

Урок 85. Побитовые логические операции

Побитовыми называются операции, выполняющие манипуляции с отдельными битами.

Побитовые операции выполняются только с операндами целого типа (кроме `BigInteger`). На этом уроке рассмотрим побитовые логические операции: `and`, `or`, `xor`, `not`.

Операции `and`, `or` и `xor` – бинарные (с двумя операндами), операция `not` – унарная (с одним операндом).

Побитовые операции осуществляются следующим образом: с каждым битом (0 принимается за `False`, 1 – за `True`) производится соответствующая логическая операция (См. Урок 9 «Логические операции»).

1. AND - побитовое логическое И (конъюнкция)

Если оба бита равны 1, то результат равен 1, иначе результат равен 0.

and	0	1
0	0	0
1	0	1

$$\begin{array}{r} 01111110 \\ \text{and } 00001011 \\ \hline 00001010 \end{array}$$

Операция `and` часто используется для сброса (обнуления) отдельных битов числа. При этом второй операнд называют *маской*. Обнуляются те биты операнда, которые в маске равны 0, значения остальных битов сохраняются.

* Ещё одно использование этой операции — вычисление остатка от деления на степень 2 (то есть на 2, 4, 8, 16, 32, 64, 128 и т.д.). Например, так можно вычислить остаток от деления `x` на 8:

```
ostatok8 := x and 7    // второй операнд = (степень2 - 1)
```

2. OR - побитовое логическое ИЛИ (дизъюнкция)

Если хотя бы один из битов равен 1, то результат равен 1, иначе результат равен 0.

or	0	1
0	0	1
1	1	1

$$\begin{array}{r} 01111110 \\ \text{or } 00001011 \\ \hline 01111111 \end{array}$$

Операция `or` часто используется для установки в 1 отдельных битов числа. Если бит маски равен 1, то бит результата будет равен 1, остальные биты сохраняют свои значения.

3. XOR - побитовое логическое ИСКЛЮЧАЮЩЕЕ ИЛИ (сумма по модулю 2)

Если биты имеют одинаковое значение, то результат равен 0, иначе результат равен 1.

xor	0	1
0	0	1
1	1	0

$$\begin{array}{r} 01111110 \\ \text{xor } 00001011 \\ \hline 01110101 \end{array}$$

«Исключающим или» эта операция называется потому, что результат равен 1, если один бит равен 1 или другой равен 1, а случай, когда оба равны 1, исключается. Ещё эта операция напоминает сложение, но в пределах одного бита, без переноса. $1+1=10_2$, но перенос в другой разряд игнорируется и получается 0, отсюда название «сумма по модулю 2».

Операцию `xor` можно использовать для инверсии отдельных битов числа. Инвертируются те биты, которые в маске равны 1, остальные сохраняют своё значение.

4. NOT - Побитовое логическое НЕ (отрицание, инверсия)

Каждый бит операнда меняет своё значение на противоположное ($0 \rightarrow 1$, $1 \rightarrow 0$).

not	-
0	1
1	0

not 01111110
 10000001

Операция `not` используется для инверсии всех битов числа. Аналогична действию операции `xor` с маской, состоящей из всех единиц (по числу битов в числе).

5. Пример программы («Управление битами»)

```
// Задание: в некотором байте сбросить биты 0 и 1,
// установить биты 6 и 7, остальные биты инвертировать.
program bits_control;

/// Выводит значение в двоичном виде
procedure print_bin(x: byte);
begin
    for var i := 7 downto 0 do
        write((x shr i) and 1);
        writeln;
    end;

begin
    var a: byte := $55;    // Некоторый байт    (пусть a := 01010101)
    print_bin(a);
    a := a and $FC;        // Сбросить биты 0 и 1    ($FC = 11111100)
    print_bin(a);
    a := a or $C0;         // Установить биты 6 и 7    ($C0 = 11000000)
    print_bin(a);
    a := a xor $3C;        // Инвертировать биты 2-5 ($3C = 00111100)
    print_bin(a);         // Что напечатает программа?
end.
```

6. Ещё пример

```
begin
    var b: byte := $55;
    print_bin(b);
    // Сбросить биты 0 и 1
    b := b and not ((1 shl 0) or (1 shl 1));
    // Установить биты 6 и 7
    b := b or ((1 shl 6) or (1 shl 7));
    // Инвертировать биты 2, 3, 4 и 5
    b := b xor ((1 shl 2) or (1 shl 3) or (1 shl 4) or (1 shl 5));
    print_bin(b);
end.
```

7. Задание для самостоятельной работы

Объявить байт `$7F`. Включить бит 7, выключить бит 0, затем инвертировать все биты.