Компьютерная арифметика

§ 28. Операции с целыми числами

Сложение и вычитание

Операции с положительными и отрицательными числами выполняются по одинаковым алгоритмам!

```
  5          0000 0101
+ -9       + 1111 0111
-----      -----------
 -4    ←     1111 1100
```

Вычитание = сложение с дополнительным кодом вычитаемого!

Переполнение

дополнительный бит

```
  0 | 01100000      96
+ 0 | 00100001      33
-----------------------
  0 | 1000001    -127
  S'  S
```

знаковый бит

```
  1 | 10100000     -96
+ 1 | 11011111     -33
-----------------------
  1 | 0111111     127
  S'  S
```

Если бит S не совпадает с битом S', произошло переполнение и результат неверный.

Умножение

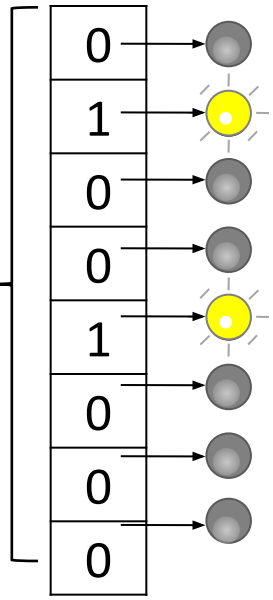
```
  × 00001001    9
    00000101    5
  ----------
    00001001
+ 00000000
  00001001
----------
  00000101101  →45
```

```
  × 11110111    -9
    00000101    5
  ----------
    11110111
+ 00000000
  11110111
----------
  10011010011  →-45
```

Умножение выполняется с помощью сложения и сдвига.

Поразрядные логические операции

Поразрядные операции выполняются с отдельными битами числа и не влияют на остальные.



? Сложение – это поразрядная операция?

Операция «НЕ» (инверсия, not):

R	1	0	0	1	1	0	1	0
not R	0	1	1	0	0	1	0	1

Логическая операция «И» (and, &)

данные маска

D	M	D and M
0	0	0
1	0	0
0	1	0
1	1	1

Маска – константа, которая определяет область применения логической операции к битам многоразрядного числа.

AA_{16} and $6C_{16}$ = ?

	7	6	5	4	3	2	1	0	
D	1	0	1	0	1	0	1	0	AA_{16}
M	0	1	1	0	1	1	0	0	$6C_{16}$
D and M	0	0	1	0	1	0	0	0	28_{16}

С помощью операции «И» можно сбросить (установить в ноль) биты, для которых маска равна 0!

Логическая операция «ИЛИ» (or, |)

D	M	D or M
0	0	0
1	0	1
0	1	1
1	1	1

AA_{16} or $6C_{16}$ = ?

	7	6	5	4	3	2	1	0	
D	1	0	1	0	1	0	1	0	AA_{16}
M	0	1	1	0	1	1	0	0	$6C_{16}$
D or M	1	1	1	0	1	1	1	0	EE_{16}

С помощью операции «ИЛИ» можно записать единицу в биты, для которых маска равна 1!

Операция «исключающее ИЛИ» (xor, ^)

