Bitwise Operators:

- They work on bits

The concept of signed and unsigned:

When we declare a varible as an int, it is a signed integer. What does this mean?

This means that the integer can be positive or negative which makes it signed while an unsigned integer only has positive numbers. But How does it make it signed?

- if we see the range of signed its is range of signed = -2^{31} to $(2^{31}-1)$

range of unsigned = $(2^{32}-0)$

There is one less bit in signed!!!

That one bits defines the the sign:

0 = +ve

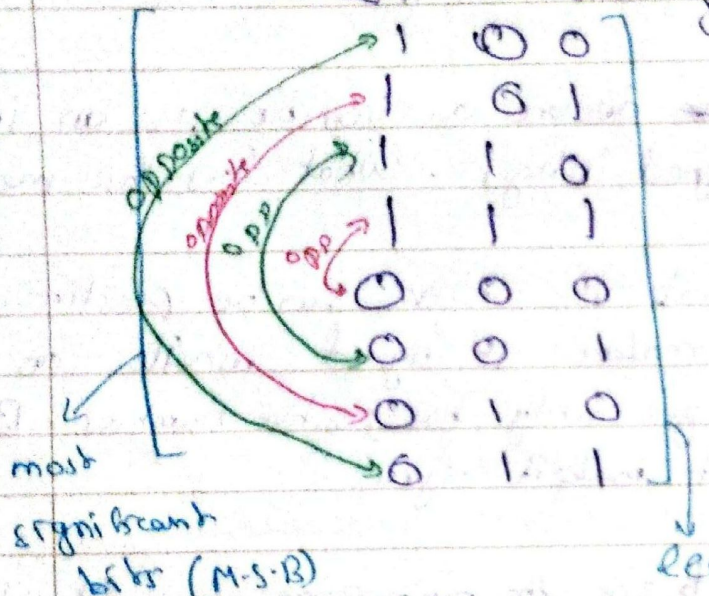
1 = -ve.

bitwise only work with integers not with floats.

Yousaf

Date: / /

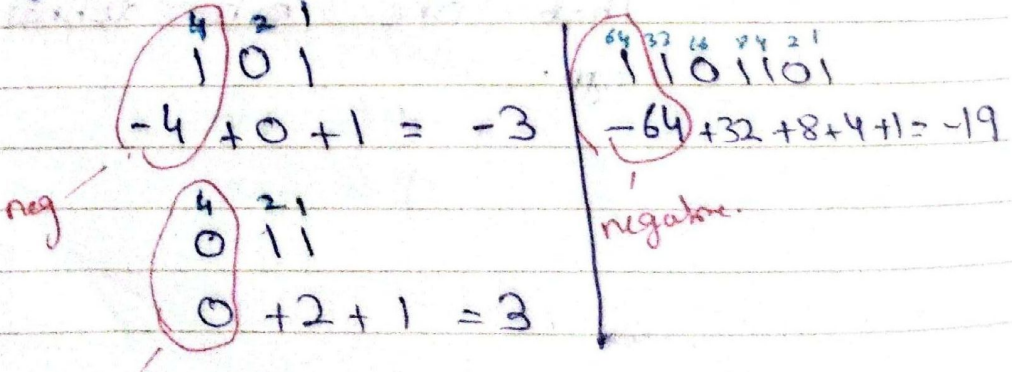
- 32 bits can be way too long to write so I am considering for 3 bit.



- The M.S.B define the sign.

Just for remembering you can take the M.S.B as a negative number while all other positive.

e.g



Yousaf

Day: Date: / /

Bitwise operators:

1) Not (~):

- This reverses all the bits
- Unary operator.

e.g i) ~(1101) = 0010

2) ~5

as 5 is an int so all 32 bits will be 0.

5 = 0000 0101

~5 = -6

= ---- 1111010

all bits reversed, this gives answer as -6.

signed has 32 bits

3) ~12

= -13

How did 5 become -6.

now if we add up all the bits in -6.

$= -2^{31} + 2^{31} + 2^{30} + \cdots 2^3 + 2^1$

$= -6$

so general formula:

$$\sim a = -(a+1)$$

Yousaf

Day: Date: / /

2) OR operator (|):

A	B	Output
0	0	0
1	0	1
0	1	1
1	1	1

- Binary operator.
- This checks each bit with it corresponding bit

e.g 1) 5 | 3

```
5 = 0 0 0 1 0 1
3 = 0 0 0 0 1 1
Answer 0 0 0 1 1 1  = 7
```

5 | 3 = 7

2) 12 | 4

```
12 = 0 0 1 1 0 0
 4 = 0 0 0 0 1 0
Answer 0 0 1 1 1 0  = 12
```

12 | 4 = 12

Day: Date: / /

3) AND operator (&):

A	B	output
0	0	0
1	0	0
0	1	0
1	1	1

- Binary operator
- checks each bit with corresponding bit.

e.g 1) 5 & 3

5	=	0	0	0	1	0	1
3	=	0	0	0	0	1	1
Answer:		0	0	0	0	0	1 = 1

5 & 3 = 1

2) 12 & 4

12	=	0	0	1	1	0	0
4	=	0	0	0	1	0	0
Answer	=	0	0	0	1	0	0 = 4

Yousaf

Day: Date: / /

4) XOR operator (^)

