Программирование на языке Java

Тема 46. Битовые операции

Битовые операции

Битовые операции

Битовые операции — некоторые операции над цепочками битов.

В программировании, как правило, рассматриваются лишь некоторые виды этих операций:

- логические побитовые операции;
- битовые сдвиги.

В **Java** битовые операции могут применяться к целочисленным типам: **`byte, char, short, int, long.`**

Побитовые операции в Java – 1

Операция	Описание
~	Побитовая унарная операция NOT (Не, инверсия)
&	Побитовое AND (И)
|	Побитовое OR (ИЛИ)
^	Побитовое XOR (исключающее ИЛИ)
>>	Сдвиг вправо
>>>	Сдвиг вправо с заполнением нулями (без учета знака)
<<	Сдвиг влево
&=	Побитовое AND с присваиванием
|=	Побитовое OR с присваиванием

Побитовые операции в Java – 2

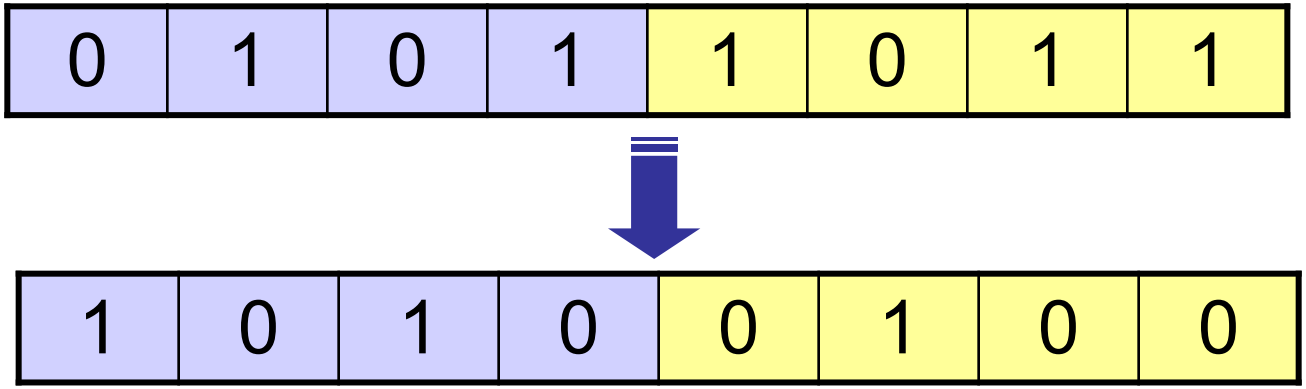
Операция	Описание
^=	Побитовое исключающее OR с присваиванием
>>=	Сдвиг вправо с присваиванием
>>>=	Сдвиг вправо с заполнением нулями (без учета знака) с присваиванием
<<=	Сдвиг влево с присваиванием

Побитовые логические операции

A	B	A | B	A & B	A ^ B	~A
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

Инверсия (операция НЕ)

Инверсия – это замена всех «0» на «1» и наоборот.




```
int n = 91;
n = ~n;
print(n);
```

