Представление с точкой

Основы информатики

Компьютерные основы программирования

u.to/DbCmFA

На основе **CMU** 15-213/18-243:
Introduction to Computer Systems

u.to/XoKmFA

Лекция 3, 17 февраля, 2023

Лектор:

Дмитрий Северов, кафедра информатики 608 КПМ

cs.mipt.ru/wp/?page_id=346

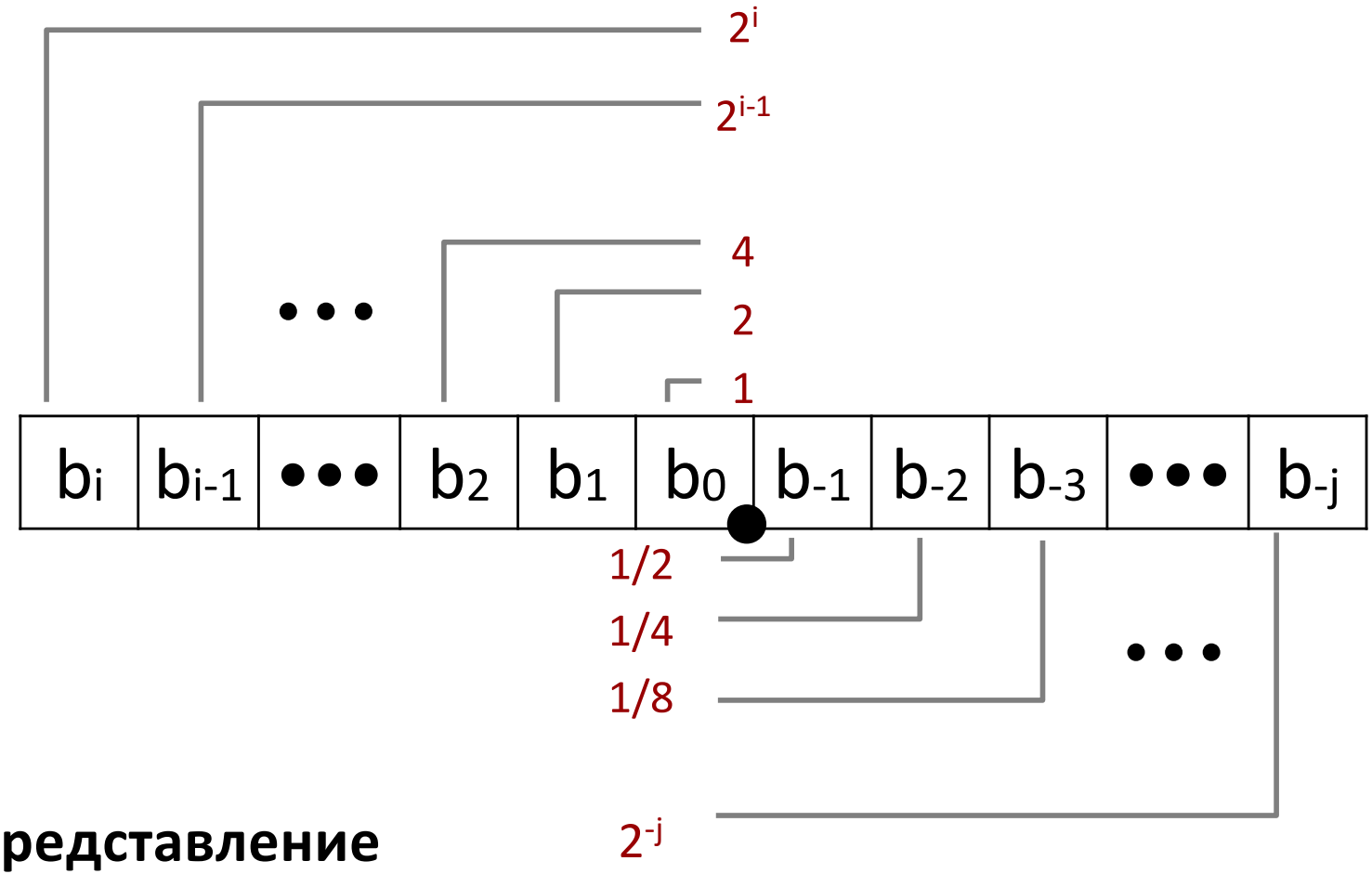
Плавающая точка

- **Основы: Двоичные дроби**
- Стандарт «плавающей точки» IEEE 754 : Определение
- Примеры и свойства
- Округление, сложение, умножение
- Плавающая точка в Си
- Сводка

