Representing Integers

Before we can examine the ways in which numbers are manipulated through arithmetical operations, we need to understand how numbers are represented. We'll start by looking at integer representation schemes. In the examples that follow, we'll assume a 5-bit representation for simplicity.

1. **Unsigned** – all bits represent powers of 2.
You already know this one: Unsigned representation is just what we have typically called the generic "binary" representation.

To determine the value of an unsigned number, add up the powers of 2 wherever you see a 1 in the bit pattern. For example, unsigned 11001 equals $2^4 + 2^3 + 2^0 = 25$.

To express a number in unsigned, we simply have to find out what powers of 2 make up the number (or you can use the base 10 to base 2 conversion algorithm).

The big problem with unsigned is that we can't represent negative values. The other 4 representation schemes attempt to address this problem. They differ in how they come up with representations for negative numbers.

2. **Sign-magnitude** – much like unsigned, except the leftmost bit becomes the sign bit. Because the leftmost bit no longer represents a large power of 2, we sacrifice some very large numbers for the sake of being able to represent negative numbers.

sign bit = 0 means positive number
sign bit = 1 means negative number

Besides the sign bit, the way to "encode" or "decode" numbers is the same as in unsigned. For example, 11001 = $-(2^3 + 2^0) = -9$.

The major disadvantage for sign-magnitude is that there are 2 possible representations for zero, either all zeros, or a 1 followed by all zeros. In other words, we have both a positive and a negative zero. Thus, comparing a number to zero (which is a common operation) becomes unnecessarily more complicated.

3. **One's-complement** – to find the negative, invert all the bits.
For positive numbers, the representation is the same as unsigned.

But again we have 2 representations for zero: either 00000, or 11111.

4. **Two's complement** – to find the negative, invert all the bits and add 1.
By adding 1 to find the negative, this gets rid of the duplicate zero problem. For example, 00000 is still the representation for 0. But what about 11111? It's a negative number, so find its opposite by inverting all the bits (to obtain 00000) and adding 1 (to obtain 00001)... and the result is +1, which means the 11111 represents −1.

One thing to note is that in 2's complement, we have one more negative number than positive.

5. **Biased** – subtract a number from the unsigned range.
In order to work with a biased representation, you need to know what the bias is. Usually, it will be half the size of the whole range, because we want about half of the numbers to be negative. For example, with 5 bits, there are 32 values that can be represented, so a common bias would be 16. Since the unsigned range is 0 to 31, we subtract 16 to find that the range of values for biased-16 is −16 to 15, the same as in 2's complement.

To find the biased-representation of n: add the bias, and then find the unsigned representation of (n + bias). For example, to find the 5-bit biased-16 representation of 7, this will be the same as the unsigned representation of 7+16=23, which is 10111.

Given a biased representation, to determine the value: first find out what the number would have been in unsigned, then subtract the bias. For example, if we are given the bit pattern 01110 and are told this is a 5-bit biased-16 number, we note that if it were unsigned it looks like 14, and subtracting the bias gives 14 − 16 = −2.

I usually find the above 2 paragraphs confusing, and it's not hard to confuse when to add versus subtract the bias. I recommend that you write out a chart like the following example so that you see where your number falls within the total range of values (assuming 5 bits with a bias of 16):

Bit pattern	Value if unsigned	Value if biased
00000	0	−16
...	...	...
01001	9	−7
...	...	...
11111	31	15

Fill out the top and bottom lines first, so you are familiar with what the smallest and largest numbers look like. Then put the number (or bit pattern) in question in the middle. Notice that the biased value = (unsigned value − bias) because this is just the definition of the biased representation. Or equivalently, the unsigned value = (biased value + bias). Instead of memorizing these formulas, you can just write out the chart and have confidence that you are right.

Summary: range of values for 5 bits

rep'n	smallest value	largest value	general range
unsigned	00000 = 0	11111 = 31	0 to $2^n - 1$
sign-magnitude	11111 = −15	01111 = 15	$-(2^{n-1} - 1)$ to $2^{n-1} - 1$
1's complement	10000 = −15	01111 = 15	$-(2^{n-1} - 1)$ to $2^{n-1} - 1$
2's complement	10000 = −16	01111 = 15	-2^{n-1} to $2^{n-1} - 1$
biased-16	00000 = −16	11111 = 15	-2^{n-1} to $2^{n-1} - 1$

value	unsigned	sign-magnitude	1's comp	2's comp	biased-16
-16				10000	00000
-15		11111	10000	10001	00001
-14		11110	10001	10010	00010
-13		11101	10010	10011	00011
-12		11100	10011	10100	00100
-11		11011	10100	10101	00101
-10		11010	10101	10110	00110
-9		11001	10110	10111	00111
-8		11000	10111	11000	01000
-7		10111	11000	11001	01001
-6		10110	11001	11010	01010
-5		10101	11010	11011	01011
-4		10100	11011	11100	01100
-3		10011	11100	11101	01101
-2		10010	11101	11110	01110
-1		10001	11110	11111	01111
0	00000	*00000*	*00000*	00000	10000
1	00001	00001	00001	00001	10001
2	00010	00010	00010	00010	10010
3	00011	00011	00011	00011	10011
4	00100	00100	00100	00100	10100
5	00101	00101	00101	00101	10101
6	00110	00110	00110	00110	10110
7	00111	00111	00111	00111	10111
8	01000	01000	01000	01000	11000
9	01001	01001	01001	01001	11001
10	01010	01010	01010	01010	11010
11	01011	01011	01011	01011	11011
12	01100	01100	01100	01100	11100
13	01101	01101	01101	01101	11101
14	01110	01110	01110	01110	11110
15	01111	01111	01111	01111	11111
16	10000				
17	10001				
18	10010				
19	10011				
20	10100				
21	10101				
22	10110				
23	10111				
24	11000				
25	11001				
26	11010				
27	11011				
28	11100				
29	11101				
30	11110				
31	11111				