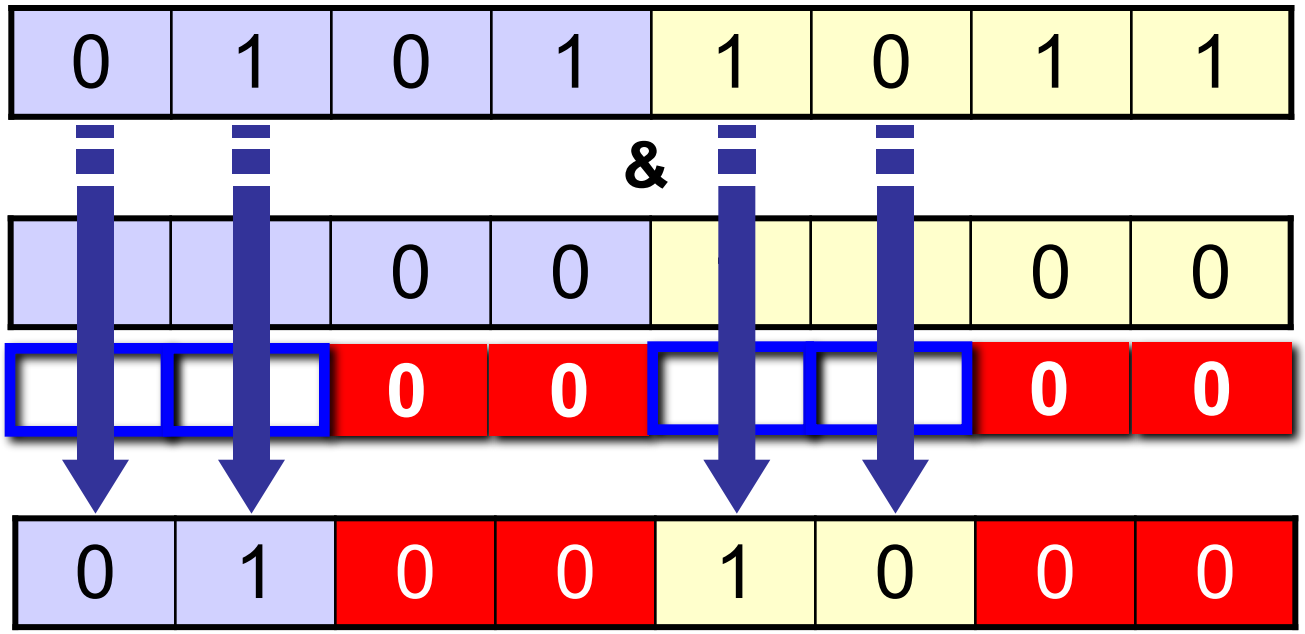
-92

Операция И

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

Обозначения:

И, **∧**, **`&` (Java)**



x & 0 = **0**
x & 1 = **x**

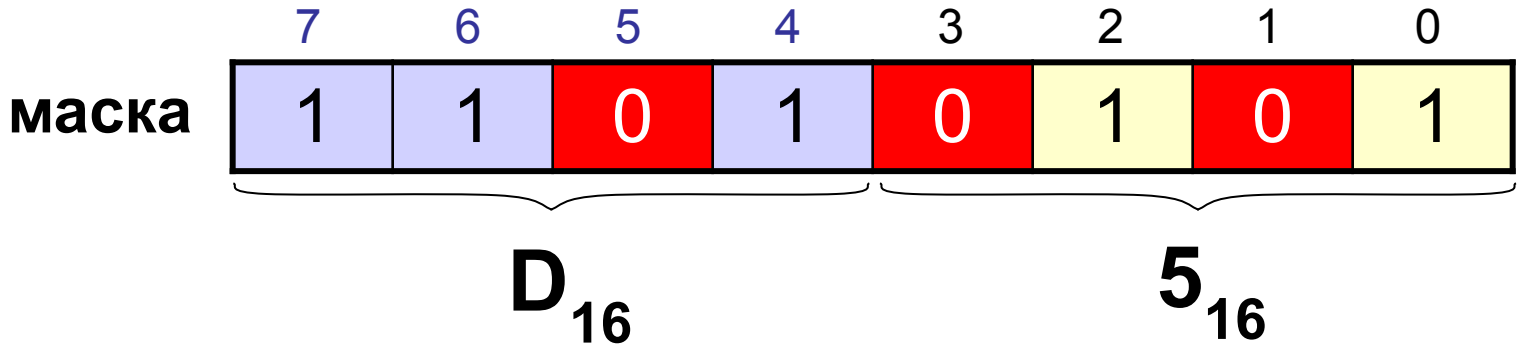
маска

$5B_{16}$ & CC_{16} = 48_{16}

Операция И – обнуление битов

Маска: обнуляются все биты, которые в маске равны «0».

Задача: обнулить 1, 3 и 5 биты числа, оставив остальные без изменения.

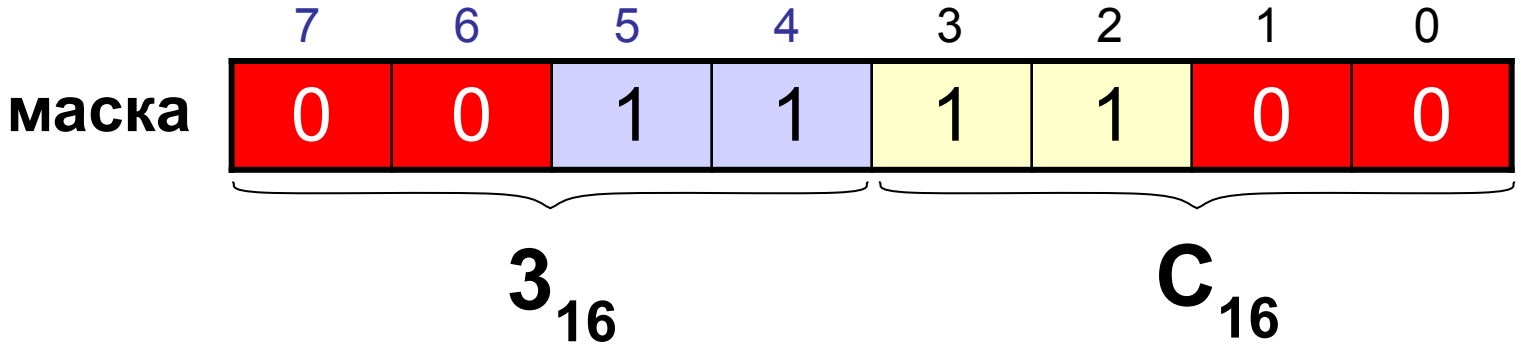



```
int n;
n = n & 0xD5;
```

Запись шестнадцатеричного числа

Операция И – проверка битов

Задача: проверить, верно ли, что все биты 2…5 – нулевые.