Двоичные дроби

- Что такое 1011.101_2?

Двоичные дроби



- **Представление**
  - Биты справа от «двоичной точки» представляют дробные степени 2
  - Представление рациональных чисел:

$$\sum_{k=-j}^{i} b_k \times 2^k$$

Двоичные дроби: Примеры

- **Значение** **Представление**

  5 и 3/4 101.11_2

  2 и 7/8 10.111_2

  1 и 7/16 1.0111_2

  23/32 0.10111_2

- **Важно**
  - **Деление на 2 сдвигом вправо**
  - **Умножение на 2 сдвигом влево**
  - **Числа вида $0.111111\ldots_2$ меньше 1.0**
    - **$1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$**
    - **Обозначим $1.0 - \varepsilon$**

(не)Представимые числа (1)

- **Ограничения**
  - Двоичными дробями точно представимы только числа вида $n/2^k$
  - Другие дроби представимы бесконечными периодическими «дробями»

- **Значение Представление**
  - 1/3 `0.[01]…`$_2$
  - 1/5 `0.[0011]…`$_2$
  - 1/10 `0.0[0011]…`$_2$

Плавающая точка

- Основы: Двоичные дроби
- **Стандарт «плавающей точки» IEEE : Определение**
- Примеры и свойства
- Округление, сложение, умножение
- Плавающая точка в Си
- Сводка

Плавающая точка IEEE

- **Стандарт IEEE 754**
  - Принят в 1985 как единый стандарт арифметики с плавающей точкой
    - До этого множество уникальных стандартов
  - Поддерживается всеми основными CPU/FPU
- **В основе - вопросы вычислений**
  - Удачно стандартизованы
    - округления,
    - переполнения,
    - потеря значимости
  - Сложно сделать быстрым в аппаратуре
    - При создании стандарта численные аналитики доминировали над разработчиками аппаратуры
  - Есть реализации «с отклонениями»
- **Есть версии и альтернативы**

Представление с плавающей точкой

- **Численная форма:**

$$(-1)^s\ M\ \ 2^E$$

  - Знаковый бит **s** определяет положительность/отрицательность
  - Мантисса **M** - обычно дробь в интервале [1.0,2.0).
  - Порядок **E** изменяет значение на степень двойки

- **Кодирование**
  - Наиболее значимый бит **s** – знаковый
  - Поле **exp** кодирует **E**, но не совпадает с двоичным значением **E**
  - Поле **frac** кодирует **M**, но не совпадает с двоичным значением **M**

