BITWISE Operators :-

Bitwise OR : (|)

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Def: It's a binary operator, It returns bit by bit OR of input values and The result of Manupulation is shown in the preceding table.

ex: 5 | 7 → 00000101 OR 00000111
= 00000111 = 7 ↓ Bit in decimal

notes: Every bitwise operator is a binary operator which is making binary bit operations not as usuall.

Bitwise AND : (&)

A	B	A & B
0	0	0
1	0	0
0	1	0
1	1	1

ex: 5 & 7 ⟶ 00000101 AND 00000111
= 00000101 = 5 in decimal

Bitwise XOR: (^)

A	B	A^B
0	0	0
1	0	1
0	1	1
1	1	0

ex: 5 ^ 7 ⟶ 00000101 XOR 00000111
= 00000010 = 2 in decimal

Bitwise Complement: (~)

A	~A
0	1
1	0

ex: ~5 ⟶ ~~NOT~~ Complement 00000101
= 11111010 = -6 in decimal
(Sign (+, -))

note: The last bit in any numeral Type is storing The sign of The number if it is negative number or positive number.

Shifting Operators :-

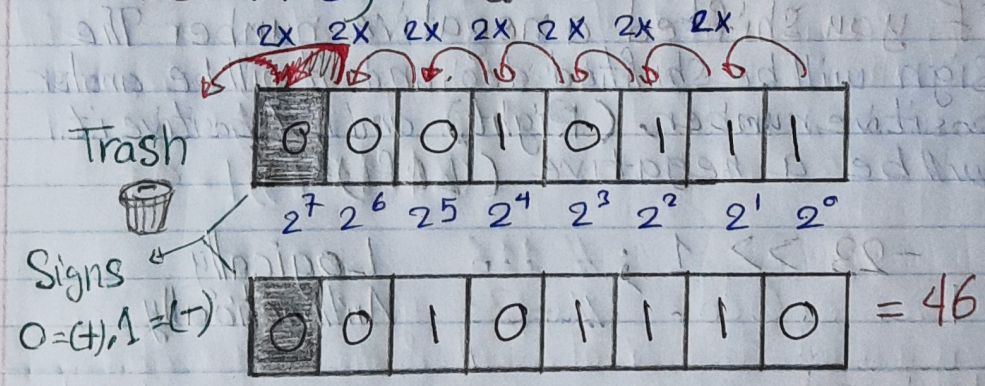
Signed Left Shift: (<<)

Def: The left shift operator move all bits by a given number of bits to the left and we can write it as following Syntax :

operand → num1 << num2; ← Amount of shifting Bits

ex:

23 << 1; → 1 bit = x2, a bit = x2a

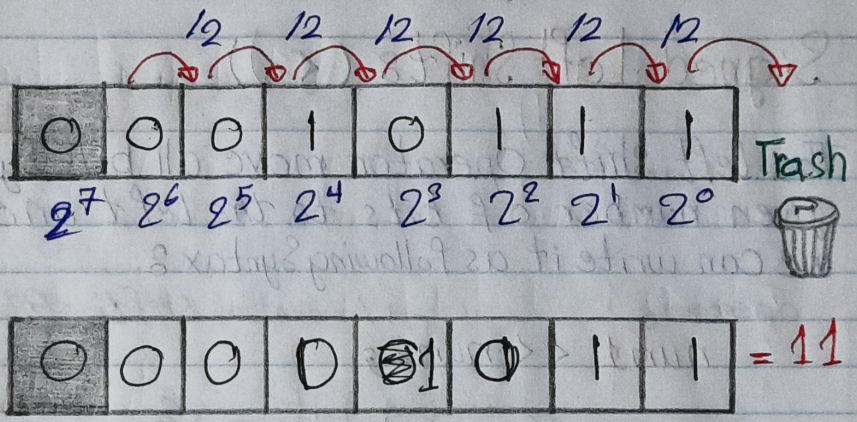


Signed Right Shift: (>>)

Def: The ~~Left~~ Right Shift Operator move all bits by a given number of bits to the Right and we can write it As following Syntax :

num1 >> num2;

ex: 23 >> 1; → 1 bit = /2
a bit = /02



note: if you shifted a negative number The Sign will be shifted too and it will be anoter positive number, (Right) and if it positive it will be a negative (Left)

ex: -23 >> 1 ; // 116 Logically
// -12 Arithmetically

~~Unsigned~~

Unsigned Left Shift (<<<)

Def: Unlike unsigned Right Shift, There is no "<<<" Operator in java because its identical To "<<" Operator Logically and arithmetically.

UnSigned Right Shift (>>>)

Def: it is the same as the signed right shift, but the vacant leftmost position is filled with 0 instead of the sign bit.