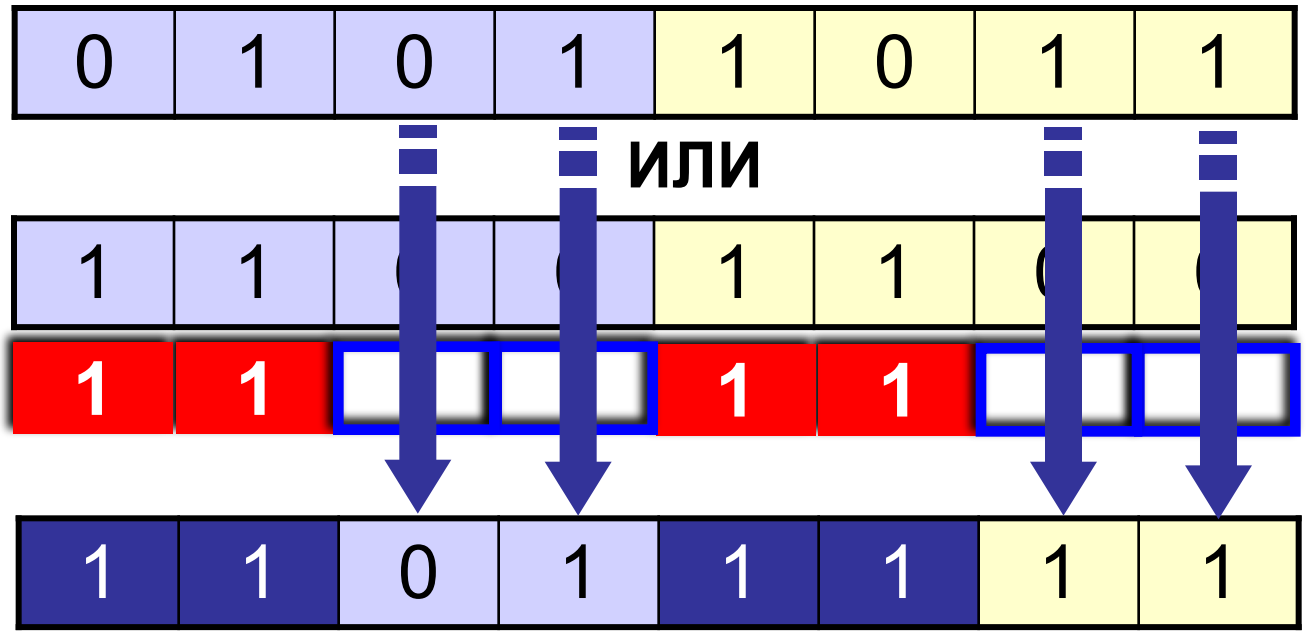

```
if ( n & 0x3C == 0 )
    print (“Биты 2-5 нулевые.”);
else
    print(“В битах 2-5 есть ненулевые.”);
```

Операция ИЛИ

A	B	A или B
0	0	0
0	1	1
1	0	1
1	1	1

Обозначения:

ИЛИ, **V**, **|** **(Java)**



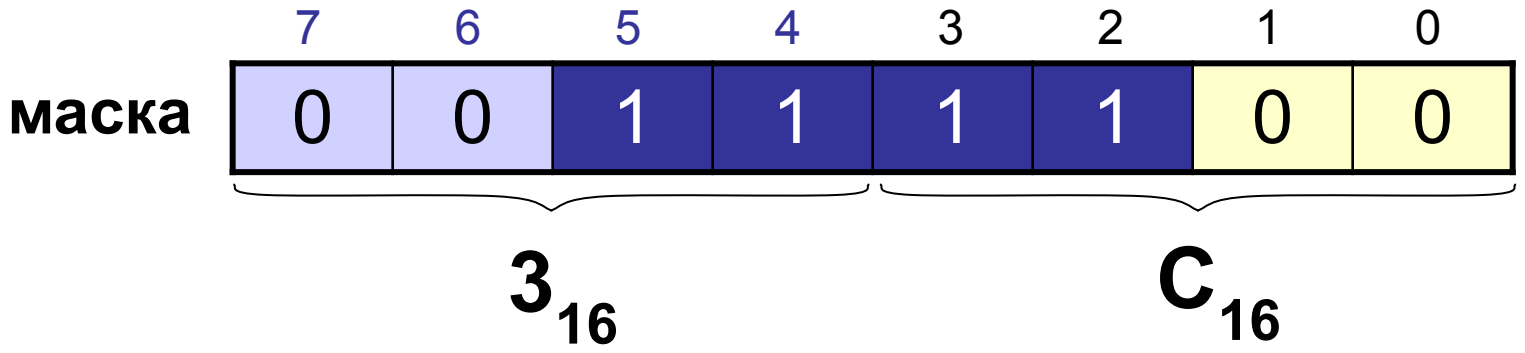
x **ИЛИ** 0 =**x**
x **ИЛИ** 1 =**1**

маска

$5B_{16}$ | CC_{16} = DF_{16}

Операция ИЛИ – установка битов в 1

Задача: установить все биты 2…5 равными 1, не меняя остальные.




```
n = n | 0x3C;
```

Операция «исключающее ИЛИ»

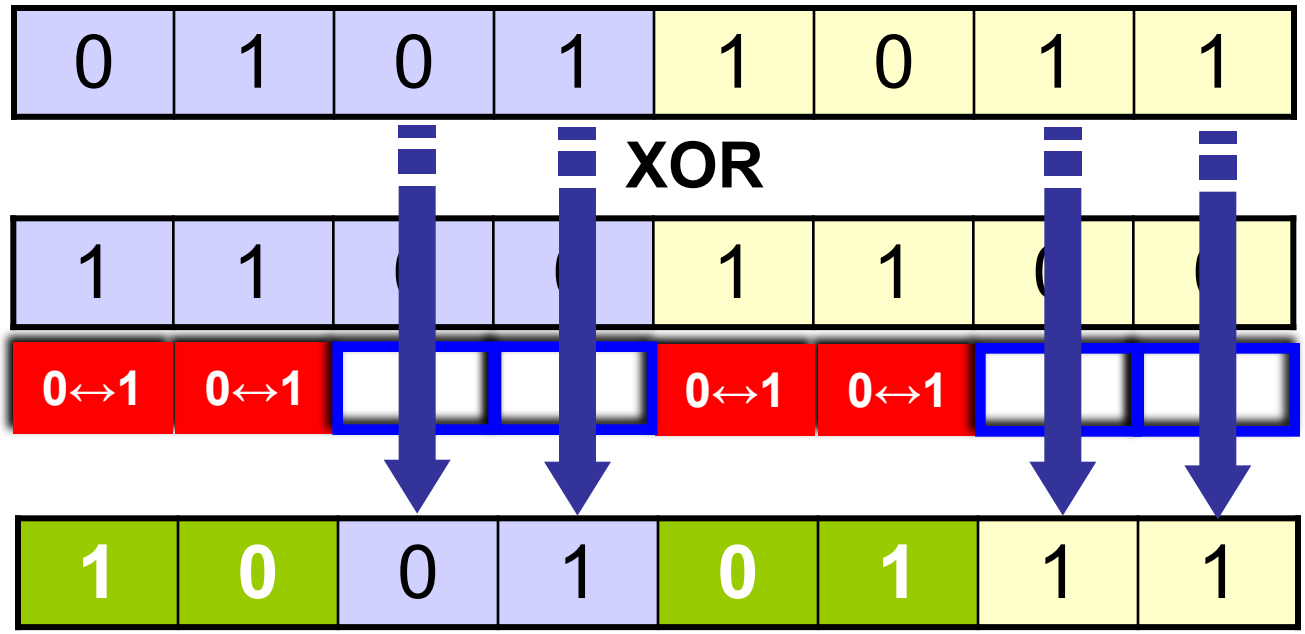
A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Обозначения:

⊕, XOR, ^ (Java)

