**JavaScript Bitwise Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shifts left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |
| >>> | Zero fill right shift | Shifts right by pushing zeros in from the left, and let the rightmost bits fall off |

**Examples**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Result** | **Same as** | **Result** |
| 5 & 1 | 1 | 0101 & 0001 | 0001 |
| 5 | 1 | 5 | 0101 | 0001 | 0101 |
| ~ 5 | 10 | ~0101 | 1010 |
| 5 << 1 | 10 | 0101 << 1 | 1010 |
| 5 ^ 1 | 4 | 0101 ^ 0001 | 0100 |
| 5 >> 1 | 2 | 0101 >> 1 | 0010 |
| 5 >>> 1 | 2 | 0101 >>> 1 | 0010 |

**JavaScript Uses 32 bits Bitwise Operands**

JavaScript stores numbers as 64 bits floating point numbers, but all bitwise operations are performed on 32 bits binary numbers.

Before a bitwise operation is performed, JavaScript converts numbers to 32 bits signed integers.

After the bitwise operation is performed, the result is converted back to 64 bits JavaScript numbers.

The examples above uses 4 bits unsigned binary numbers. Because of this ~ 5 returns 10.

Since JavaScript uses 32 bits signed integers, it will not return 10. It will return -6.

00000000000000000000000000000101 (5)

11111111111111111111111111111010 (~5 = -6)

A signed integer uses the leftmost bit as the minus sign.

**Bitwise AND**

When a bitwise AND is performed on a pair of bits, it returns 1 if both bits are 1.

One bit example:

|  |  |
| --- | --- |
| **Operation** | **Result** |
| 0 & 0 | 0 |
| 0 & 1 | 0 |
| 1 & 0 | 0 |
| 1 & 1 | 1 |

4 bits example:

|  |  |
| --- | --- |
| **Operation** | **Result** |
| 1111 & 0000 | 0000 |
| 1111 & 0001 | 0001 |
| 1111 & 0010 | 0010 |
| 1111 & 0100 | 0100 |

**Bitwise OR**

When a bitwise OR is performed on a pair of bits, it returns 1 if one of the bits are 1:

One bit example:

|  |  |
| --- | --- |
| **Operation** | **Result** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

4 bits example:

|  |  |
| --- | --- |
| **Operation** | **Result** |
| 1111 | 0000 | 1111 |
| 1111 | 0001 | 1111 |
| 1111 | 0010 | 1111 |
| 1111 | 0100 | 1111 |

**Bitwise XOR**

When a bitwise XOR is performed on a pair of bits, it returns 1 if the bits are different:

One bit example:

|  |  |
| --- | --- |
| **Operation** | **Result** |
| 0 ^ 0 | 0 |
| 0 ^ 1 | 1 |
| 1 ^ 0 | 1 |
| 1 ^ 1 | 0 |

4 bits example:

|  |  |
| --- | --- |
| **Operation** | **Result** |
| 1111 ^ 0000 | 1111 |
| 1111 ^ 0001 | 1110 |
| 1111 ^ 0010 | 1101 |
| 1111 ^ 0100 | 1011 |

**JavaScript Bitwise AND (&)**

Bitwise AND returns 1 only if both bits are 1:

|  |  |
| --- | --- |
| **Decimal** | **Binary** |
| 5 | 00000000000000000000000000000101 |
| 1 | 00000000000000000000000000000001 |
| 5 & 1 | 00000000000000000000000000000001 (1) |

**Example**

var x = 5 & 1;

**JavaScript Bitwise OR (|)**

Bitwise or returns 1 if one of the bits are 1:

|  |  |
| --- | --- |
| **Decimal** | **Binary** |
| 5 | 00000000000000000000000000000101 |
| 1 | 00000000000000000000000000000001 |
| 5 | 1 | 00000000000000000000000000000101 (5) |

**Example**

var x = 5 | 1;

**JavaScript Bitwise XOR (^)**

Bitwise XOR returns 1 if the bits are different:

|  |  |
| --- | --- |
| **Decimal** | **Binary** |
| 5 | 00000000000000000000000000000101 |
| 1 | 00000000000000000000000000000001 |
| 5 ^ 1 | 00000000000000000000000000000100 (4) |

**Example**

var x = 5 ^ 1;

**JavaScript Bitwise NOT (~)**

|  |  |
| --- | --- |
| **Decimal** | **Binary** |
| 5 | 00000000000000000000000000000101 |
| ~5 | 11111111111111111111111111111010 (-6) |

**Example**

var x = ~5;

**JavaScript (Zero Fill) Bitwise Left Shift (<<)**

This is a zero fill left shift. One or more zero bits are pushed in from the right, and the leftmost bits fall off:

|  |  |
| --- | --- |
| **Decimal** | **Binary** |
| 5 | 00000000000000000000000000000101 |
| 5 << 1 | 00000000000000000000000000001010 (10) |

**Example**

var x = 5 << 1;

**JavaScript (Sign Preserving) Bitwise Right Shift (>>)**

This is a sign preserving right shift. Copies of the leftmost bit are pushed in from the left, and the rightmost bits fall off:

|  |  |
| --- | --- |
| **Decimal** | **Binary** |
| -5 | 11111111111111111111111111111011 |
| -5 >> 1 | 11111111111111111111111111111101 (-3) |

**Example**

var x = -5 >> 1;

**JavaScript (Zero Fill) Right Shift (>>>)**

This is a zero fill right shift. One or more zero bits are pushed in from the left, and the rightmost bits fall off:

|  |  |
| --- | --- |
| **Decimal** | **Binary** |
| 5 | 00000000000000000000000000000101 |
| 5 >>> 1 | 00000000000000000000000000000010 (2) |

**Example**

var x = 5 >>> 1;

**Binary Numbers**

Binary numbers with only one bit set is easy to understand:

|  |  |
| --- | --- |
| **Binary Representation** | **Decimal value** |
| 00000000000000000000000000000001 | 1 |
| 00000000000000000000000000000010 | 2 |
| 00000000000000000000000000000100 | 4 |
| 00000000000000000000000000001000 | 8 |
| 00000000000000000000000000010000 | 16 |
| 00000000000000000000000000100000 | 32 |
| 00000000000000000000000001000000 | 64 |

Setting a few more bits reveals the binary pattern:

|  |  |
| --- | --- |
| **Binary Representation** | **Decimal value** |
| 00000000000000000000000000000101 | 5 (4 + 1) |
| 00000000000000000000000000001101 | 13 (8 + 4 + 1) |
| 00000000000000000000000000101101 | 45 (32 + 8 + 4 + 1) |

JavaScript binary numbers are stored in two's complement format.

This means that a negative number is the bitwise NOT of the number plus 1:

|  |  |
| --- | --- |
| **Binary Representation** | **Decimal value** |
| 00000000000000000000000000000101 | 5 |
| 11111111111111111111111111111011 | -5 |
| 00000000000000000000000000000110 | 6 |
| 11111111111111111111111111111010 | -6 |
| 00000000000000000000000000101000 | 40 |
| 11111111111111111111111111011000 | -40 |

**Converting Decimal to Binary**

**Example**

function dec2bin(dec){  
    return (dec >>> 0).toString(2);  
}