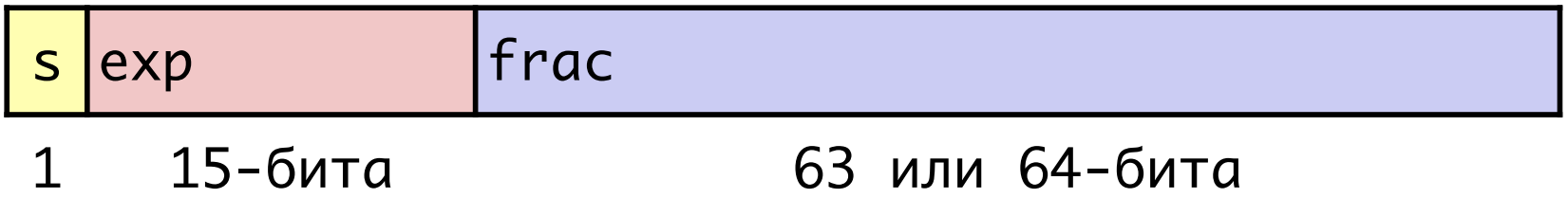
s	exp	frac

Применяемые точности

- **Одинарная: 32 бита**

- **Двойная: 64 бита**

- **Расширенная: 80 бит (только для Intel)**



Нормализованные значения

- **Признак: exp ≠ 000…0 и exp ≠ 111…1**

- **Порядок кодируется со смещением : E = Exp – Bias**
  - Exp: беззнаковое значение поля **exp**
  - Bias = 2^{k-1} - 1, где k - количество бит порядка – смещение
    - Одинарная точность: 127 (Exp: 1…254, E: -126…127)
    - Двойная точность: 1023 (Exp: 1…2046, E: -1022…1023)

- **Код мантиссы подразумевает старшую 1: M = 1.xxx…x**$_2$
  - xxx…x: биты поля **frac**
  - Минимальное значение M = 1.0 , когда **frac** = 000…0
  - Минимальное значение M = 2.0 – ε , когда **frac** = 111…1
  - Старший бит мантиссы не расходует ресурсы оборудования
  - <u>Не подразумевается для расширенной точности 80-бит Intel !</u>

Пример нормализованного кода

- **Значение:** `Float F = 15213.0;`
  - $15213_{10} = 11101101101101_2$
    $= 1.1101101101101_2 \times 2^{13}$

- **Мантисса**

  $M = 1.\underline{1101101101101}_2$

  `frac=` $\underline{1101101101101}0000000000_2$

- **Порядок**

  $E = 13$

  $Bias = 127$

  $Exp = 140 = 10001100_2$

- **Результат:**

0	10001100	11011011011010000000000
s	exp	frac

Денормализованные значения

- **Признак: exp = 000…0**

- **Значение порядка: E = 1–Bias (вместо E = 0 – Bias)**
- **Код мантиссы подразумевает старший 0: M = 0.xxx…x_2**
  - **xxx**…**x**: биты **frac**
- **Варианты**
  - **exp** = 000…0, **frac** = 000…0
    - Обозначает нулевое значение
    - Представление неоднозначно: +0 and –0 (почему?)
  - **exp** = 000…0, **frac** ≠ 000…0
    - числа очень близкие к 0.0
    - чем меньше, тем хуже относительная погрешность
    - равноотстоящие

