Number Representations

Fixed Point Representations

Let the bits in an N-bit binary number be designated

$$b_{N-1}, b_{N-2}, b_{N-3}, \cdots b_2, b_1, b_0$$

where b_{N-1} is called the most significant bit (MSB) and b_0 is called the least significant bit (LSB). Let a number M be represented by

$$M = 2^{N-1} \times b_{N-1} + 2^{N-2} \times b_{N-2} + \cdots + 2^2 \times b_2 + 2^1 \times b_1 + 2^0 \times b_0$$

For example, if $N = 3$ all the representable numbers would be

Decimal M	0	1	2	3	4	5	6	7
Binary M	000	001	010	011	100	101	110	111

This type of representation is called **unsigned binary**.

Fixed Point Representations

The biggest problem with unsigned binary is that it cannot represent negative numbers. There are three common ways of representing negative numbers,

1. Sign-magnitude representation
2. One's complement representation
3. Two's complement representation

In sign-magnitude representation the MSB indicates the sign of the number and the remaining bits represent its magnitude.

Using an MSB of 1 for a negative sign with $N = 3$,

Decimal M	0	1	2	3	0	–1	–2	–3
Binary M	000	001	010	011	100	101	110	111

Notice there are two zero representations 000 and 100.

Fixed Point Representations

In one's complement, negative numbers are formed by simply complementing all the bits in the corresponding positive number.

Decimal M	0	1	2	3	0	–1	–2	–3
Binary M	000	001	010	011	111	110	101	100

Again there are two zero representations 000 and 111.

In two's complement, negative numbers are formed by complementing all the bits and then adding one.

Decimal M	0	1	2	3	–1	–2	–3	–4
Binary M	000	001	010	011	111	110	101	100

Now there is only one zero representation 000. The addition is modulo-2 so when an overflow occurs it is ignored.

Fixed Point Representations

So far we have only represented integers. We can represent fractions by moving the "binary point" to the left. For integers it is at the right end.

$$M = 2^{N-1} \times b_{N-1} + 2^{N-2} \times b_{N-2} + \cdots + 2^{1} \times b_{1} + 2^{0} \times b_{0} \quad \underset{\text{binary point}}{\uparrow}$$

but we can change it to, for example,

$$M = 2^{N-3} \times b_{N-1} + 2^{N-4} \times b_{N-2} + \cdots + 2^{0} \times b_{2} + \underset{\text{binary point}}{\uparrow} \; 2^{-1} \times b_{1} + 2^{-2} \times b_{0}$$

and now the two least significant bits have weights 1/2 and 1/4 and can represent a fraction.

Fixed Point Representations

It is common in digital signal processing to make all numbers fractions lying between -1 and +1. For the 3-bit case

In sign-magnitude

Decimal M	$0/4$	$1/4$	$2/4$	$3/4$	$0/4$	$-1/4$	$-2/4$	$-3/4$
Binary M	0.00	0.01	0.10	0.11	1.00	1.01	1.10	1.11

In one's complement,

Decimal M	$0/4$	$1/4$	$2/4$	$3/4$	$0/4$	$-1/4$	$-2/4$	$-3/4$
Binary M	0.00	0.01	0.10	0.11	1.11	1.10	1.01	1.00

In two's complement,

Decimal M	$0/4$	$1/4$	$2/4$	$3/4$	$-1/4$	$-2/4$	$-3/4$	$-4/4$
Binary M	0.00	0.01	0.10	0.11	1.11	1.10	1.01	1.00

Fixed Point Representations

The name "two's complement" comes from the fact that to form the negative of a fraction we use its two's complement, meaning that number subtracted from two. For example, $1/4$ in 3-bit unsigned binary is 0.01 and $-1/4$ is represented by $2 - 1/4 = 7/4$ which in 3-bit unsigned binary is 1.11.

The overwhelming majority of digital signal processing with fixed point numbers uses two's complement and that is all we will use.

Two’s Complement Arithmetic Example

Find the sums and differences of these fractions using four-bit two’s complement arithmetic.

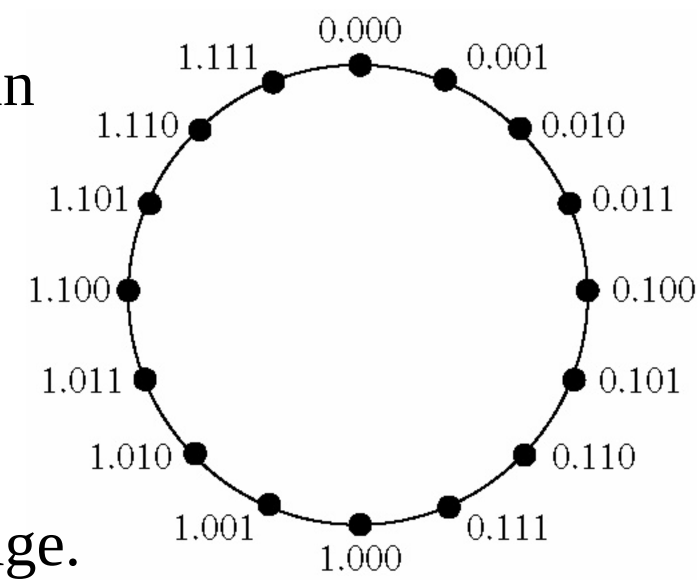
$3/4 \quad 3/8 \quad -5/8 \quad -7/8$

The available numbers are

Decimal	0	1/8	2/8	3/8	4/8	5/8	6/8	7/8
Binary	0.000	0.001	0.010	0.011	0.100	0.101	0.110	0.111
Decimal	–1	–7/8	–6/8	–5/8	–4/8	–3/8	–2/8	–1/8
Binary	1.000	1.001	1.010	1.011	1.100	1.101	1.110	1.111

Two’s Complement Arithmetic Example

Two numbers add like unsigned integers (ignoring the binary point). If there is an overflow, the carry is ignored making the addition of the integers effectively modulo-16. Then the binary point is re-introduced. The range of two’s complement numbers can be conceived as circular. When a sum or difference would exceed the allowed range , it overflows back into the allowed range (at a wrong anwer) by continuing to rotate in the same direction. To form the sum, 1/2 + 1/2 (0.100 + 0.100), start at the point 0.100 on the circle and move four positions clockwise This puts us at 1.000 which is -1 decimal, a wrong answer because we overflowed the range.



Two’s Complement Arithmetic Example

Notice that if we add -3/4 (1.010) and 7/8 (0.111), we start on the circle at 1.010 and move 7 positions clockwise arriving at 0.001 which is decimal 1/8 which is correct *even though the addition of 1010 and 0111 overflows 16*.