x XOR 0 = x

x XOR 1 НЕ x

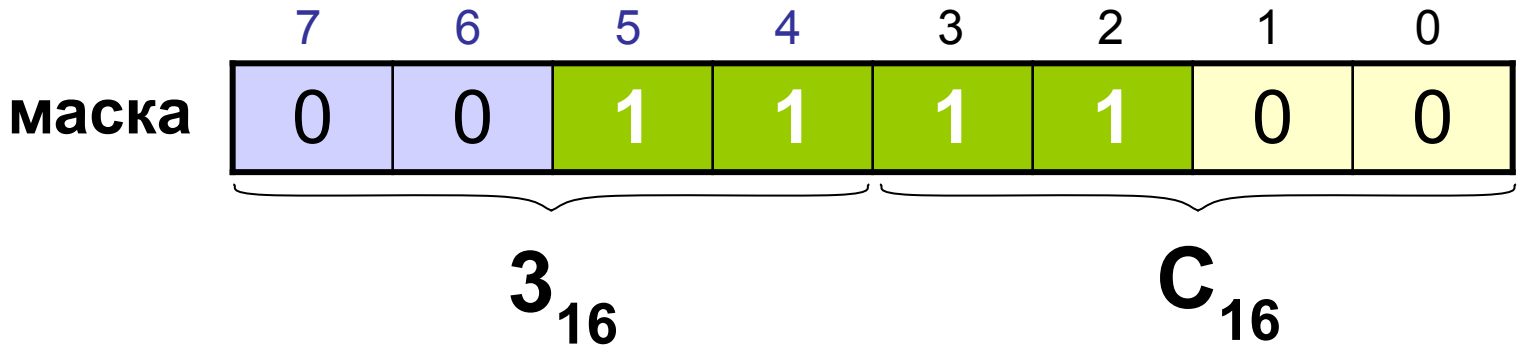


маска

$5B_{16}$ ^ CC_{16} = 97_{16}

«Исключающее ИЛИ» – инверсия битов

Задача: выполнить инверсию для битов 2…5, не меняя остальные.




```
n = n ^ 0x3C;
```

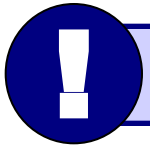
«Исключающее ИЛИ» – шифровка

```
(0 xor 0) xor 0 = 0
(1 xor 0) xor 0 = 1
```

```
(0 xor 1) xor 1 = 0
(1 xor 1) xor 1 = 1
```

код (шифр)

```
(X xor Y) xor Y = X
```



«Исключающее ИЛИ» – обратимая операция

Шифровка:

выполнить для каждого байта текста операцию **XOR** с байтом-шифром.

Расшифровка: сделать то же самое с **тем же** шифром.

Побитовые логические операции. Пример

```
int a = 3;
int b = 6;

int c = a | b;

int d = a & b;

int e = a ^ b;

int f = (~a & b) | (a & ~b);

int g = ~a & 0x0f;
```

0011

0110

7 (0111_2)

2 (0010_2)

5 (0101_2)

5 (0101_2)

12 (1100_2)

15 (1111_2)

Битовые сдвиги

При сдвиге значения битов копируются в соседние биты по направлению сдвига.

В зависимости от обработки крайних битов различают следующие сдвиги:

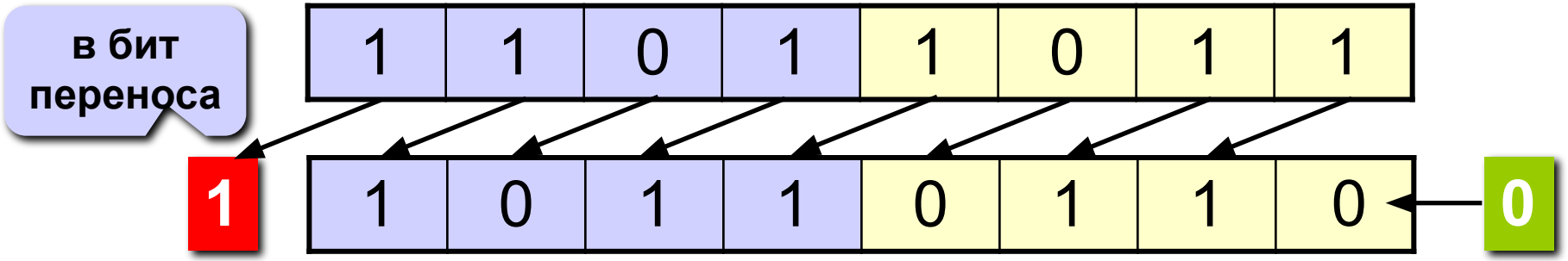
- логический;
- циклический;
- арифметический.