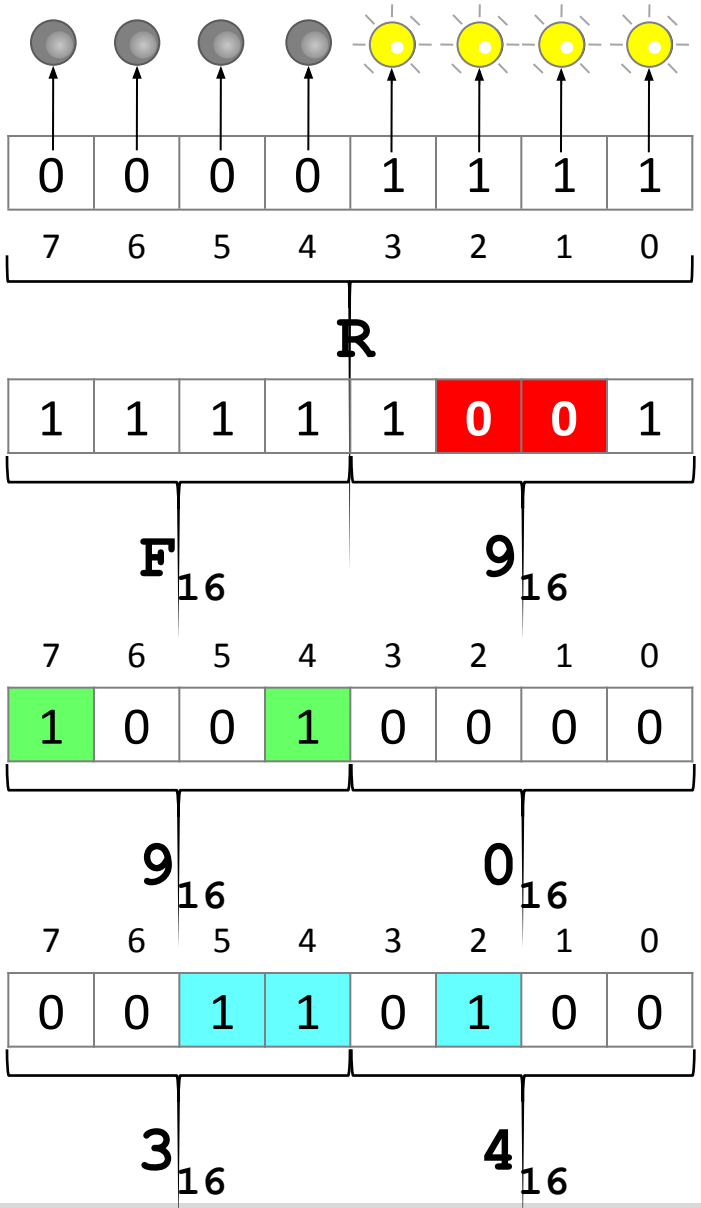
D	M	D xor M
0	0	0
1	0	1
0	1	1
1	1	0

AA_{16} xor $6C_{16}$ = ?

	7	6	5	4	3	2	1	0	
D	1	0	1	0	1	0	1	0	AA_{16}
M	0	1	1	0	1	1	0	0	$6C_{16}$
D xor M	1	1	0	0	0	1	1	0	$C6_{16}$

С помощью операции «исключающее ИЛИ» можно инвертировать биты, для которых маска равна 1!

Битовые логические операции (итог)



1) отключить лампочки 2 и 1, не трогая остальные

$$R = R \text{ and } F9_{16}$$

2) включить лампочки 7 и 4

$$R = R \text{ or } 90_{16}$$

3) изменить состояние лампочек 5, 4 и 2

$$R = R \text{ xor } 34_{16}$$

Шифрование с помощью xor

Идея: `(A xor B) xor B = A`

Операция «исключающее ИЛИ» *обратима*, то есть ее повторное применение восстанавливает исходное значение!

Текст: `2*2=4`

Коды символов:

$'2' = 32_{16} = 00110010_2$

$'*' = 2A_{16} = 00101010_2$

$'=' = 3D_{16} = 00111101_2$

$'4' = 34_{16} = 00110100_2$

Шифрование с помощью xor

Маска: $23 = 17_{16} = 00010111_2$

'2' → 32_{16} xor 17_{16} = 25_{16} → '%'
'*' → $2A_{16}$ xor 17_{16} = $3D_{16}$ → '='
'=' → $3D_{16}$ xor 17_{16} = $2A_{16}$ → '*'
'4' → 34_{16} xor 17_{16} = 23_{16} → '#'

Исходный текст: 2*2=4

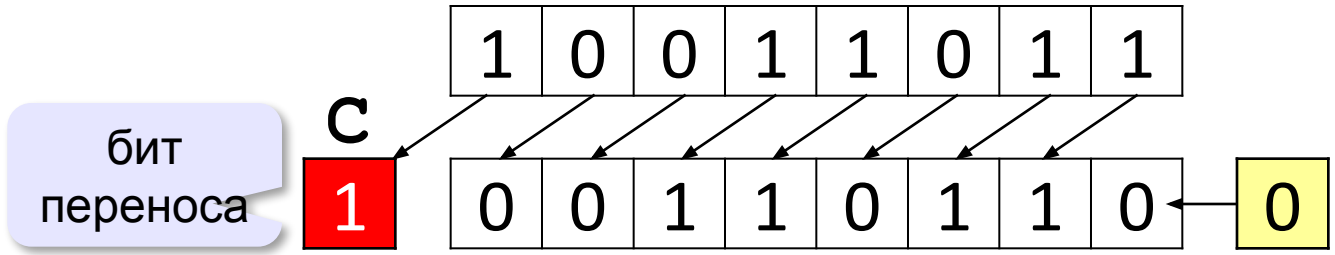
Зашифрованный текст: %=%*#

Расшифровка:

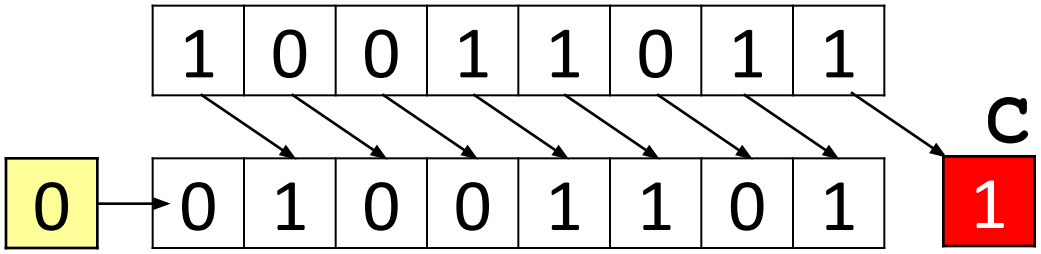
'%' → 25_{16} xor 17_{16} = 32_{16} → '2'
'=' → $3D_{16}$ xor 17_{16} = $2A_{16}$ → '*'
'*' → $2A_{16}$ xor 17_{16} = $3D_{16}$ → '='
'#' → 23_{16} xor 17_{16} = 34_{16} → '4'

Логический сдвиг

Влево:



Вправо:



C, C++, Python :

```
N = N << 1;
N = N >> 1;
```

Паскаль:

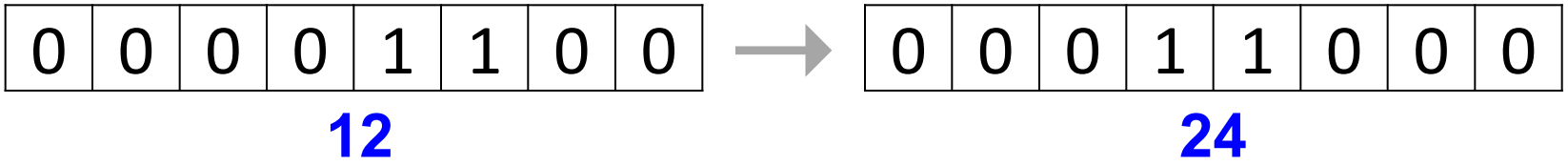
shift left

```
N := N shl 1;
N := N shr 1;
```

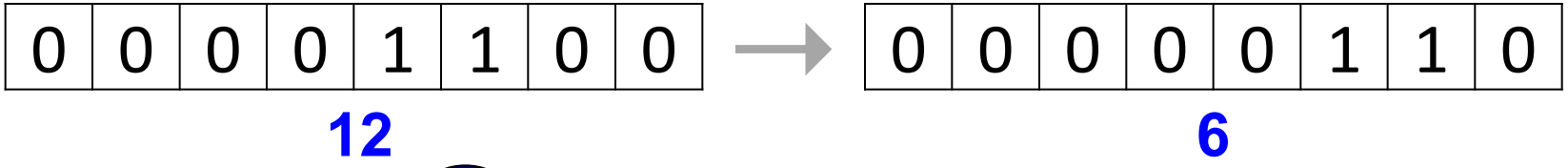
shift right

Логический сдвиг

Влево:



Вправо:

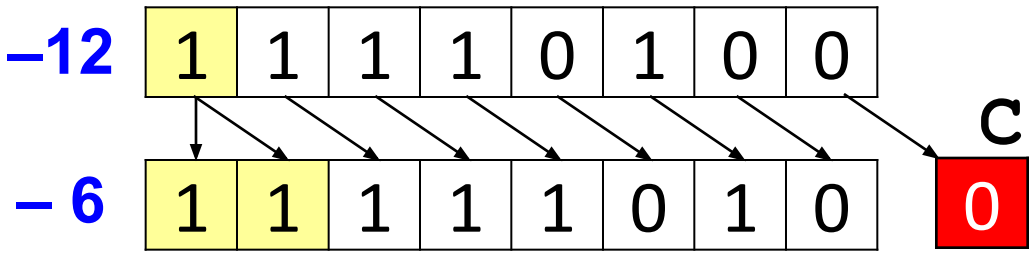


? Если число нечётное?

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

? Если число отрицательное?

Арифметический сдвиг (вправо)



Если число нечётное?

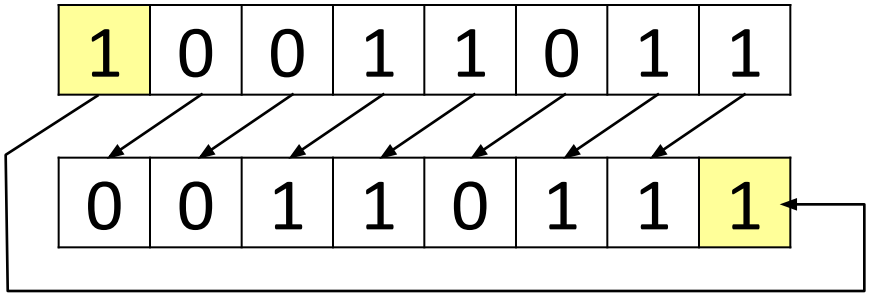
Примеры:

20 → 10	–20 → –10
15 → 7	–15 → –8
11 → 5	–11 → –6
3 → 1	–3 → –2
1 → 0	–1 → –1

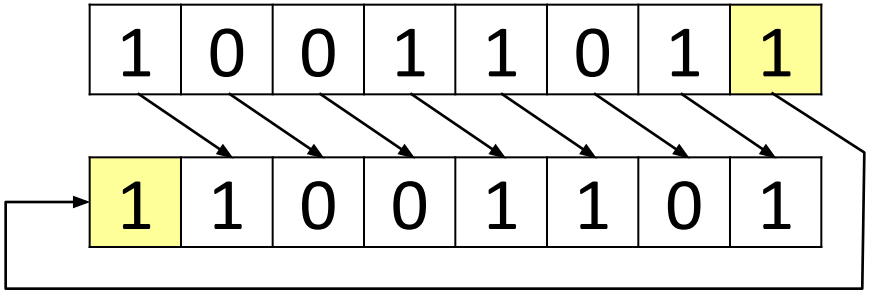
Арифметический сдвиг вправо – деление на 2 нацело **с округлением «вниз»** (к ближайшему меньшему целому).

Циклический сдвиг

Влево:



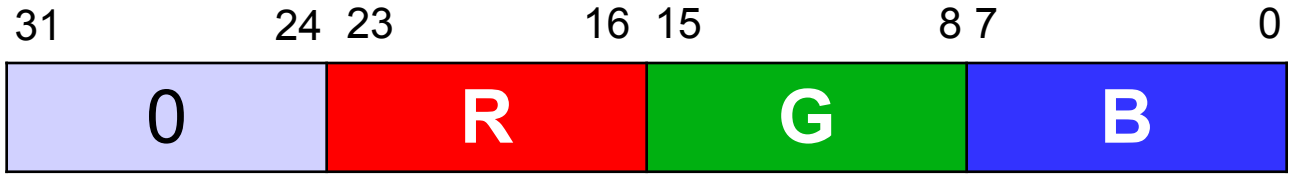
Вправо:



Циклический сдвиг предназначен для последовательного просмотра битов без потери данных.

Пример

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



Записать в переменные R, G, B составляющие цвета.

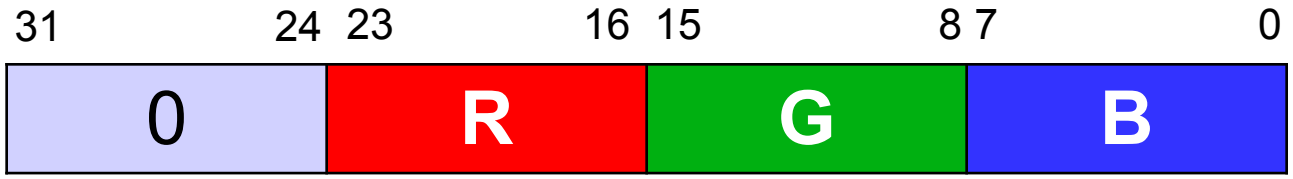
Вариант 1:

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

C, C++, Python:
```
G = (N & 0xFF00) >> 8;
```

Паскаль:
```
G:=(N and $FF00) shr 8;
```

Пример



Вариант 2:

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

C, C++, Python:

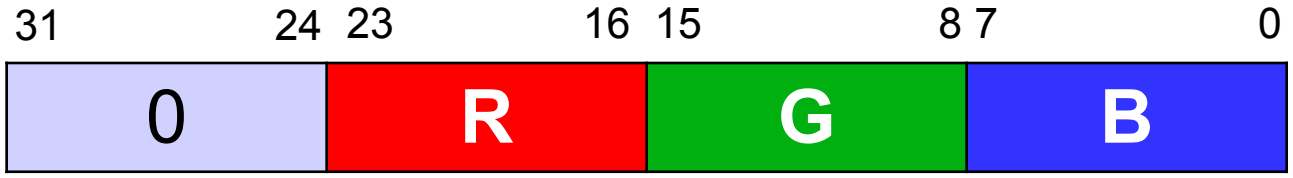
```
G=(N >> 8) & 0xFF;
```

Паскаль:

```
G:=(N shr 8) and $FF;
```

? Как решить, используя только сдвиги?

Пример



Си:

```
R =

B =
```

Паскаль:

```
R:=

B:=
```

Конец фильма

ПОЛЯКОВ Константин Юрьевич
д.т.н., учитель информатики
ГБОУ СОШ № 163, г. Санкт-Петербург
kpolyakov@mail.ru

ЕРЕМИН Евгений Александрович
к.ф.-м.н., доцент кафедры мультимедийной дидактики и ИТО ПГГПУ, г. Пермь
eremin@pspu.ac.ru

Источники иллюстраций

1. авторские материалы