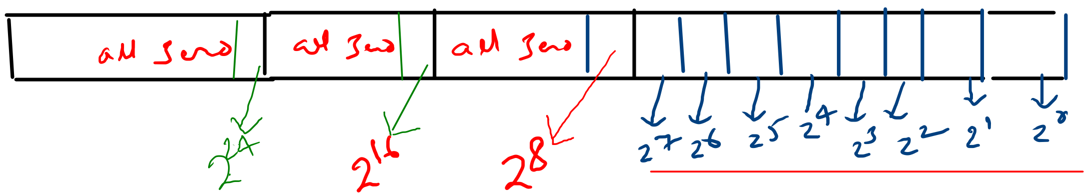
"Bitwise Utilities"

Understanding number of leading & trailing zeros in an integer.

In Java integer is 32 bits i.e 4 bytes.

e.g. $x = 83 \rightarrow$ lies between $2^{6} \left(\lfloor \log_2 83 \rfloor\right)$ & $2^{7} \left(\lceil \log_2 83 \rceil\right)$



as we know
$$\sum_{i=0}^{n} 2^n = 2^{n+1} - 1$$

So, 2^8 bit position is always greater than the number formed by using bit positions 0 to 7.

If $x < 2^8 \longrightarrow x$ lies in first byte

If $x >= 2^{24} \longrightarrow x$ spans all 4 bytes

If $x >= 2^{16} \longrightarrow$ first 2 bytes are not enough to accommodate x, means x doesn't completely lie in first 2 bytes.

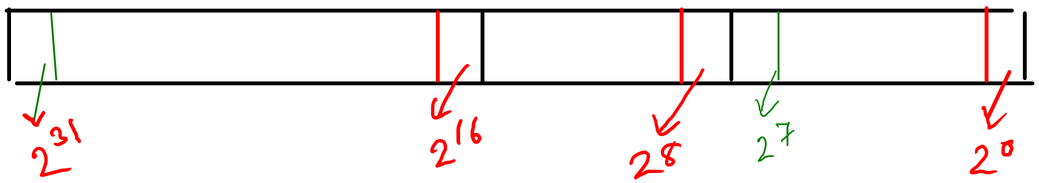
If $x >= 2^8 \rightarrow x$ not completely lie in 1st byte (0 to 7)

If $x >= 2^4 \rightarrow x$ not completely lie in 1st 4 bits (0 to 3)

If $x >= 2^2 \rightarrow x$ not completely lie in 1st 2 bits (0 to 1)

If $x >= 2^1 \rightarrow x$ not completely lie on 0th bit

Q Why 16 bits window is enough to inspect leading or trailing zeros of 32 bits number ?



[$x >= 2^{16}$ → x doesn't lie completely in first 16 bits]

We will start our inspection at $x >= 2^{16}$ because once we get to know x doesn't lie completely in first 2 bytes (means can have at max 31−16 = 15 leading zeros) then we need to inspect 3rd & 4th byte, this can be easily done by shifting 3rd & 4th byte to 1st & 2nd byte position.

So, checks will be

(i) $x >= 2^{16}$ or $x >= (1 << 16)$

(ii) $x >= 2^{8}$ or $x >= (1 << 8)$

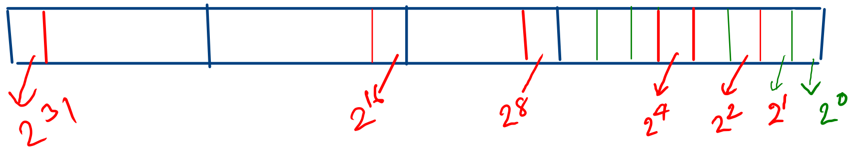
(iii) $x >= 2^{4}$ or $x >= (1 << 4)$

(iv) $x >= 2^{2}$ or $x >= (1 << 2)$

(v) $x >= 2^{1}$ or $x >= (1 << 1)$

x = given number, Max_index = 31

initialize LeadingZeros = 31



Algo: If $x >= 2^{16}$: → means max leading zeros may be $31 - 16 = 15$

→ and further need to inspect 3rd & 4th byte

→ zero fill right shift 16 so that 3rd & 4th bytes comes at 2nd & 1st byte position. → To get an inspection window of 16 bits

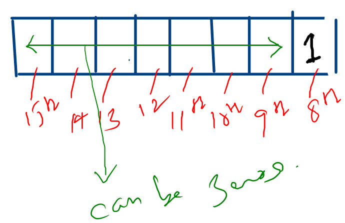
→ $x >>> 16$

then

step2: If $x >= 2^{8}$: Here we might be inspecting a fresh number or $x >>> 16$ shifted bytes of step 1 that completely lies in 1st 2 bytes.

since x is $>= 2^{8}$, so x spans 1st byte completely and span portion of 2nd byte is unknown.

so, max leading_zeros = $15 - 8 = 7$



→ now, we shift the $x >>> 8$

Similarly we do for $x >= 2^{4}$, $x >= 2^{2}$ & $x >= 2^{1}$.

$$\boxed{\sim n \,\&\, (n-1)}$$ → makes trailing zero bits as 1 and all other bits as zero

Understanding calculations of trailing bits.

Prerequisite:

Odd number cannot have trailing zeros, as last bit of odd number is always 1. So only even numbers can have trailing zeros.
We can convert the problem of trailing zeros, to the problem of leading zeros.

152 =

7	6	5	4	3	2	1	0
1	0	0	1	1	0	0	0

→ even number

no. of trailing zeros.

lowest 1 bit position

If we substract 1 from an even number all the trailing zeros becomes 1.

152 - 1 = 151

all trailing zeros are getting replaced by 1.

151 =

1	0	0	1	0	1	1	1

&

0	1	1	0	0	1	1	1

← One's complement of 152

now, all these bit positions are zero

→ result [0, 0, 0, 0, 0, 1, 1, 1]

This can be considered as problem of leading zeros.

Calculating trailing zero bits

$[\sim n \,\&\, (n-1)]$ → sets only the trailing bit position and makes others to zero.

→ Now we can apply two strategies to calculate set bits:

1. count the set bits

Integer.bitcount$(\sim n \,\&\, (n-1))$

||

Number of trailing zeros.

2. 32 − Integer.numberOfLeadingZeros$(\sim n \,\&\, (n-1))$