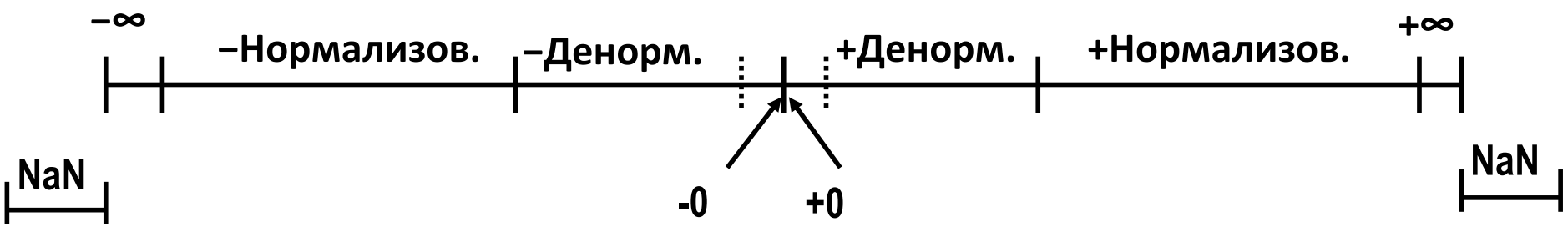
Специальные коды

- **Признак: `exp` = 111…1**

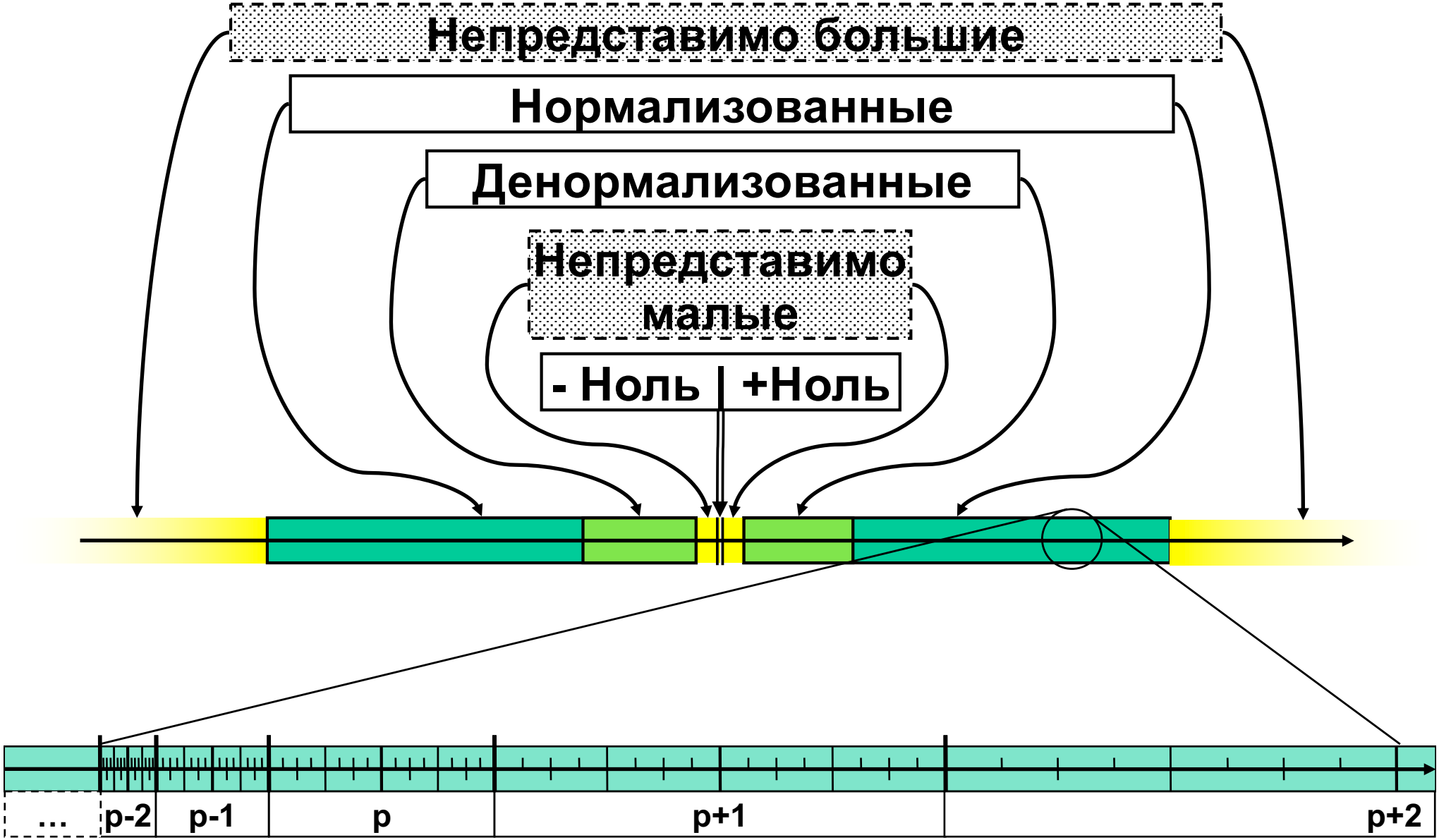
- **Вариант 1: `exp` = 111…1, `frac` = 000…0**
  - Обозначает значение ∞ (бесконечность)
  - Результат операции при переполнении
  - Есть оба варианта: отрицательная и положительная
  - E.g., 1.0/0.0 = −1.0/−0.0 = +∞, 1.0/−0.0 = −∞

