IT ШКОЛА SAMSUNG

Модуль 1. Основы программирования

Урок 7-8. Поразрядные и логические операции

SAMSUNG

ПОРАЗРЯДНЫЕ ОПЕРАЦИИ

Действительно ли 1 байт = 8 бит?

Байт – минимальная единица адресации памяти компьютера.

~~Чтение и запись бит~~

С битами можно только работать используя поразрядные операции

NOT (~) – побитовое отрицание

меняет каждый бит на противоположный

```
~ 00000000 00000000 00000000 01111011 (123)
=
  11111111 11111111 11111111 10000100 (-124)
```

Применяется при выполнении вычитания
(сложение с инвертированным числом)

ПОРАЗРЯДНЫЕ ОПЕРАЦИИ

AND (&) – побитовое умножение (И)

Бит выставляется 1, если у множителей биты = 1

```
00000000 00000000 00000000 01111011 (123)
&
00000000 00000000 00000001 11001000 (456)
=
00000000 00000000 00000000 01001000 (57)
```

Применяется чтобы выяснить какое значение стоит в определенном бите:

1)Число **маска –** нужный бит устанавливается в 1, остальные в 0

2)Выполняем побитовое умножение между числом и маской

3)Результат дает информацию о нужном бите

```
  xxxxx?xx
& (поразрядное И)
  00000100
  00000?00
```

Либо 0 либо не 0

ПОРАЗРЯДНЫЕ ОПЕРАЦИИ

OR (|) – побитовое сложение (ИЛИ)

Бит выставляется 1, если хотя бы одно число = 1

```
00000000 00000000 00000000 01111011 (123)
|
00000000 00000000 00000001 11001000 (456)
=
00000000 00000000 00000001 11111011 (507)
```

Применяется для установки 1 в любом бите числа
- используется число-маска

xxxxx?xx
| (поразрядное ИЛИ)
00000100
xxxxx1xx

Всегда будет 1

Маски позволяют в определенных заранее битах сохранять несколько двоичных значений и считывать их

ПОРАЗРЯДНЫЕ ОПЕРАЦИИ

XOR (^) – исключающее ИЛИ

Бит выставляется 1, если только одно любое число = 1

```
00000000 00000000 00000000 01111011 (123)
^
00000000 00000000 00000001 11001000 (456)
=
00000000 00000000 00000001 10110011 (435)
```

Очень часто применяется в шифровании

5 ^ 7 → 2 (значение изменилось - зашифровалось)

2 ^ 7 → 5 (используя ключ можно расшифровать

Для усложнения кодирования можно использовать «одноразовый блокнот» - каждый символ шифруется своим ключом

ПОРАЗРЯДНЫЕ ОПЕРАЦИИ

SAMSUNG

>> – сдвиг вправо

Все биты смещаются вправо, бит слева устанавливается в 1, если число отрицательное, либо обнуляется, если число положительное. Один сдвиг вправо соответствует делению на 2

```
42 >> 5 = 8
```

Если делится нечетное число, то остаток отбрасывается для положительных чисел и сохраняется для отрицательных

<< – сдвиг влево

Все биты смещаются влево, оставшиеся справа обнуляются – один сдвиг влево соответствует умножению на 2

```
5 << 3 = 40
```

соответствует 5*8(2 в степени 3)

```
01111111 11111111 11111111 11111111 (2147483647)
<<
11111111 11111111 11111111 11111110 (-2 в дополнительном коде)
```

Операции сдвига выполняются быстрее операций умножения и деления

ПОРАЗРЯДНЫЕ ОПЕРАЦИИ

SAMSUNG

JAVA

```
int i = 192;  // 00000000 00000000 00000000 11000000 (192)
i<<1;         // 00000000 00000000 00000001 10000000 (384)
i>>1;         // 00000000 00000000 00000000 01100000 (96)
int j =-192;  // 11111111 11111111 11111111 01000000 (-192)
j<<1;         // 11111111 11111111 11111110 10000000 (-384)
j>>1;         // 11111111 11111111 11111111 10100000 (-96)
```

ЛОГИЧЕСКИЕ ОПЕРАЦИИ

Операции сравнения

> строго больше

< строго меньше

>= больше или равно

<= меньше или равно

== равно

!= не равно

Логические операции

&& И (**конъюнкция**)

Результат ИСТИНА, если оба операнда ИСТИНА, в остальных случаях - ЛОЖЬ

||ИЛИ (**дизъюнкция**)

Результат ЛОЖЬ, если оба операнда ЛОЖЬ, в остальных случаях - ИСТИНА

! НЕ (**отрицание**)

Унарная операция. Если операнд ИСТИНА, то результат - ЛОЖЬ, и наоборот

Приоритет (по уменьшению)

!

&&

||

В Java двойные условия типа **1 < x < 6** записываются не прямо, а через два простых и логическую связку И **1 < x && x < 6**

ТИП Boolean и ТЕРНАРНАЯ ОПЕРАЦИЯ

Тип Boolean

Используется для хранения логических величин: `true, false`

Занимает 1 байт памяти

С такими переменными можно выполнять **только логические операции!**

Тернарная операция

```
//ПСЕВДОКОД
<условие> ? <значение, если условие истинно>
:
<значение, если ложно>
```

```
max = (a > b ? a : b);
```

```
out.println(a > 0 ? a * a : "WRONG");
```

САМОСТОЯТЕЛЬНАЯ РАБОТА

1) Напишите Android-приложение, которое трижды будет применять операцию исключающего ИЛИ к двум переменным:
   `first, last` – любого типа, значения запрашиваются у пользователя
   ```
   first ^= last;
   last ^=first;
   first ^=last;
   ```
   Выведите значение этих переменных до и после указанного фрагмента. Сделайте вывод: что делает этот фрагмент

2) Напишите Android-приложение, которое осуществит сдвиг вправо для числа -1. Попробуйте объяснить результат (ответ)

3) Возьмите целую переменную i равную 1. Выведите ее значение побитово.
   Далее осуществите действия: i<<29, i<<30, i<<31, i<<32 каждый раз на экран выводите значение переменной i. Посмотрите на закономерность. Попробуйте сделать вывод (ответ)

Программа с прошлого занятия

```
Float f = new Float("124.32432");
 int intBits = Float.floatToIntBits(f);
 String binary = Integer.toBinaryString(intBits);
 out.println("Binary = " + binary);
```

Домашнее задание

Прочитать http://habrahabr.ru/post/187606/

Изучить http://dark-barker.blogspot.ru/2012/03/bit-operations-java-pitfalls.html - подводные камни при работе с битовыми сдвигами. Каждый пример запрограммировать по аналогии работы в классе

Спасибо!

В презентации использованы материалы К. Полякова
http://kpolyakov.spb.ru/

Ответы

2) Есть некоторое отличие от целочисленного деления на 2: если сдвигать отрицательное число вправо, то сначала это аналогично целочисленному делению на 2, но когда от числа останется -1, то при следующих сдвигах остаётся всё время -1.
При целочисленном делении же получится, как понятно, 0. То есть происходит округление не к нулю, как при целочисленном делении, а к -1.

назад

Ответы

3) Здесь тоже таится некоторая неожиданность. Это свойственно не только JVM, но и большинству обычных процессоров. Нельзя сдвинуть на количество бит, большее, чем разрядность операнда. При этом происходит неявное сокращение правого (кол-во бит) операнда.

назад