1 Computer arithmetic with unsigned integers

All numbers are w-bit unsigned integers unless otherwise noted.

A w-bit unsigned integer x can be written out in binary as

$$x \equiv x_{w-1}x_{w-2}\ldots x_2x_1x_0,$$

where x_i is the ith bit of x. Each x_i is either 0 or 1, and the least-significant bit of x is written on the right.

Mathematically,

$$\begin{aligned} x &= 2^{w-1}x_{w-1} + 2^{w-2}x_{w-2} + \cdots + 2^2x_2 + 2x_1 + x_0 \\ &= \sum_{i=0}^{w-1} 2^i x_i. \end{aligned}$$

1.1 Powers of two

Here's an important fact:

$$\sum_{i=0}^{k-1} 2^i = 2^k - 1.$$

This should be intuitive. Think of decimal numbers: $9+1=10$; $99+1=100$; $999+1=1000$; and so forth. In each case,

$$\underbrace{999\ldots999}_{k-1 \text{ digits}} = \sum_{i=0}^{k-1} 9 \times 10^i = 10^k - 1.$$

The binary version, above, is just the same, but with base 2 instead of 10. The funny thing about $b=2$ is that $b-1=1$, so the equivalent of the "9" factor inside the summation is 1, which we don't need to write.

Also, the theory of geometric series tells us that

$$\sum_{i=0}^{k-1} ar^k = a\frac{1-r^k}{1-r}.$$

Here, $a=1$ and $r=2$, so

$$\sum_{i=0}^{k-1} 2^i = \frac{1-2^k}{1-2} = 2^k - 1.$$

1.2 And, or, xor, complement

Given $x \equiv x_{w-1} \ldots x_1 x_0$ and $y \equiv y_{w-1} \ldots y_1 y_0$, the bitwise operators $x \,\&\, y$ (and), $x \,|\, y$ (or), and $x \,\hat{}\, y$ (xor/exclusive-or) may be defined bitwise as follows.

$$(x \,\&\, y)_i = \begin{cases} 1 & \text{if } x_i = 1 \text{ and } y_i = 1, \\ 0 & \text{otherwise;} \end{cases}$$

$$(x \,|\, y)_i = \begin{cases} 1 & \text{if } x_i = 1 \text{ or } y_i = 1, \\ 0 & \text{otherwise;} \end{cases}$$

$$(x \,\hat{}\, y)_i = \begin{cases} 1 & \text{if } x_i \neq y_i, \\ 0 & \text{otherwise.} \end{cases}$$

These definitions are textual translations of the basic truth tables for single-bit and, or, and xor.

&	0	1
0	0	0
1	0	1

|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

The bitwise complement, or "twiddle," operator is unary; its bitwise definition is:

$$(\tilde{}\, x)_i = \begin{cases} 1 & \text{if } x_i = 0, \\ 0 & \text{otherwise.} \end{cases}$$

We call these operators *bitwise* because they affect their operands' bits independently. In non-bitwise operators, like normal addition or unary negation, the value(s) at one bit position in the operand(s) can affect *other* bit positions in the result.

Now, if you think about it, you'll see that these alternate definitions of the bitwise operators are equivalent to those above!

$$\begin{aligned} (x \,\&\, y)_i &= \min(x_i, y_i); \\ (x \,|\, y)_i &= \max(x_i, y_i); \\ (x \,\hat{}\, y)_i &= (x_i + y_i) \bmod 2; \\ (\tilde{}\, x)_i &= 1 - x_i. \end{aligned}$$

The simplicity of these alternate definitions—which comes mostly from the simplicity of bits themselves—leads to some of the "magic" we saw in class.

Some basic identities, true for all x:

$$\begin{aligned} (x \,\&\, x) = (x \,|\, x) &= x \\ (x \,\hat{}\, x) &= 0 \\ (x \,\&\, 0) &= 0 \\ (x \,|\, 0) = (x \,\hat{}\, 0) &= x \end{aligned}$$

Proving these identities is pretty easy. If they're not intuitive or obvious, why not try one?

Now, here are some proofs of some slightly more complex properties, worked out in more detail.

Lemma 1.1 *For any x and y, $(x \,\&\, y) \leq \min(x, y)$.*

Proof At each bit position i, $(x \,\&\, y)_i = \min(x_i, y_i)$. But $\min(x_i, y_i) \leq x_i$ by definition of min. Thus $(x \,\&\, y)_i \leq x_i$, and $(x \,\&\, y) \leq x$. A similar argument holds for y, and thus for $\min(x, y)$. □

To be very explicit, we can expand the latter part of the argument. Write

$$(x \,\&\, y) = \sum_{i=0}^{w-1} 2^i (x \,\&\, y)_i.$$

Consider term i in the sum, namely $2^i (x \,\&\, y)_i$. We saw above that $(x \,\&\, y)_i \leq x_i$, so the corresponding term is also less: $2^i (x \,\&\, y)_i \leq 2^i x_i$. This holds for every term, so it also holds for their sum:

$$(x \,\&\, y) = \sum_{i=0}^{w-1} 2^i (x \,\&\, y)_i \leq \sum_{i=0}^{w-1} 2^i x_i = x.$$

Lemma 1.2 *For any x and y, $(x \,|\, y) \geq \max(x, y)$.*

Proof Analogous to above. □

Lemma 1.3 *If $(x \,\&\, y) = 0$, then $x + y = (x \,|\, y) = (x \,\hat{}\, y)$.*

Proof The intuitive argument has to do with carries. Addition on binary numbers is like bitwise or, *except* for carries: when two bits in the same position are both 1, addition causes a carry that affects higher bit positions, while bitwise or doesn't. If $(x \,\&\, y) = 0$, there can be no carries.