- **Вариант 2: `exp` = 111…1, `frac` ≠ 000…0**
  - Не число, Not-a-Number (NaN)
  - Обозначает случаи когда невозможно определить численное значение
  - Примеры: sqrt(–1), ∞ − ∞, ∞ × 0

Коды с плавающей точкой визуально



(не)Представимые числа (2)



Плавающая точка

- Основы: Двоичные дроби
- Стандарт «плавающей точки» IEEE : Определение
- **Примеры и свойства**
- Округление, сложение, умножение
- Плавающая точка в Си
- Сводка

Укороченный пример с плавающей точкой

s	exp	frac
1	4-бита	3-бита

- **8-битное представление с плавающей точкой**
  - Знаковый бит – самый значимый
  - следующие четыре бита - показатель со смещением $2^{4-1}-1 = 7$
  - последние три бита - **код мантиссы**

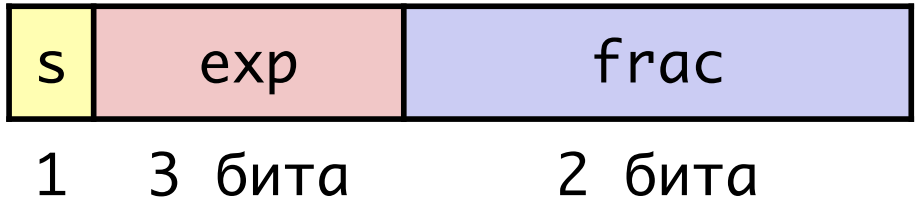
- **Форма таже что в стандарте IEEE**
  - нормализованные и денормализованные
  - обозначаются 0, NaN, бесконечность

Динамический диапазон положительных

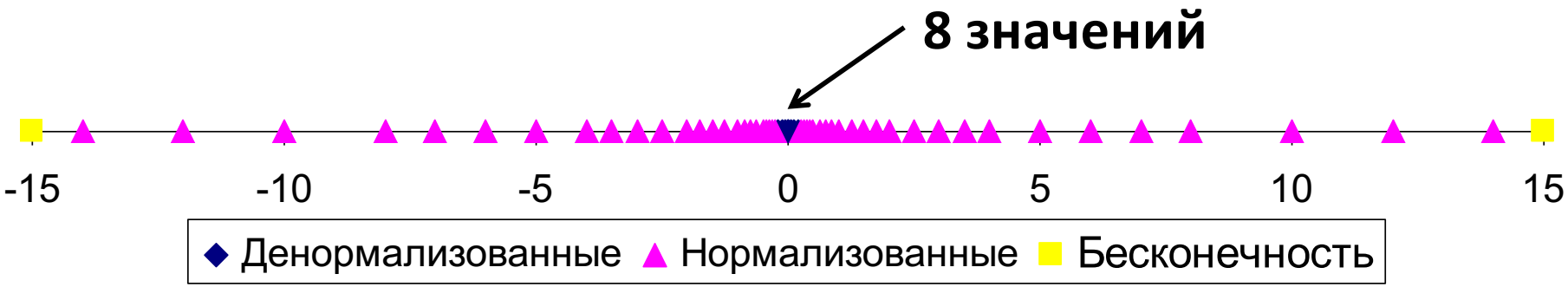
	s	exp	frac	E	Значение	
Деномализ. числа	0	0000	000	-6	0	
	0	0000	001	-6	1/8*1/64 = 1/512	Ближайшее к нулю
	0	0000	010	-6	2/8*1/64 = 2/512	
	…					
	0	0000	110	-6	6/8*1/64 = 6/512	
	0	0000	111	-6	7/8*1/64 = 7/512	Наибольшее денорм.
Нормализов. числа	0	0001	000	-6	8/8*1/64 = 8/512	Наименьшее нормализ.
	0	0001	001	-6	9/8*1/64 = 9/512	
	…					
	0	0110	110	-1	14/8*1/2 = 14/16	
	0	0110	111	-1	15/8*1/2 = 15/16	Ближайшее к 1 снизу
	0	0111	000	0	8/8*1 = 1	
	0	0111	001	0	9/8*1 = 9/8	Ближайшее к 1 сверху
	0	0111	010	0	10/8*1 = 10/8	
	…					
	0	1110	110	7	14/8*128 = 224	
	0	1110	111	7	15/8*128 = 240	Наибольшее нормализ.
	0	1111	000	n/a	inf	