Логический сдвиг

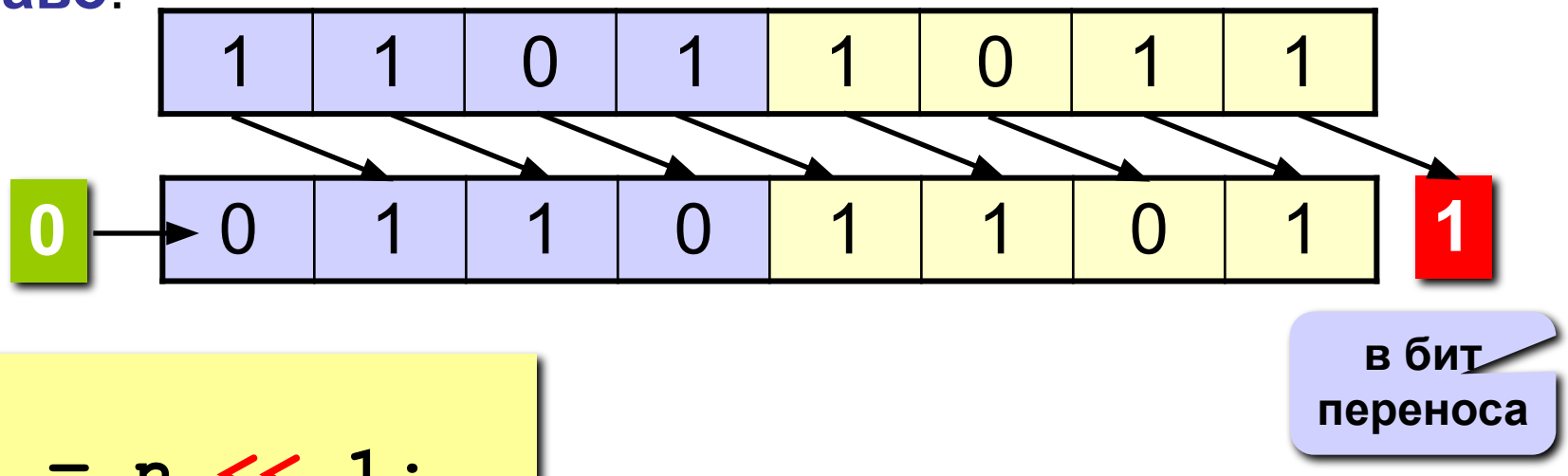
При логическом сдвиге значение последнего бита по направлению сдвига теряется (копируясь в бит переноса), а первый приобретает нулевое значение.

Логический сдвиг

Влево:



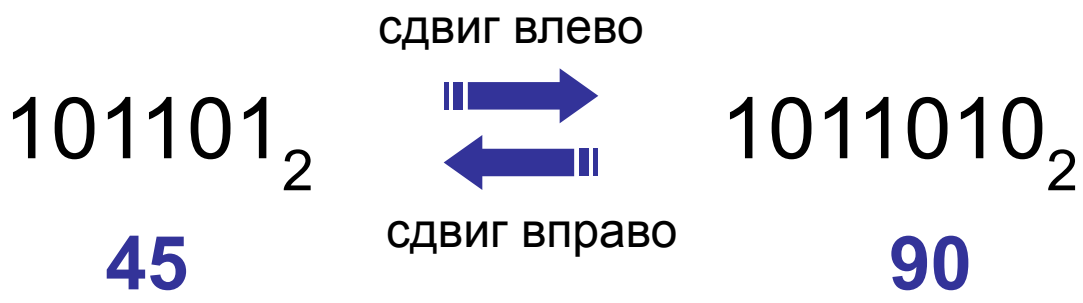
Вправо:




```
n = n << 1;
n = n >>> 1;
```

Логический сдвиг

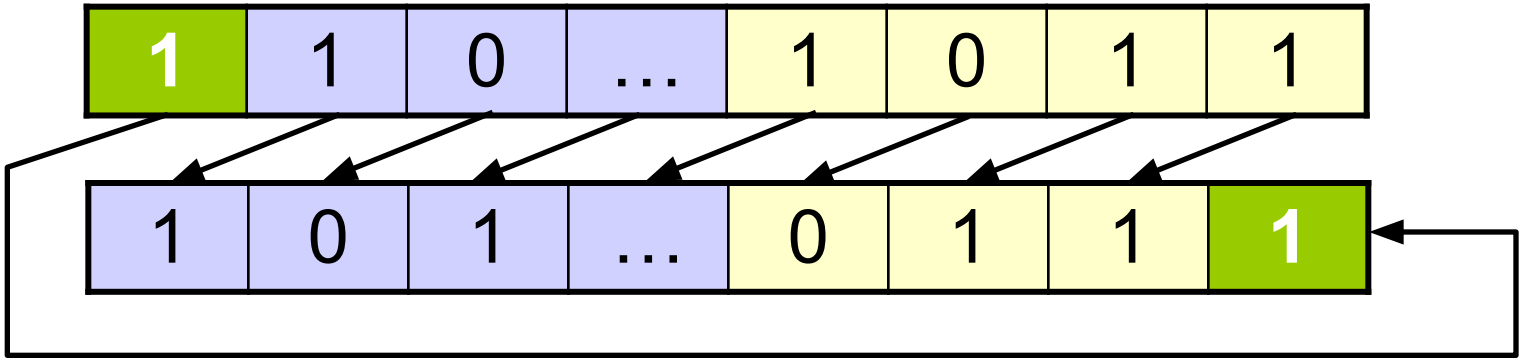
Какой арифметической операции равносилен логический сдвиг влево (вправо)? При каком условии?



