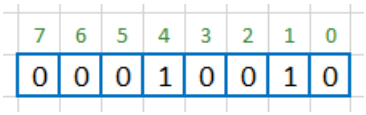
Битовые (побитовые, поразрядные) операции

Напомним, что в C++ имеется 3 вида логических операций:

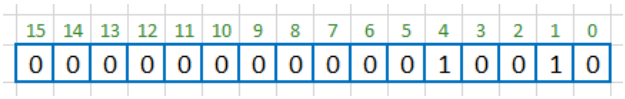
Логические операции	
&&	и
||	или
!	отрицание

Все числа в памяти компьютера хранятся в *двоичной системе счисления*. Например, целое число 18 имеет двоичное представление в зависимости от того, сколько байтов оно должно занимать в памяти:

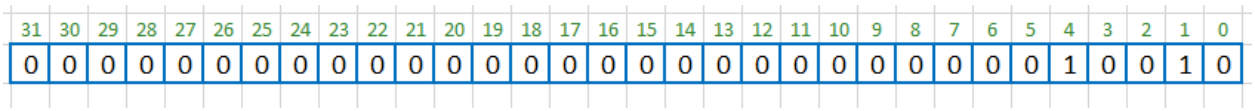
а) один байт (тип char)



б) два байта (тип short int)



в) четыре байта (тип int)



Битовые операции по смыслу похожи на логические, *но применяются в каждой позиции* двоичного разложения отдельно. Они определены ***только для целочисленных типов*** и не могут применяться к вещественным числам.

Битовые операции	
&	и
|	или
~	отрицание
^	исключающее или
>>	сдвиг вправо
<<	сдвиг влево

Применение битовых операций:

- проверка, сброс и установка отдельных битов в составе целого числа
- экономия памяти (упаковка нескольких логических значений в одну целую переменную)
- повышение производительности программ (битовые операции могут выполняться одновременно над всеми разрядами в отличие от арифметических операций, где нужно учитывать перенос)
- работа с регистрами аппаратуры

Таблицы истинности битовых и обычных логических операций И, ИЛИ, НЕ совпадают. Отличие лишь в том, что битовые операции выполняются над отдельными битами операндов. Операция «исключающее ИЛИ» имеет следующую таблицу истинности:

x	y	x^y
0	0	0
0	1	1
1	0	1
1	1	0

Если соответствующие биты операндов одинаковы, то результат операции «исключающее ИЛИ» равен нулю, иначе – единице.

Пример 1. Битовая операция И.

Переменная	Значение переменной	Двоичное представление
x	14	0 0 0 0 1 1 1 0
y	11	0 0 0 0 1 0 1 1
z=x&y	10	0 0 0 0 1 0 1 0

```
int x = 14, y = 11, z;
z = x&y;
cout << z << endl;     // 10
```

Пример 2. Битовая операция ИЛИ.

Переменная	Значение переменной	Двоичное представление
x	10	0 0 0 0 1 0 1 0
y	9	0 0 0 0 1 0 0 1
z=x|y	11	0 0 0 0 1 0 1 1

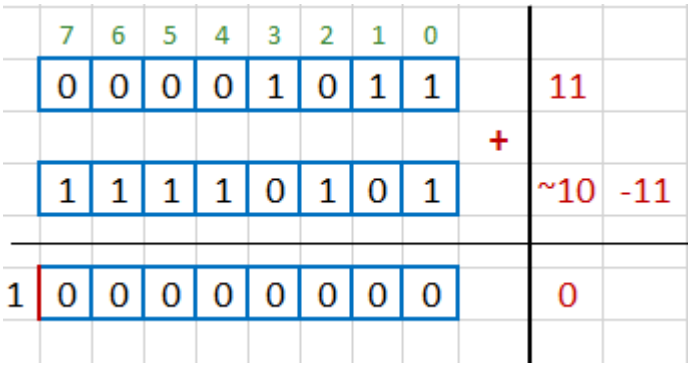
```
int x = 10, y = 9, z;
z = x|y;
cout << z << endl;     // 11
```

Пример 3. Битовая операция НЕ.