Распределение значений

- **Шестибитный формат подобный IEEE**
  - 3 бита порядка
  - 2 дробных бита
  - смещение: $2^{3-1}-1 = 3$

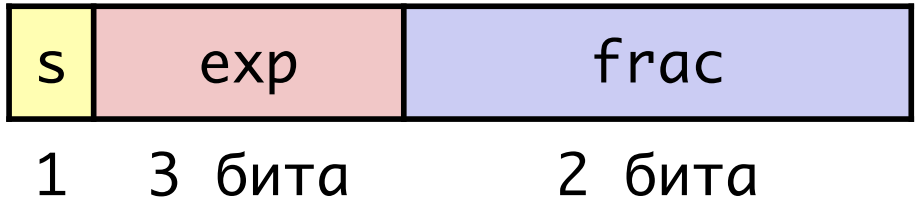


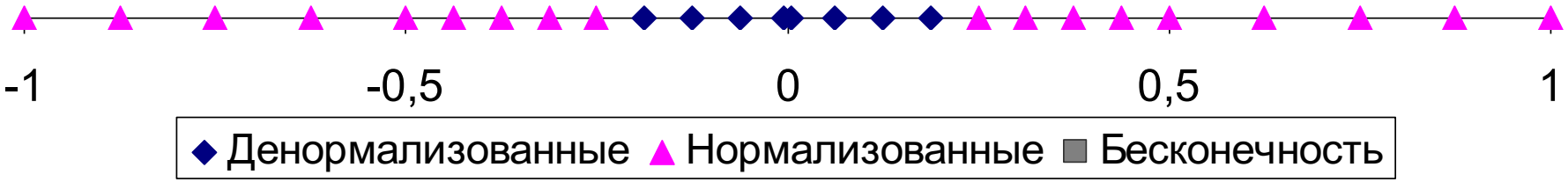
- **Сгущение вокруг 0**



Распределение значений (крупным планом)

- **Шестибитный формат подобный IEEE**
  - 3 бита порядка
  - 2 дробных бита
  - смещение: $2^{3-1}-1 = 3$





Особые свойства кодирования IEEE

- **Совпадают коды целочисленного нуля и «положительного» нуля с плавающей точкой**
  - Все биты - нулевые

- **Можно ли сравнить как беззнаковые целые?**
  - Сравнить знаковые биты
  - Учесть –0 = 0
  - Решить проблему с NaN
    - NaN код больше любого численного кода
    - Что должно выдать сравнение?
  - В остальном - OK
    - Денормализованные с нормализованными
    - Нормализованные с бесконечностью

Плавающая точка

- Основы: Двоичные дроби
- Стандарт «плавающей точки» IEEE : Определение
- Примеры и свойства
- **Округление, сложение, умножение**
- Плавающая точка в Си
- Сводка

Операции с плав. точкой: Основная идея

- $x +_f y = \mathrm{Round}(x + y)$

- $x \times_f y = \mathrm{Round}(x \times y)$

- **Основная идея**
  - Сначала вычислить полный результат
  - Затем втиснуть его в желаемую точность
    - Возможно с переполнением, если порядок слишком велик
    - Возможно с округлением, чтобы уместиться во **`frac`**

Округление

Режим округления	**1.40**	**1.60**	**1.50**	**2.50**	**–1.50**
К нулю	1	1	1	2	–1
Вниз (–∞)	1	1	1	2	–2
Вверх (+∞)	2	2	2	3	–1
К чётному (по умолч.)	1	2	2	2	–2