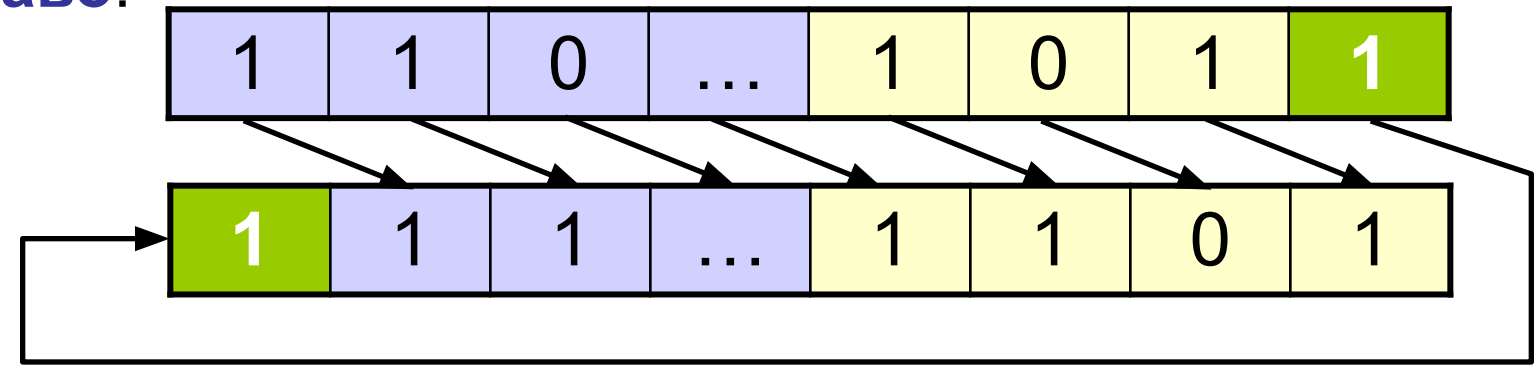
Логический сдвиг влево (вправо) – это быстрый способ **умножения** (**деления без остатка**) **на 2**.

Циклический сдвиг

Влево:



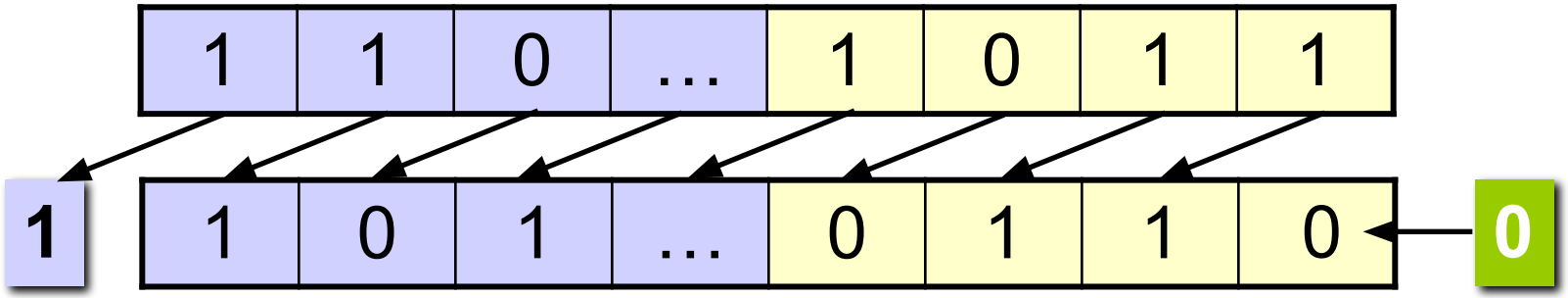
Вправо:



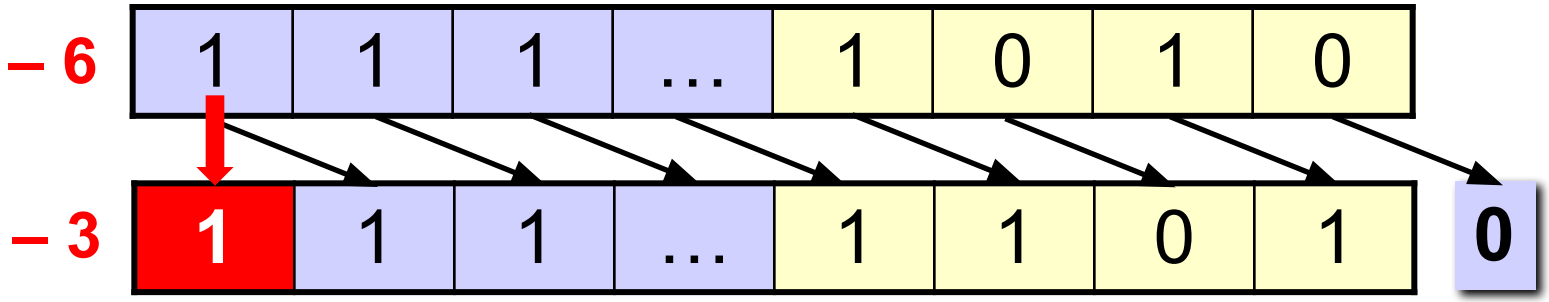
В языке **Java** циклический сдвиг не реализован