Переменная	Значение переменной	Двоичное представление
x	10	0 0 0 0 1 0 1 0
y=~x	?	1 1 1 1 0 1 0 1

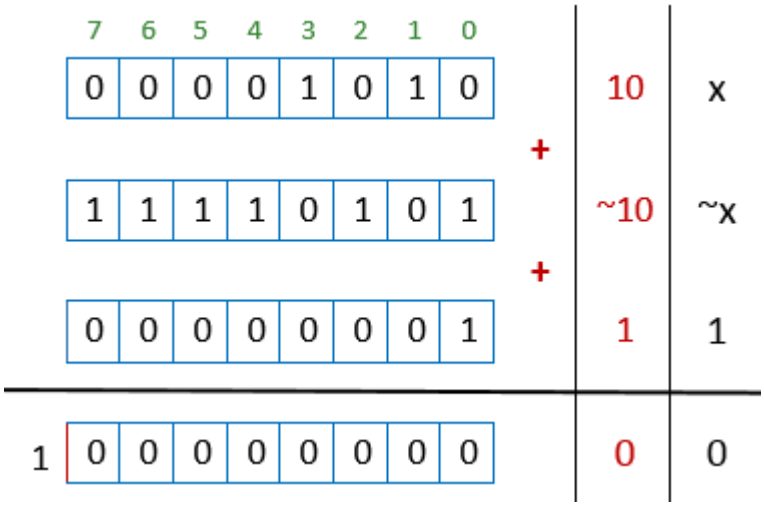
В этом примере в случае, если переменная x имеет тип unsigned char, то значением переменной y будет 245, а если тип char, то значением y будет -11.

Проверим, что в случае знакового типа (char) результат действительно будет равен -11. Для этого рассмотрим значение суммы 11 и -11 по модулю 2^8=256:



Из рисунка следует, что результат будет равен нулю.

Как представляются *отрицательные числа* в памяти компьютера? Рассмотрим другой пример: сложим двоичные представления чисел 10, ~10 и 1:



Нетрудно видеть, что результат равен нулю. Дело в том что, в действительности, отрицательные числа в памяти компьютера представляются по модулю соответствующей степени двойки (для типа char – по модулю 2^8).

Из рисунка следует, что число -10 можно представить как ~10+1. В общем случае, для любого целого числа x справедлива формула:

$$-x=1+\sim x$$

Операции битового сдвига

Сдвиг влево:

переменная << количество разрядов

Пример 4.

Переменная	Значение переменной	Двоичное представление
x	10	0 0 0 0 1 0 1 0
y=x<<1	20	0 | 0 0 0 1 0 1 0 0
y=x<<2	40	0 0 | 0 0 1 0 1 0 0 0
y=x<<3	80	0 0 0 | 0 1 0 1 0 0 0 0
y=x<<5	64	0 0 0 0 1 | 0 1 0 0 0 0 0 0

```
int x=10, y;
y = x<<1;
cout << y << endl;     //20
```

Нетрудно видеть, что сдвиг влево на один разряд умножает число на 2.

Сдвиг вправо:

переменная >> количество разрядов

Пример 5.

В данном примере переменная x имеет тип unsigned char.

Переменная	Значение переменной	Двоичное представление
x	10	0 0 0 0 1 0 1 0
y=x>>1	5	0 0 0 0 0 1 0 1 | 0
y=x>>2	2	0 0 0 0 0 0 1 0 | 1 0
y=x>>3	1	0 0 0 0 0 0 0 1 | 0 1 0
y=x>>5	0	0 0 0 0 0 0 0 0 | 0 1 0 1 0

```
int x=10, y;
y = x>>1;
cout << y << endl;     //5
```

Сдвиг вправо на один разряд делит число на 2 (неполное частное).

Замечание. Если тип переменной будет char, то при сдвиге вправо пустующие позиции заполняются копией знакового бита, т. е. если число отрицательно, то будут единицы.