Bit Manipulation

and Bitwise operators

Why Bit Manipulation

- Bit shifting is extremely useful in embedded applications, when memory is tight and speed is everything.
- deciphering the protocols of online games
- cryptographic methods
- image compression/decompression
- transposing the endian-ness of integers for cross-platform applications
- Unpacking the data from the bit-fields (many network protocols use them)
- manipulating bitmaps, for example changing the colour depth, or converting RGB <-> BGR
- games programming, when you need every last bit of performance.
- Interviews: Seen less frequently than other topics. Seen More often for more experienced candidates and for positions working with low level

Binary Review

- Binary: base-2 numeral system which represents numbers using 0 (zero) and 1 (one).
- In computers, signed number representations are required to encode negative numbers. Almost all computers in use today use a system called twos complement for signed int types.
- Twos Complement: negative numbers are represented by inverting the absolute value and add 1 (one).
- Contrast with unsigned representation (see table)

Bits	Unsigned value	Two's complement value
0111 1111	127	127
0111 1110	126	126
0000 0010	2	2
0000 0001	1	1
0000 0000	0	0
1111 1111	255	−1
1111 1110	254	−2
1000 0010	130	−126
1000 0001	129	−127
1000 0000	128	−128

Binary Review

- Since standard C does not have a notation for entering binary numbers, using hex is the easiest way. Setting single bits in hex is relatively easy, like so:

Bit pattern	Hex
00000000	0x00
00000001	0x01
00000010	0x02
00000100	0x04
00001000	0x08
00010000	0x10
00100000	0x20
01000000	0x40
10000000	0x80

Bit Operations

- & (Bitwise And)
  - note: single ampersand

  Are they both true?

&	0	1
0	0	0
1	0	1

- | (bitwise Or)

  Is at least one true

|	0	1
0	0	1
1	1	1

Bit Operations

- ~ (bitwise not)
  invert bits

~	0	1
	1	0

^	0	1
0	0	1
1	1	0

- ^ (bitwise XOR - exclusive Or)
  Are they different?

Bit Operations

- << left shift - shift all bits to the left, filling the right most bits with 0 (zero)

10110101	-75
01101010	106

- >> arithmetic right shift - shifts in bits that match original left-most bit, preserving sign

10110101	-75
11011010	-38

Bit Operations

- >>> (logical) right shift - fills leftmost bits with 0 (zero) - does not preserve sign of negative numbers. (Not all languages provide this.)

10110101	-75
01011010	90

Bit Facts

x ^ 0s = x	Xor a value with zeros gives same value
x ^ 1s = ~x	Xor a value with ones gives inverse
x ^ x = 0	XOR a value with itself gives zero
x & 0s = 0	And a value with zeros gives zero
x & 1s = x	And a value with ones gives the value
x & x = x	And a value with itself gives the value
x | 0s = x	Or a value with zeros gives the value
x | 1s = 1s	Or a value with ones gives ones
x | x = x	Or a value with itself gives the value

Bit Facts

- -1 in 2s complement binary is all ones, i.e. 8bit values, -1 is 11111111
- binary representation of (x-1) can be obtained by simply flipping all the bits to the right of rightmost 1 in x and also including the rightmost 1
- Size of Ints in 2s complement representation
  - $-2n-1 \le$ Two's Complement $\le 2n-1 - 1$
  - $-128 \le x[8] \le +127$
  - $-32768 \le x[16] \le +32767$
  - $-2147483648 \le x[32] \le +2147483647$

Bit Tasks - Get Bit

```
boolean getBit(int num, int i) {
  return ((num & ( 1 << i)) != 0);
}
```

- Set a value to all zeros except for the bit of interest, by left shifting one by the bit position
- 'And' this shifted value with num
- If the result is not zero, the bit is set.

Bit Tasks - Set Bit

```
boolean setBit(int num, int i) {
  return num | (1 << i);
}
```

- Set a value to all zeros except for the bit of interest, by left shifting one by the bit position
- 'Or' this value with num and return

Bit Tasks - Clear Bit

```
boolean clearBit(int num, int i) {
  int mask = ~(1 << i);
  return num & mask;
}
```

- Set a value to all zeros except for the bit of interest, by left shifting one by i.
- Invert the value so that it's all ones except for the bit of interest
- 'And' this value with num and return

Bit Tasks - Clear Bit

- To clear all bits from the most significant bit through i, inclusive

```
boolean clearMSBThroughI(int num, int i) {
  int mask = (1 << i) -1;
  return num & mask;
}
```

- Set a value to all zeros except for the bit of interest, by left shifting one by the position of the bit. i.e. 00001010
- Subtract 1 from it (i.e. add the inverse + 1, (1111110 + 1), or 1111111 = so 1111111
- 'And' this value with num (i.e. 00001010 & 1111111 = 00001010) return result

Bit Tasks - Clear Bit

- To clear all bits from i, through 0

```
boolean clearIThrough0(int num, int i) {
  // replace 31 by (sizeof int) - 1
  // with 8 bit numbers, use 7
  int mask = ~(-1 >>> (31 - i));
  return num & mask;
}
```

- take inverse of logically right shift -1 by the difference between 7 and the position of interest, i.e. if i = 3, 7-3 = 4 so ~(-1 >>> 4) = ~(11111111 >>> 4) = ~(00001111) = 11110000
- 'And' this value with num (i.e. 00101010 & 11110000 = 00100000) and return result

Bit Tasks - Update Bit

- To set the ith bit of a num, first clear the bit, using the mask mask, then 'Or' it with the value, then 'Or' it with the requested bit.

```
boolean updateBit(int num, int i,
boolean bitIs1) {
  int value = bitIs1 ? 1 : 0;
  int mask = ~(1 << i));
  return (num & mask) | (value << i);
}
```

Classic Hacks

- Swapping values without using a temporary variable. For years, the XOR swap has served as an example of bit twiddling. Nowadays, its performance advantage is completely gone, but it's an interesting hack.
  - `#define SWAP(a, b) (((a) ^= (b)), ((b) ^= (a)), ((a) ^= (b)))`

Classic Hacks

Reversing Bits

```
unsigned int v;     // input bits to be reversed
unsigned int r = v; // r will be reversed bits of v; first
get LSB of v
int s = sizeof(v) * CHAR_BIT - 1; // extra shift needed at
end

for (v >>= 1; v; v >>= 1)
{
  r <<= 1;
  r |= v & 1;
  s--;
}
r <<= s; // shift when v's highest bits are zero
```

Classic Hacks

Check if an integer is even or odd

```
if ((x & 1) == 0) {
  x is even
}
else {
  x is odd
}
```

Check if an integer is a power of two

```
bool isPowerOfTwo(int x)
{
    // x will check if x == 0 and !(x & (x - 1)) will check if x is a
power of 2 or not
    return (x && !(x & (x - 1)));
}
```

Isolate the rightmost bit

```
y = x & (-x)
```

Classic Hacks

Return the rightmost 1 in the binary representation of x

```
x ^ ( x & (x-1))
```

Check if an integer is a power of two

```
bool isPowerOfTwo(int x)
{
    // x will check if x == 0 and !(x & (x - 1)) will
check if x is a power of 2 or not
    return (x && !(x & (x - 1)));
}
```

Isolate the rightmost bit

```
y = x & (-x)
```

Resources

- Bit Twiddling Hacks https://graphics.stanford.edu/~seander/bithacks.html
- Two's Complement https://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Data/twoscomp.html
- Binary Arithmatic https://www.cs.tcd.ie/~waldroj/3d1/04-Arithmetic.pdf