Operatory bitowe (tabela 2.5) pozwalają na wykonywanie operacji na poszczególnych bitach liczb. Są to: iloczyn bitowy (koniunkcja bitowa, operacja AND), suma bitowa (alternatywa bitowa, operacja OR), negacja bitowa (uzupełnienie do jedynki, operacja NOT) oraz suma bitowa modulo 2 (alternatywa bitowa wykluczająca, różnica symetryczna, operacja XOR) oraz operacje przesunięć bitów. Wszystkie operatory są dwuargumentowe, oprócz operatora bitowej negacji, który jest jednoargumentowy.

Tabela 2.5. Operatory bitowe

Operator	Wykonywane działanie	Przykład
&	iloczyn bitowy AND	a & b
|	suma bitowa OR	a | b
~	negacja bitowa NOT	~a
^	bitowa różnica symetryczna XOR	a ^ b
>>	przesunięcie bitowe w prawo	a >> n
<<	przesunięcie bitowe w lewo	

a << n

>>>

przesunięcie bitowe w prawo z wypełnieniem zerami

a >>> n

Iloczyn bitowy

Iloczyn bitowy to operacja, w wyniku której włączone pozostają tylko te bity, które były włączone w obu argumentach. Wynik operacji AND na pojedynczych bitach został zobrazowany w tabeli 2.6.

Tabela 2.6. Działanie iloczynu bitowego

Argument 1

Argument 2

Wynik

1

1

1

1

0

0

0

1

0

0

0

0

Zatem w wyniku wykonania operacji:

var liczba = 179 & 38;

zmiennej liczba zostanie przypisana wartość 34.

Dlaczego 34? Najłatwiej pokazać to, jeśli obie wartości (czyli 179 i 38) przedstawi się w postaci dwójkowej. 179 w postaci dwójkowej to 10110011, natomiast 38 to 00100110. Operacja AND będzie zatem miała postać:

10110011 (179)

00100110 (38)

00100010 (34)

Wynikiem jest więc 34. **Suma bitowa**

Suma bitowa to operacja, w której pozostają włączone te bity, które były włączone w przynajmniej jednym z argumentów. Wynik operacji OR na pojedynczych bitach został zobrazowany w tabeli 2.7.

Tabela 2.7. Działanie sumy bitowej

Argument 1

Argument 2

Wynik

1

1

1

1

0

1

0

1

1

0

0

0

Zatem w wyniku wykonania operacji:

var liczba = 34 | 65;

zmiennej liczba zostanie przypisana wartość 99.

Jeśli obie liczby zostaną rozpisane do postaci dwójkowej, da to wynik 00100010 (34) i 01000001 (65). Zatem całe działanie będzie miało postać:

00100010 (34)

01000001 (65)

01100011 (99) **Negacja bitowa**

Negacja bitowa powoduje zmianę stanu bitów. Czyli tam, gdzie dany bit miał wartość 0, będzie miał wartość 1, natomiast tam, gdzie dany bit miał wartość 1, będzie miał wartość 0. Działanie operacji NOT na pojedynczych bitach zostało zobrazowane w tabeli 2.8.

Tabela 2.8. Działanie negacji bitowej

Argument

Wynik

1

0

0

1

Bitowa różnica symetryczna

Bitowa różnica symetryczna, czyli operacja XOR, powoduje, że włączone zostają te bity, które miały różne stany w obu argumentach. Pozostałe bity zostają wyłączone. Wynik operacji XOR na pojedynczych bitach został zobrazowany w tabeli 2.9.

Tabela 2.9. Działanie bitowej różnicy symetrycznej

Argument 1

Argument 2

Wynik

1

1

0

1

0

1

0

1

1

0

0

0

Wykonanie przykładowej operacji:

var liczba = 34 ^ 118;

spowoduje przypisanie zmiennej liczba wartości 84. Jeśli obie wartości zostaną zapisane w postaci dwójkowej, to 34 przyjmie postać 00100010, natomiast 118 — postać 01110110. Operacja XOR będzie zatem wyglądała następująco:

00100010 (34)

01110110 (118)

01010100 (84) **Przesunięcie bitowe w lewo**

Przesunięcie bitowe w lewo to operacja polegająca na przesunięciu wszystkich bitów argumentu znajdującego się z lewej strony operatora w lewo, o liczbę miejsc wskazaną przez argument znajdujący się z prawej strony operatora. Tym samym wykonanie przykładowej operacji:

var liczba = 84 << 1;

spowoduje przypisanie zmiennej liczba wartości 168, gdyż działanie 84 << 1 oznacza: przesuń wszystkie bity wartości 84 o jedno miejsce w lewo. Skoro 84 w postaci dwójkowej ma postać 01010100, to po przesunięciu powstanie 10101000, czyli 168.

Warto zauważyć, że przesunięcie bitowe w lewo odpowiada mnożeniu wartości przez wielokrotność liczby 2. Czyli przesunięcie w lewo o jedno miejsce to pomnożenie przez 2, o dwa miejsca — pomnożenie przez 4, o trzy

miejsca — pomnożenie przez 8 itd. **Przesunięcie bitowe w prawo**

Analogicznie do przesunięcia w lewo, przesunięcie bitowe w prawo polega na przesunięciu wszystkich bitów argumentu znajdującego się z lewej strony operatora w prawo, o liczbę miejsc wskazaną przez argument znajdujący się z prawej strony operatora. A zatem wykonanie operacji:

```
var liczba = 84 >> 1;
```

spowoduje przypisanie zmiennej liczba wartości 42, oznacza to bowiem przesunięcie wszystkich bitów wartości 01010100 o jedno miejsce w prawo, czyli powstanie wartości 00101010 dwójkowo (42 dziesiętnie).

Tu również należy zwrócić uwagę, że przesunięcie bitowe w prawo odpowiada podzieleniu wartości przez wielokrotność liczby 2. Czyli przesunięcie w prawo o jedno miejsce to podzielenie przez 2, o dwa miejsca — podzielenie przez 4, o trzy miejsca — podzielenie przez 8 itd. Należy jednak pamiętać, że jeżeli dzielona liczba będzie nieparzysta, to w wyniku takiego dzielenia zostanie utracona część ułamkowa wyniku.