**Setting a bit**

Use the bitwise OR operator (|) to set a bit.

number |= 1UL << n;

That will set the nth bit of number.

Use 1ULL if number is wider than unsigned long; promotion of 1UL << n doesn't happen until after evaluating 1UL << n where it's undefined behaviour to shift by more than the width of a long. The same applies to all the rest of the examples.

**Clearing a bit**

Use the bitwise AND operator (&) to clear a bit.

number &= ~(1UL << n);

That will clear the nth bit of number. You must invert the bit string with the bitwise NOT operator (~), then AND it.

**Toggling a bit**

The XOR operator (^) can be used to toggle a bit.

number ^= 1UL << n;

That will toggle the nth bit of number.

**Checking a bit**

You didn't ask for this, but I might as well add it.

To check a bit, shift the number n to the right, then bitwise AND it:

bit = (number >> n) & 1U;

That will put the value of the nth bit of number into the variable bit.

**Changing the *n*th bit to *x***

Setting the nth bit to either 1 or 0 can be achieved with the following on a 2's complement C++ implementation:

number ^= (-x ^ number) & (1UL << n);

Bit n will be set if x is 1, and cleared if x is 0. If x has some other value, you get garbage. x = !!x will booleanize it to 0 or 1.

To make this independent of 2's complement negation behaviour (where -1 has all bits set, unlike on a 1's complement or sign/magnitude C++ implementation), use unsigned negation.

number ^= (-(unsigned long)x ^ number) & (1UL << n);

or

unsigned long newbit = !!x; // Also booleanize to force 0 or 1

number ^= (-newbit ^ number) & (1UL << n);

It's generally a good idea to use unsigned types for portable bit manipulation.