Chapter 85: Bitwise Operators - Real World Examples (snippets)

Section 85.1: Swapping Two Integers with Bitwise XOR (without additional memory allocation)

```
var a = 11, b = 22;
a = a ^ b;
b = a ^ b;
a = a ^ b;
console.log("a = " + a + "; b = " + b);// a is now 22 and b is now 11
```

Section 85.2: Faster multiplication or division by powers of 2

Shifting bits left (right) is equivalent to multiplying (dividing) by 2. It's the same in base 10: if we "left-shift" `13` by `2` places, we get `1300`, or `13 * (10 ** 2)`. And if we take `12345` and "right-shift" by `3` places and then remove the decimal part, we get `12`, or `Math.floor(12345 / (10 ** 3))`. So if we want to multiply a variable by `2 ** n`, we can just left-shift by `n` bits.

```
console.log(13 * (2 ** 6)) //13 * 64 = 832
console.log(13    <<   6)  //          832
```

Similarly, to do (floored) integer division by `2 ** n`, we can right shift by `n` bits. Example:

```
console.log(1000 / (2 ** 4)) //1000 / 16 = 62.5
console.log(1000    >>   4)  //            62
```

It even works with negative numbers:

```
console.log(-80 / (2 ** 3)) //-80 / 8 = -10
console.log(-80      >> 3)  //          -10
```

In reality, speed of arithmetic is unlikely to significantly impact how long your code takes to run, unless you are doing on the order of 100s of millions of computations. But C programmers love this sort of thing!

Section 85.3: Number's Parity Detection with Bitwise AND

Instead of this (unfortunately too often seen in the real code) "masterpiece":

```
function isEven(n) {
    return n % 2 == 0;
}

function isOdd(n) {
    if (isEven(n)) {
        return false;
    } else {
        return true;
    }
}
```

You can do the parity check much more effective and simple:

```
if(n & 1) {
    console.log("ODD!");
} else {
    console.log("EVEN!");
}
```

(this is actually valid not only for JavaScript)