Bit Manipulation

Get Bit:

- *This operation SHIFTS 1 over by i bits, creating a value that looks like 00010000. By performing an AND with num, we clear all bits other than the bit at i-th position. Finally, we compare that to 0. If that new value is not zero, then bit i must have a 1. Oherwise, bit i is a 0.*

```
num & (1 « i)
```

```
num:                    01101001
1 << 4:                 00010000
--------------------------------
num & (1 << 4):         00000000
```

- *The following operation isolate the lowest (rightmost) set bit of **num.** The method returns an integer which is the index of the rightmost set bit of **num.***

```
num & ~(num-1)
```

```
num:                    01101000
1:                      00000001
--------------------------------
num-1:                  01100111
--------------------------------
~(num-1):               10011000
num:                    01101000
--------------------------------
num & ~(num-1):         00001000
```

- *The following method isolate the lowest (rightmost) unset bit of **num.** . The method returns an integer which is the index of the rightmost unset bit of **num.***

```
~num & (num+1)
```

```
num:                    01100111
1:                      00000001
--------------------------------
num+1:                  01101000
~num:                   10011000
--------------------------------
~num & (num+1):         00001000
```

Set Bit:

- *Set Bit shifts 1 over by i bits, creating a value like 00010000. By performing an **OR** with num,only the value at bit i will change. All other bits of the mask are zeroand will not affect num.*

SEP

```
num | (1 « i)
```

```
num:                    01101001
1 << 4:                 00010000
-------------------------------
num | (1 << 4):         01111001
```

Clear Bit:

- *This method operates in almost the reverse of setBit. First,we create a number like 11101111 by creating the reverse of it (00010000) and negating it. Then, we perform an AND with num. This will clear the ith bit and leave the remainder unchanged.*

num & ~ (1 << 3)

```
num:                    01101001
1 << 3:                 00001000
-------------------------------
~(1 << 3):              11110111
num:                    01101001
-------------------------------
num & ~(1 << 3):        01100001
```

- *The following method clear the lowest set bit in **num***

```
num & (num-1)
```

```
num:                    01101000
1:                      00000001
-------------------------------
num-1:                  01100111
num:                    01101000
-------------------------------
num & (num-1):          01100000
```

- *To clear all bits from the most significant bit through i **(inclusive)**, we do:*

```
num & (1 << i) – 1
```

```
num:                          01101001
------------------------------------
1 << 4:                       00010000
1:                            00000001
------------------------------------
(1 << 4) – 1:                 00001111
num:                          01101001
------------------------------------
num & (1 << 4) – 1:           00001001
```

- *To clear all bits from i through 0 **(inclusive)**, we do:*

```
int clearBitsIThrough0(int num, int i) {
     int mask = ~((1 « (i+1)) – 1);
     return num & mask;
}
```

```
num:                          01101001
------------------------------------
1 << 4+1:                     00100000
1:                            00000001
------------------------------------
(1 << 4+1) – 1:               00011111
------------------------------------
~(1 << 4+1) – 1:              11100000
num:                          01101001
------------------------------------
num & ~((1 << 4+1) – 1):01100000
```

Update Bit:

- *This method merges the approaches of **setBit** and **clearBit**. First, we clear the bit at position i by using a mask that looks like11101111. Then, we shift the intended value, v, left by i bits. This will create a number with bit i equal to v and all other bits equal to 0. Finally, we Or these two numbers, updating the ith bit if v is1and leaving it as 1 otherwise.*

```
int updateBit(int num, int i, int v) {
     int mask = ~(1 « i);
     return (num & mask) | (v « i);
}
```

```
num:                                  01101001
v:                                    00010010
--------------------------------------------
1 << 4:                               00010000
~(1 << 4):                            11101111
num:                                  01101001
--------------------------------------------
num & (~(1 << 4)):                    00011111
v << 4:                               00100000
--------------------------------------------
(num & (~(1 << 4))) | (v << 4): 00111111
```