Арифметический сдвиг

Влево (= логическому):



Вправо (знаковый бит не меняется!):

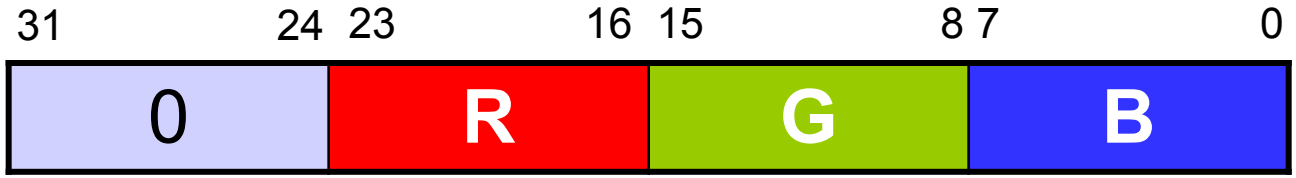



```
n = -6;
n = n >> 1;
```

-3

Задача

Задача: в целой переменной **n** (**32 бита**) закодирована информация о цвете пикселя в **RGB**:



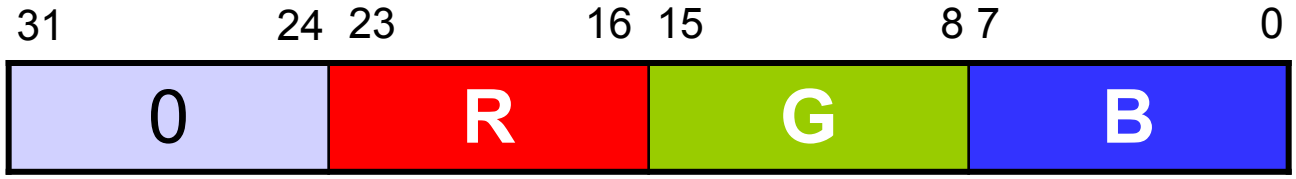
Выделить в переменные R, G, B составляющие цвета.

Вариант 1:

1. Обнулить все биты, кроме **G**.
   Маска для выделения **G**: $0000FF00_{16}$
2. Сдвинуть вправо так, чтобы число **G** передвинулось в младший байт.

```
G = (n & 0xFF00) >> 8;
```

Задача

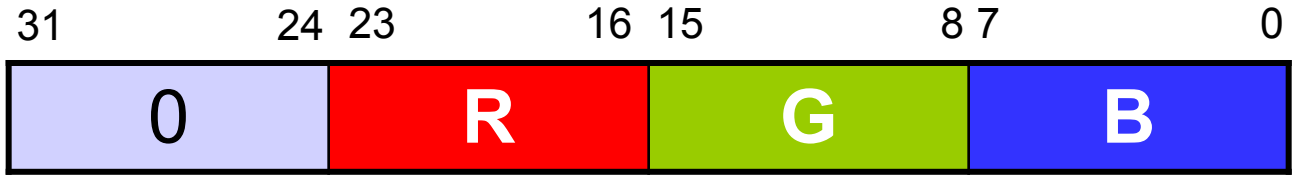


Вариант 2:

1. Сдвинуть вправо так, чтобы число **G** передвинулось в младший байт.
2. Обнулить все биты, кроме **G**.
   Маска для выделения **G**: **$000000FF_{16}$**

```
G = (n >> 8) & 0xFF;
```

Задача




```
R =

B =
```

Автоматическое повышение типов

Внимание! При работе с типами **byte, short, int** происходит автоматическое повышение типа до **long**.

```
byte a = 64, b, c;
int i;
i = a << 2;
b = (byte) (a << 2);
c = a << 2;
```

256

0

Ошибка!
Несоответствие типов

Задачи

Запишите значения x, y и что будет выведено на экран

```
double x = 2./0;
double y = -1/0.;
System.out.print(x+y);
```

Задачи

Запишите значение b и что будет выведено на экран

```
long m = -130;
byte b = (byte)-m;
System.out.print("b" + b);
```

Задачи

Запишите значения b, c и что будет выведено на экран

```
int a = -125;
int b = (a>>2);
int c = (a<<2);
System.out.println((byte)(b + c));
```

Задачи

1. Запишите x&y, используя | и ~
2. Выведите n-ый байт заданного числа x (нумерация справа налево, начиная с 0)
3. Запишите наименьшее отрицательное число в дополнительной кодировке, не использую Integer.MIN_VALUE
4. Выясните достаточно ли n бит для представления числа x
5. Вычислите $x/2^n$ и $x*2^n$, не используя операции умножения и деления
6. Дано число x, вычислите -x без обращения знака
7. Выясните является ли число x неотрицательным