**Bit Manipulation**

**Byte**

* A byte is a sequence of 8 bits.
* Hard Disk and RAM capacities are measured in bytes.
* A byte can have 2^8 values, ie 256 values, or values from 0 to 2^8-1, or 0 to 255.
* for an unsigned int, a byte can represent values from 0 to 255.
* for a signed int, a byte can represent values from -128 to 127.

**Basics of Boolean Algebra**

* XOR(Exclusive-OR) (this or that, but not both)
  + symbol is '^'
  + A ^ B will set the bits to 0 where A and B have same bits, and to 1 where the bits are different.
  + 0 ^ 0 = 0
  + 1 ^ 0 = 1
  + 0 ^ 1 = 1
  + 1 ^ 1 = 0
* Complement (not this)
  + symbol is '~'
  + flips the bits, 0 to 1, and vice versa.
  + ~0 = 1
  + ~1 = 0
* Left Shift
  + symbol is '<<'
  + x << y means x shifted y bits to the left.
  + If you run out of space, bits drop off from the left.
* Right Shift
  + symbol is '>>'
  + x >> y means x shifted y bits to the right.
  + If you run out of space, bits drop off from the right.

**Additional**

* XOR
  + for flipping selective bits, XOR is chosen.
  + for flipping a bit, XOR it with 1, it will get reversed.
  + N ^ 1 = ~N (bits flipped)
  + N ^ 0 = N (No change)
* NOT
  + The bitwise complement operator, ~, flips every bit in a number.

**Usages**

* N << 1 = N\*2
* N << 2 = N\*pow(2,2)
* N << k = N\*(pow(2,k))
* N >> 1 = floor(N/2)
* N >> 2 = floor(N/2^2)
* N >> k = floor(N/2^k)
* N & 1 = last bit in N
* N & 3 = last 2 bits in N
* N & 7 = last 3 bits in N
* N & (pow(2, k)-1) = last k bits in N
* ~N + 1 = -N
* N & 0xFF = least significant byte of integer or the last 8 bits of integer.
  + An Integer normally has 4 bytes(32 bits)
  + F in hex is 1111 in binary, so FF(or 0xFF) is 11111111 in binary
  + Doing N & 0xFF removes the first 3 bytes and only keeps the last byte(8 bits) of integer
  + Eg 1783 in binary is 11011110111
  + 1783 & 0xFF only keeps the last 8 bits of 11011110111, and is, 11110111, which is 247

**Example Interview Questions**

**Multiply a no by 2**

* left shift N by 1
* N << 1

**Divide a no by 2**

* right shift N by 1
* N >> 1

**set the kth bit of N(counting from right) to 1.**

* take 1, shift it k-1 places left, so its at kth place.
* do 'OR' with N.
* N = N | (1 << (k-1))

**clear the kth bit of N(counting from right).**

* take 1, shift it k-1 places left, so its at kth place.
* inverse it, so it becomes 0 and everything else is 1
* do 'AND' with N.
* N = N & ~(1 << (k-1))

**toggle/flip the kth bit of N(counting from right).**

* take 1, shift it k-1 places left, so it is at kth place.
* do 'XOR' with N.
* N = N ^ (1 << (k-1))

**turn off the first set bit(1 bit) of a number N.**

* N-1 will have all bits reversed before and including the first bit set.
* N & (N-1) will turn off the first set bit.
* N & (N-1)

**get the count of 1s in a no.**

* Idea: N & 1 gives the first bit.
* initialize count to 0;
* while N > 0
* count = count + (N & 1);
* N = N >> 1;
* end while
* return count;

Alternate method: suggested by [thevagabond85](https://disqus.com/by/thevagabond85/) in comments.

* count =0;
* while(n){
* n = n & (n-1); // turn off the first set bit(1 bit) of a number N.
* count++;

}

* return count;

**How to calculate the no of bits to convert from no A to no B.**

* for flipping a bit, XOR is chosen.
* A XOR B will give you 1 wherever the bits are different and you need to change bits.
* now we need to count the no of 1s in C, which is done by above method
* the result is no of set bits(1 bits) in A ^ B

**Check if N is a power of 2 or not.**

* if N is a power of 2, it will have only one 1, and rest all 0s.
* if N is a power of 2, N-1 will have all 1s, except that the 1 bit of N will be converted to 0.
* so, if N is a power of 2, then (N & (N-1) == 0)

**Check if N is a power of 4 or not.**

* check that N is a power of 2(from above)
* check that there are a total of even 0s in the binary representation of N.

**How to get the last 3 bits of an integer.**

* we just do AND with 111.
* N & ((1 << 3)-1)

**Get the 5 highest bits of an integer(8 bit integer).**

* for getting the 5 highest bits, we will remove the lower 3 bits and do 'AND' with 11111.
* we create 11111 by left shifting 1 by 5, 1 << 5, which gives 100000, and subtracting 1, which gives 11111.
* so we create 11111 by (1 << 5)-1
* we remove the lower 3 bits of x by (x >> 3)
* Doing 'AND' gives us the final answer.
* (x >> 3) & ((1 << 5)-1)

**check whether the kth bit in N is 1.**

* take 1, shift 1 by k-1 to the left, (1 << (k-1))
* Do N AND above, ie N & (1 << (k-1))
* The above result will have all digits 0, except the most significanr digit, which will be 0 if the nth bit in x is 0.
* So, if the nth significant bit in x is 0, then expression x & (1 << (n-1)) will be 0.
* So, if the nth significant bit in x is 1, then expression x & (1 << (n-1)) will not be 0.
* N & (1 << (k-1)) != 0

**swap two nos using bitwise operations.**

* A = A ^ B
* B = A ^ B
* A = A ^ B
* Note that (A ^ B) ^ B => (flips every bit of 'A' twice or zero times, essentially which means gives back 'B').
* Note that (A ^ B) ^ ((A ^ B) ^ B) => (A ^ B) ^ A => B => (flips every bit of 'B' twice, essentially giving back 'B')

**swap even and odd bits in a no(4 byte integer)**

* (N & 0xaaaaaaaa) gives the even bits(because a is 1010, so aa is 10101010, doing & with N, gives the even bits)
* (N & 0x55555555) gives the odd bits(because 5 is 0101, so 55 is 01010101, doing & with N, gives the odd bits)
* (N & 0xaaaaaaaa) >> 1 shifts even to odd bits
* (N & 0x55555555) << 1 shifts odd bits to even bits.
* ((N & 0xaaaaaaaa) >> 1) | ((N & 0x5555555555) << 1) gives us the required no where even and odd bits are swapped.

**Misc**

* IP address
  + normally represented as A:B:C:D
  + has 4 bytes, each of A, B, C, D representing a byte(8 bits).
  + each of A, B, C, D is 1 byte or 8 bits, and can have values from 0 to 255
* Right most bit(assuming 16 bit integer)???
  + N & 0xff;
* Left most bit(assuming 16 bit integer)???
  + (N>>8) & 0xff;
* Sign bit(assuming 16 bit integer)???
  + N & 0x8000;