For a more rigor, let's first write out $x + y$.

$$\begin{aligned} x + y &= \sum_{i=0}^{w-1} 2^i x_i + \sum_{i=0}^{w-1} 2^i y_i \\ &= \sum_{i=0}^{w-1} (2^i x_i + 2^i y_i) \\ &= \sum_{i=0}^{w-1} 2^i (x_i + y_i). \end{aligned}$$

Now, we know that $(x \,\&\, y) = 0$. This means that at every bit position i, $(x \,\&\, y)_i = \min(x_i, y_i) = 0$: at least one of x_i and y_i equals 0. And this, in turn, means that $x_i + y_i$ is either 0 or 1! (Both values are bits—either 0 or 1 by definition—so their sum equals 2 only if both bits are 1, causing a carry.) By enumerating cases, then, it's easy to see that $x_i + y_i = \max(x_i, y_i)$. But then, since this equals $(x \,|\, y)_i$, we see $(x \,|\, y) = x + y$.

The $\hat{}$ piece is just as easy. By enumerating cases, we see that $(a \,|\, b)_i = (a \,\hat{}\, b)_i$ unless $a_i = b_i = 1$. Again, this bad case never happens for our x and y. □

Lemma 1.4 *For any x, $-x = \tilde{}x + 1$.*

Proof We simply evaluate the term $x + (\tilde{}x + 1)$. If this term is 0, then $\tilde{}x + 1 = -x$ by the definition of $-x$. We use the word width w at one critical step.

$$\begin{aligned}
x + (\tilde{}x + 1) &= \sum_{i=0}^{w-1} 2^i x_i + \sum_{i=0}^{w-1} 2^i (\tilde{}x)_i + 1 \\
&= \sum_{i=0}^{w-1} 2^i (x_i + (\tilde{}x)_i) + 1 \\
&= \sum_{i=0}^{w-1} 2^i (x_i + 1 - x_i) + 1 \\
&= \sum_{i=0}^{w-1} 2^i + 1 \\
&= (2^w - 1) + 1 \\
&= 2^w \\
&\equiv 0 \pmod{2^w}.
\end{aligned}$$

□

Lemma 1.5 *If $0 \le x < 2^k$, then $(2^k + x)_k = 1$. (Assuming $k < w$.)*

Proof First, we show that $(2^k \,\&\, x) = 0$. It's quite easy to see that $x_i = 0$ for all $i \ge k$ (otherwise x would be 2^k or larger), and that $(2^k)_i = 0$ for all $i < k$ (by bitwise expansion of 2^k). The two numbers never have 1 in the same bit position, so their bitwise and is 0.

As a result, by Lemma 1.3, $2^k + x = (2^k \,|\, x)$, and $(2^k + x)_k = (2^k \,|\, x)_k = \max((2^k)_k, x_k) = 1$. □

Lemma 1.6 $(x \,\&\, (x-1)) = 0$ *if and only if x is either 0 or a power of 2.*

Proof ($\Rightarrow$) Assume x is 0 or a power of 2. If $x = 0$, then obviously $(x \,\&\, (x-1)) = 0$. So assume $x = 2^k$ for some k with $0 \le k < w$. Let's write out the bitwise expansions of x and $x - 1$.

$$\begin{aligned}
x &= 2^k; \\
x - 1 = 2^k - 1 &= \sum_{i=0}^{k-1} 2^i.
\end{aligned}$$

Thus, x_i (the ith bit of x) is 1 if and only if $i = k$. But $(x-1)_i = 1$ if and only if $i < k$. The two numbers never have 1 bits in the same positions, $\min(x_i, (x-1)_i) = (x \,\&\, (x-1))_i = 0$ for all i, and $(x \,\&\, (x-1)) = 0$.

($\Leftarrow$) Assume x is *not* 0 or a power of 2. Then x can be written as $2^k + r$, where $1 \le r < 2^k$. Here, k is the index of the highest 1 bit in x. And $x - 1$ can be written

as $2^k + (r-1)$, where $0 \leq r-1 < 2^k - 1$. Apply Lemma 1.5 twice, and we see that $x_k = (x-1)_k = 1$. So So $(x \,\&\, (x-1))_k = 1$ and $(x \,\&\, (x-1)) \neq 0$. $\square$