# Operações de Bitwise no JavaScript

[❮ Anterior](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/js_type_conversion.asp&usg=ALkJrhjCCrCXWL-i-KsoENwDbjPajQ-Ing)[Próximo ❯](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/js_regexp.asp&usg=ALkJrhhUmvXKu92_bnuhRFq3hMcpiZuSew)

## Operadores Bitwise do Bit

|  |  |  |
| --- | --- | --- |
| **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 |

## Exemplos

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

## O JavaScript usa os operandos Bitwise de 32 bits

O JavaScript armazena números como números de ponto flutuante de 64 bits, mas todas as operações bit a bit são executadas em números binários de 32 bits.

Antes de executar uma operação bit a bit, o JavaScript converte números em números inteiros assinados de 32 bits.

Após a operação bit a bit é executada, o resultado é convertido de volta para números de JavaScript de 64 bits.

Os exemplos acima usam números binários não assinados de 4 bits. Por causa disso, ~ 5 retorna 10.

Como o JavaScript usa 32 bits inteiros assinados, ele não retornará 10. Ele retornará -6.

00000000000000000000000000000101 (5)

11111111111111111111111111111010 (~ 5 = -6)

Um inteiro assinado usa o bit mais à esquerda como o sinal de menos.

## Bitwise AND

Quando um AND bit a bit é executado em um par de bits, ele retorna 1 se ambos os bits forem 1.

Exemplo de um bit:

|  |  |
| --- | --- |
| **Operação** | **Resultado** |
| 0 e 0 | 0 |
| 0 e 1 | 0 |
| 1 e 0 | 0 |
| 1 e 1 | 1 |

Exemplo de 4 bits:

|  |  |
| --- | --- |
| **Operação** | **Resultado** |
| 1111 e 0000 | 0000 |
| 1111 e 0001 | 0001 |
| 1111 e 0010 | 0010 |
| 1111 e 0100 | 0100 |

## Bitwise OR

Quando um OR bit a bit é executado em um par de bits, ele retorna 1 se um dos bits for 1:

Exemplo de um bit:

|  |  |
| --- | --- |
| **Operação** | **Resultado** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Exemplo de 4 bits:

|  |  |
| --- | --- |
| **Operação** | **Resultado** |
| 1111 | 0000 | 1111 |
| 1111 | 0001 | 1111 |
| 1111 | 0010 | 1111 |
| 1111 | 0100 | 1111 |

## Bitwise XOR

Quando um XOR bit a bit é executado em um par de bits, ele retorna 1 se os bits forem diferentes:

Exemplo de um bit:

|  |  |
| --- | --- |
| **Operação** | **Resultado** |
| 0 ^ 0 | 0 |
| 0 ^ 1 | 1 |
| 1 ^ 0 | 1 |
| 1 ^ 1 | 0 |

Exemplo de 4 bits:

|  |  |
| --- | --- |
| **Operação** | **Resultado** |
| 1111 ^ 0000 | 1111 |
| 1111 ^ 0001 | 1110 |
| 1111 ^ 0010 | 1101 |
| 1111 ^ 0100 | 1011 |

## JavaScript Bitwise AND (&)

Bitwise AND retorna 1 somente se ambos os bits forem 1:

|  |  |
| --- | --- |
| **Decimal** | **Binário** |
| 5 | 00000000000000000000000000000101 |
| 1 | 00000000000000000000000000000001 |
| 5 e 1 | 00000000000000000000000000000001 (1) |

### Exemplo

var x = 5 & 1;

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_bitwise_and&usg=ALkJrhhh03JqTUdRuO_hOOdaz5soiKtP9g)

## JavaScript Bitwise OR (|)

Bitwise ou retorna 1 se um dos bits for 1:

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

### Exemplo

var x = 5 | 1;

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_bitwise_or&usg=ALkJrhjG6QpqKNiZmuGO7VnbTzAfknxU5w)

## JavaScript Bitwise XOR (^)

Bitwise XOR retorna 1 se os bits forem diferentes:

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

### Exemplo

var x = 5 ^ 1;

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_bitwise_xor&usg=ALkJrhjqCrX4K4oOQIB5CUIcvDkxWWcUhA)

## JavaScript Bitwise NOT (~)

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

### Exemplo

var x = ~5;

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_bitwise_not&usg=ALkJrhiD_6NksfQgVLmXzSM_Ow8G4k3JYg)

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

Este é um turno de preenchimento zero preenchido. Um ou mais bits de zero são empurrados a partir da direita e os bits mais à esquerda ficam:

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

### Exemplo

var x = 5 << 1;

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_bitwise_left&usg=ALkJrhh_yY0LB4f-9yPDdZKVSow7FI_vdg)

## JavaScript (Sign Preserving) Shift à direita do bit (>>)

Este é um sinal que preserva o turno certo. As cópias do bit mais à esquerda são inseridas a partir da esquerda e os bits mais à direita ficam:

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

### Exemplo

var x = -5 >> 1;

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_bitwise_right_sign&usg=ALkJrhhTfXXH9QEB_fj9vdRtEMna_TSF0A)

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

Este é um turno de preenchimento zero. Um ou mais bits de zero são empurrados a partir da esquerda e os bits mais à direita caem:

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

### Exemplo

var x = 5 >>> 1;

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_bitwise_right&usg=ALkJrhib2wBhWfIu00ygHbxpNvPCbXcSHQ)

## Números binários

Os números binários com apenas um conjunto de bits são fáceis de entender:

|  |  |
| --- | --- |
| **Representação Binária** | **Valor decimal** |
| 00000000000000000000000000000001 | 1 |
| 00000000000000000000000000000010 | 2 |
| 00000000000000000000000000000100 | 4 |
| 00000000000000000000000000001000 | 8 |
| 00000000000000000000000000010000 | 16 |
| 00000000000000000000000000100000 | 32 |
| 00000000000000000000000001000000 | 64 |

Definir alguns bits mais revela o padrão binário:

|  |  |
| --- | --- |
| **Representação Binária** | **Valor decimal** |
| 00000000000000000000000000000101 | 5 (4 + 1) |
| 00000000000000000000000000001101 | 13 (8 + 4 + 1) |
| 000000000000000000000000101101 | 45 (32 + 8 + 4 + 1) |

Os números binários de JavaScript são armazenados no formato de complemento de dois.

Isso significa que um número negativo é o NOT bitwise do número mais 1:

|  |  |
| --- | --- |
| **Representação Binária** | **Valor decimal** |
| 00000000000000000000000000000101 | 5 |
| 11111111111111111111111111111011 | -5 |
| 00000000000000000000000000000110 | 6 |
| 11111111111111111111111111111010 | -6 |
| 00000000000000000000000000101000 | 40 |
| 11111111111111111111111111011000 | -40 |

## Conversão decimal em binário

### Exemplo

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

[Tente você mesmo "](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&ie=UTF8&prev=_t&rurl=translate.google.com.br&sl=en&sp=nmt4&tl=pt-BR&u=https://www.w3schools.com/js/tryit.asp%3Ffilename%3Dtryjs_bitwise_convert&usg=ALkJrhiyPAEBJw2CujBrP3hMtslwLNQ0HA)

## Convertendo binário para decimal

### Exemplo

function bin2dec(bin){  
    return parseInt(bin, 2).toString(10);  
}