A	B	output
0	0	0
1	0	1
0	1	1
1	1	0

* - Exclusive OR.
- If both operands same than 0, if both different than 1.

e.g.1) 5 ^ 3

5 =	0	0	0	1	0	1	
3 =	0	0	0	0	1	1	
Answer	0	0	0	1	1	0	= 6

2) 12 ^ 4

12 =	0	0	1	1	0	0	
4 =	0	0	0	1	0	0	
Answer =	0	0	1	0	0	0	= 8

Yousaf

Day: Date: / /

5) left shift operator (<<)

- binary operator
- It shifts the bits by a number of times.

binary number << number of times to shift.

e.g 1) 15 << 2

15 = 0 0 0 0 1 1 1 1
one time shift
= 0 0 0 1 1 1 1 0
Two time shift
= 0 0 1 1 1 1 0 0 = 60

left shift multiplies the number by 2 as power of 2 increases.

* If 15 was an int, int only has 32 bits. Now if would have shifted it 32 times or more the value will become zero as all ~~the~~ one's are thrown out.

** But if 1 only shifted it 31 times then the number would have become a Negative with the greatest value ie 10000 ... 31 times.
I told consider this negative $-2^{32} + 0 + 0 + 0 \ldots$ so it will greatest pasible - ve number.

Yousaf

(6) Right Shift (>>)

Same as left shift but this shifts the bits to right.

e.g 1) 13 >> 2

13 = 0 0 0 0 1 1 0 1

first shift

= 0 0 0 0 1 1 0 = 6

second shift

= 0 0 0 0 0 1 1 = 3

Right shift divides by 2 and if the answer is 6.5 then it is 6.

Common mistakes in shifting:

```
1) int a = 5;
2) cout << a << 2 << endl;
```

we want to shift here

Output:

527 → no shifting, it just showed the numbers.

Instead:

```
1 | int a=5
2 | cout << (a<<2) << endl;
```

output

20 ← correct way.

Some questions related to Bitwise:

Q: Check if a number is even or odd?

If we look at binary, a number will be even or odd depending on its L.S.B.

eg 1 = 0001 odd have 1 as L.S.B
2 = 0010 even have 0 as L.S.B
3 = 0011
4 = 0100

Code:

```
1  int main(){
2      int a;
3      cin>>a;
4
5      int mask=1;
6
7      if(mask & a ==0){
8
9          cout<<"Even"<<endl;}
10     else{
11         cout<<"odd"<<endl;
12     }
13 return 0;
14 }
```

checking code with

1) 3 = 0011

3 = 0011
a = 0001 &
0001, so odd.

2) 38
38 = 0100110
a = 0000001 &
0000000
so even

Day: Date: / /

Q Turning any bit on? (ie converting it to one)

We can turn the desired bit on by taking an OR with one with that bit.

e.g
00101101 , we need to turn the 5th bit on

so we can

```
  0 0 1 0 1 1 0 1
  0 0 0 1 0 0 0 0   |
  ---------------
  0 0 1 1 1 1 0 1
```

Code:

```
1  Int main() {
2    int num = 0, bit = 0, mask = 0;
3
4    cout << " Enter number: ";
5    cin >> num;
6
7    cout << " Which bit do you want to turn on? ";
8    cin >> bit;
9
10   mask = 1;
11   mask = (mask << (bit-1)); //also can write like mask << = bit-1
12
13   num = num | mask; //also can write like num |= mask
14
15     cout << num << endl;
16   return 0;
17 }
```

Q Turning a bit off

Turning the bit off is a little different. Because you cannot take (OR) with 0 or it will still result in 1. You cannot take the whole number & with 0 or this will make the whole number 0.

Solution:

You can only for the bit to change e.g 3rd bit
The same process will be followed as we need to turn on the bit.

- (0001) << (3-1)
- 00100

Here we will take negation (~) and negate the mask.

- ~(00100)
- 11011

Now we can take and(&) with the number entered

Day: Date: / /

Q: Toggle / Flipping a bit?

- We can use a (~) to flip every bit but How can we flip only a particular bit?

  - We can use XOR (^) with 1.
  - If a bit is 0 it become 1 and if a bit is 1 it becomes 0.

code:

```
1  int main(){
2      int num=0, mask=0, bit=0;
3
4      cout << "Enter a number: ";
5      cin >> num;
6
7      cout << "Which bit to turn on? ";
8      cin >> bit;
9
10     mask = 1;
11     mask <<= bit-1;
12
13     num ^= mask;
14     cout << num << endl;
15 return 0;
16 }
```

Page: Date: / /

Q Swapping two numbers using bitwise:

```
int main() {

    int num1, num2;

    cout << "Enter first number: ";
    cin >> num1;
    cout << "Enter second number: ";
    cin >> num2;


    num1 = num1 ^ num2;
    num2 = num1 ^ num2;
    num1 = num1 ^ num2;

    cout << "Swapped numbers \n1. " << num1 << "\n2. "
         << num2 << endl;

    return 0;
}
```