- *Addition using ONLY Bitwise Operator without using any Arithmatic Operator*

```
static long add(long x, long y){

        while (y != 0) {
            long carry = x & y;
            x = x ^ y;
            y = carry << 1;
        }
        return x;
    }
```

X = 0111; Y = 0110;
X + Y = 1101

0111 (&) 0110 ---------- 0110	0111 (^) 0110 ---------- 0001	Y = 1100
1100 (&) 0001 ---------- 0000	0001 (^) 1100 ---------- 1101	Y = 0000

```
static long multiply(long x, long y) {
    long sum = 0;
    while (x != 0) {
        if ((x & 1) != 0) sum = add(sum, y);
        x >>>= 1;
        y <<= 1;
    }
    return sum;
}
```

Y = 0111; X = 0110; Y × X = 101010

0 1 1 1
(×) 0 1 1 0
---
0 0 0 0
0 1 1 1 ×
0 1 1 1 × ×
0 0 0 0 × × ×
---
0 1 0 1 0 1 0

let, x is a int & n is wordlength of significant digit in x. Then,

- $n = floor(log_{10}x) + 1$
- $\frac{y=x}{x^{n-1}}$: *Extract the MSD of x*
- $y = x \,\%\, 10$: *Extract the LSD of x*
- $x = x \,\%\, 10^{n-1}$: *Remove MSD of x*
- $x = \frac{x}{10}$: *Remove LSD of x.*

Example: *let* $x = 157793$ *and* $z = 099910$

- $n_x = 5 + 1 = 6$; $n_z = 4 + 1 = 5$
- $\frac{x}{x^{n_x-1}} = 1$; $\frac{z}{z^{n_z-1}} = 9$
- $x \,\%\, 10 = 3$; $z \,\%\, 10 = 0$
- $x = x \,\%\, 10^{n-1} = 57793$;
- $z = z \,\%\, 10^{n-1} = 9910$
- $x = \frac{x}{10} = 15779$; $z = \frac{z}{10} = 9991$

- $1111\ 1111\ 1111\ 1111 = 0xFFFF = 65535 = 2^{16} - 1$
- $1\ 0000\ 0000\ 0000\ 0000 = 0x10000 = 65536 = 2^{16}$
- $1010\ 1010 = 0xAA$
- $0101\ 0101 = 0x55$
- $\sim((1 << 4) - 1) = 111110000$

Bit Facts and Trick:

Bitwise: XOR (^)	*Bitwise: AND (&)*	*Bitwise: OR (|)*
X ^ X = 0s: 0 1 1 0 (XOR) 0 1 1 0 ---------- 0 0 0 0	X & X = X: 0 1 1 0 (AND) 0 1 1 0 ---------- 0 1 1 0	X | X = X: 0 1 1 0 (OR) 0 1 1 0 ---------- 0 1 1 0
X ^ (~X) = 1s: 0 1 1 0 (XOR) 1 0 0 1 ---------- 1 1 1 1	X & (~X) = 0s: 0 1 1 0 (AND) 1 0 0 1 ---------- 0 0 0 0	X ^ (~X) = 1s: 0 1 1 0 (OR) 1 0 0 1 ---------- 1 1 1 1
X ^ (0s) = X : 0 1 1 0 (XOR) 0 0 0 0 ---------- 0 1 1 0	X & (0s) = 0s : 0 1 1 0 (AND) 0 0 0 0 ---------- 0 0 0 0	X ^ (0s) = X : 0 1 1 0 (OR) 0 0 0 0 ---------- 0 1 1 0
X ^ (1s) = ~X : 0 1 1 0 (XOR) 1 1 1 1 ---------- 1 0 0 1	X & (1s) = X : 0 1 1 0 (AND) 1 1 1 1 ---------- 0 1 1 0	X ^ (1s) = 1s : 0 1 1 0 (OR) 1 1 1 1 ---------- 1 1 1 1