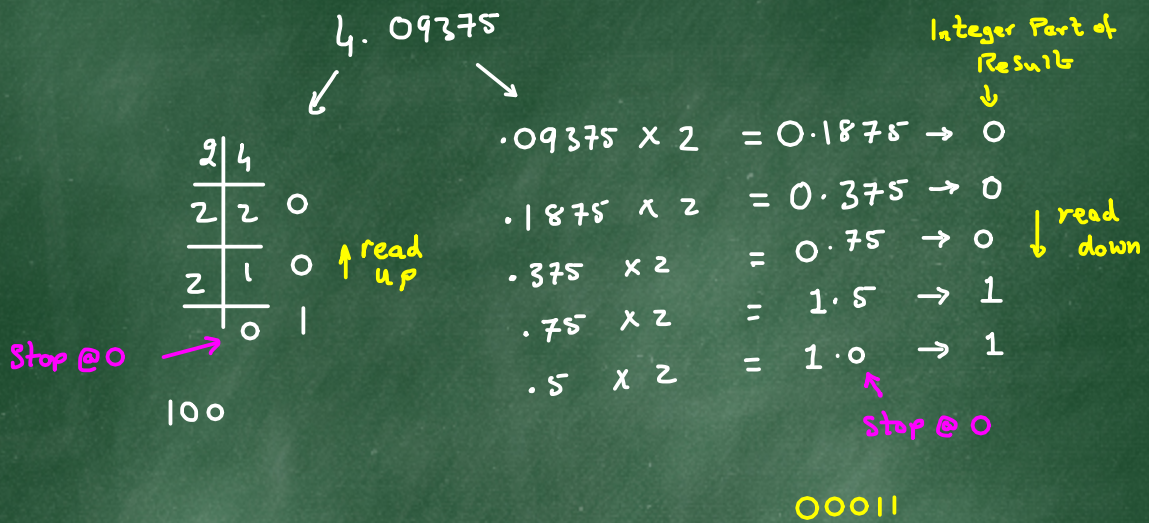
4 ↓ ↳ $\frac{1}{16} + \frac{1}{32} = \frac{3}{32} = .09375$

The downside is that we now only have 3 bits to represent the whole number part and so the range is halved

Note: Regardless of the number of bits we choose to represent our fractional part – there will alway be numbers between two successive fractions, that we cannot represent. There is always a limit to the resolution of our number representation. As programmers we may have to take account of this in our calculations.

Convert a decimal number → Binary with 3 bits before decimal point and 5 bits after

Example: 4.09375



Answer 10000011

Notice that if we had decided on 4 bits before the binary point, and 4 bits after, the result would have been:

01000001

This number is actually 4.0625, meaning that there is an inaccuracy in our representation!

We see that fixing the position of the binary point leads to a simple representation for fractional numbers.

However, it fixes ("sets in stone") the accuracy and the resolution of the resultant representation.

If the decimal point could be moved dynamically, we could trade-off between accuracy and range, as appropriate for the calculation being performed.

This is the idea behind floating-point representation. This is a complex topic and we will not cover it further in this course.

We note, however, that floating-point arithmetic is ubiquitous in all modern computer systems and programming languages.

- Fixed-point binary is used in Digital Signal Processing and in some games, where performance is more important than accuracy.
- It is simple and uses cheap processing hardware
- It is fast, but trades-off precision with range.