BitMasks :-)

Introduction To Bits :

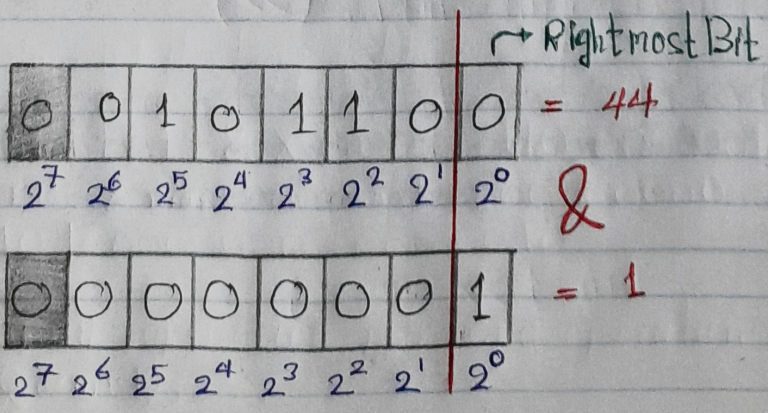
In This part we want to Control on every bit in any single byte. and we will learn Some Operations on bits.

Get ith Bit :

Def: Suppose you want to make indication on Some bit we will use the & Bitwise AND operator, Cause it's only need activated bit from the byte, so all The bits must be 0. ~~(x & (1<<i)) != 0~~

ex:

Even or Odd :- Suppose you have a number you want to indicate wether its even or odd. you only want to check on the Rightmost bit by anding it with 1 :



The result would be 0, Cause the result of anding The right most bits is 0 So 44 is even number.

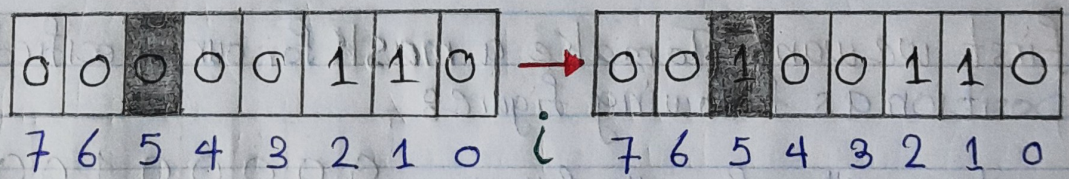
5 & 1 = 1 odd number
2 & 1 = 1 even number

Modify Bits:

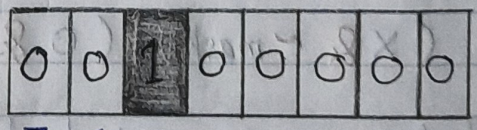
1 Set ith Bit:-

Def: is to change value of a cleared bit or 0's bit to 1 Bit

ex:



first we want to make a mask for the ith position as following figure:



and we will perform The following statement

```
mask = 1 << i;
```

Code:

```
(X & ~mask) | (-1 & mask);
```

let me explain what's happend here:

~mask 11011111 & X 00000110 = 00000110
-1 11111111 & mask 00100000 = 00100000
00000110 | 00100000 = 00100110 .

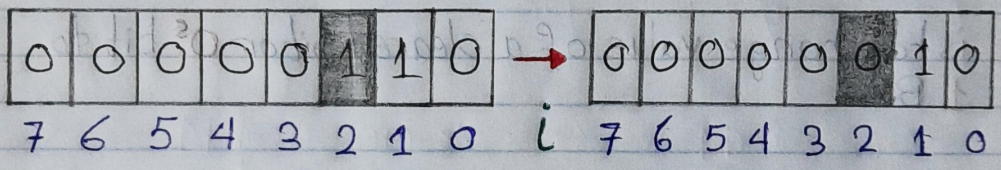
note: we can add another simple Syntax To perform This operation :

```
(X | (1&1<<i)) ;
```

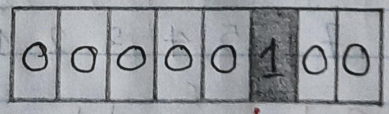
2 Clear ith Bit :-

Def: Is to change a value of 1 Bit to a 0 Bit

ex:



first we want To make a mask for our specified position as following figure:



mask = 1 << i ;

and we will perform the following statement

Code: `(X & ~mask) | (0 & mask)`

Let me explain what's happend here :

~mask 11111011 & X 00000110 = 00000010
0 00000000 & mask 00000010 = 00000000
00000010 | 00000000 = 00000010

note: we can modify The operations into one operation.

```
(X & ~mask);
```

Examples on BitWise Operations :

1. Check IF power of 2 :-

   $(X \& X-1) == 0$

   → 4 00000100 & 3 00000011 = 00000000
   So 4 is power of 2.

   → 5 00000101 & 4 00000100 = 00000100
   So 5 isn't power of 2.

2. number with odd occurence :-

```
int[] arr = { 2,3,5,4,5,2,4,3,5,2,4
              ,4,2};

int result;
for(int i=0; i < arr.length; i++)
    result ^= arr[i];
return result;
```

3. Counting number of different bits between 2 ints :-

```
int num1 = X, num2 = Y; int count = 0;
int num = num1 ^ num2;
while (num) {
    Count += num & 1; T. Complexity O(log n)
    num >>= 1;
}
return count;
```