Bitowe operacje logiczne

Język C był tworzony po to, żeby zastąpić assembler. Z tego wynika konieczność wsparcia większości operacji, które wykonuje assembler. Operacje bitowe – to sprawdzenie, ustalenie albo przesunięcie bitów w bajcie lub słowie, które odpowiadają podstawowym typom języka C, mianowicie *char*, *int*.

Operatory bitowe:

~	**uzupełnienie**	**(NOT)**
|	**alternatywa**	**(OR)**
&	**koniunkcja**	**(AND)**
^	**różnica symetryczna**	**(XOR)**
<<	**przesunięcie w lewo**	
>>	**przesunięcie w prawo**	

Przykład 1 (bitwise_operations)

```
char a = 4;   //0100
char b = 7;   //0111
char c;
c = a&b;       //0100          4
c = a|b;        //0111          7
c = a^b;        //0011          3

c ^= c; //bitowe wyzerowuje zmiennej c = c^c;
c |= 1;  // ustawia 0 bit w c          0001          1                  c = c|1

c ^= c;                                                                c = c^c
c |= 2; // ustawia 1 bit w c          0010          2                   c = c|2

c ^= c;
c |= 3; // ustawia 0 i 1 bity w c     0011          3
```

Przykład 2

int i = 0xAB00;	1010 1011 0000 0000	
int j = 0xABCD, n;	1010 1011 1100 1101	
n = i & j;	1010 1011 0000 0000	0xAB00
n = i | j;	1010 1011 1100 1101	0xABCD
n = i ^ j;	0000 0000 1100 1101	0x00CD

Przykład 3

```
char x;
```

x = 7;	0000 0111	7
x = x << 1;	0000 1110	14
x = x << 3;	0111 0000	112
x = x << 2;	1100 0000	192
x = x >> 1;	0110 0000	96
x = x >> 2;	0001 1000	24

Każde przesunięcie w lewo powoduje mnożenie przez 2, a w prawo – dzielenie przez 2.

Po przesunięciu x << 2 lewe bity są zgubione, ponieważ pozostałe przesunięte poza granicę bajta.

Przesunięcie w prawo nie przewróciło zgubionych bitów.

Przykład 4

```
unsigned int x, y, z;
x = 0x00AA;                 //0000 0000 1010 1010
y = 0x5500;                 //0101 0101 0000 0000

z = (x << 8) + (y >> 8); //1010 1010 0000 0000     x << 8    0xAA00
                         //0000 0000 0101 0101     y >> 8    0x0055
                         //……..    +
                         //1010 1010 0101 0101               0xAA55
// z = 0xAA55
```