- **В чём преимущества режимов?**

Подробнее об округлении к чётному

- **Режим округления по умолчанию**
  - Любой другой режим требует обращения к ассемблеру
  - Все остальные режимы дают статистический сдвиг
    - Сумма округлённых устойчиво меньше или устойчиво больше округлённой суммы

- **Применяя к десятичным разрядам**
  - Если точно посередине между двумя возможными значениями
    - Округляется так, что младшая цифра чётная
  - Например, округление к ближайшей сотой

1.2349999	1.23	(Меньше половины - вниз)
1.2350001	1.24	(Больше половины - вверх)
1.2350000	1.24	(Половина - вверх)
1.2450000	1.24	(Половина - вниз)

Округление двоичных чисел

- **Двоичные дроби**
  - “Чётные” когда наименьший бит = 0
  - “Половина” когда биты справа от позиции округления = $100\ldots_2$

- **Примеры**
  - Округление к 1/4 (2 бита справа от двоичной точки)

Значение	Двоичное Округлённое	Двоичное Округление	Действие	Значение
2 3/32	10.00011_2	10.00_2	(<1/2—вниз)	2
2 3/16	10.00110_2	10.01_2	(>1/2—вверх)	2 1/4
2 7/8	10.11100_2	11.00_2	(1/2—вверх)	3
2 5/8	10.10100_2	10.10_2	(1/2—вниз)	2 1/2

Умножение с плавающей точкой

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ \ x\ \ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- **Полный результат:** $(-1)^{s}\ M\ \ 2^{E}$
  - Знак s: s1 ^ s2
  - Мантисса M: M1 x M2
  - Порядок E: E1 + E2

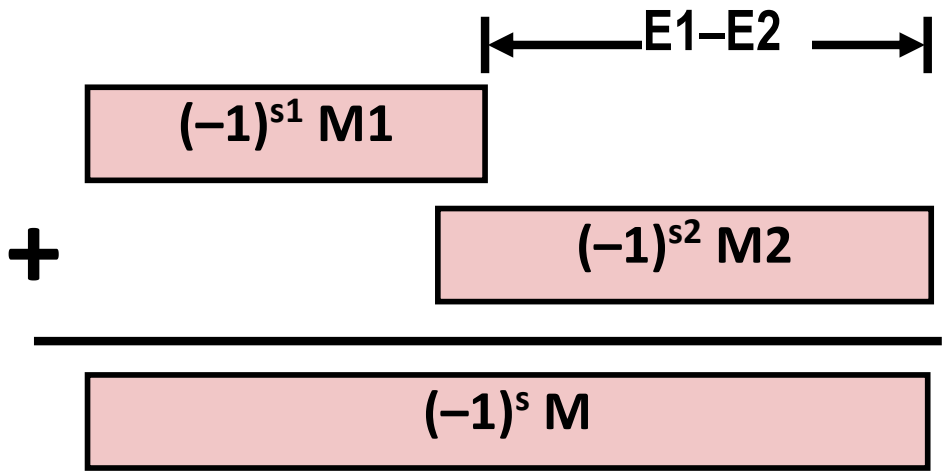
- **Нормализация**
  - Если M ≥ 2, сдвинуть M вправо, увеличив E
  - Если E за рамками, то переполнение
  - Округлить M чтобы уложиться в точность `frac`

- **Реализация**
  - Трудоёмкая часть - перемножение мантисс

Сложение с плавающей точкой

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ +\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
  - Предположим E1 > E2

- **Полный результат:** $(-1)^{s}\ M\ \ 2^{E}$
  - Знак s, мантисса M:
    - Результат знакового выравнивания и сложения
  - Порядок E: E1



- **Нормализация**
  - Если M ≥ 2, сдвинуть M вправо, увеличив E
  - Если M < 1, сдвинуть M влево на k разрядов, уменьшив E на k
  - Если E за рамками, то переполнение
  - Округлить M чтобы уложиться в точность `frac`

Математические свойства сложения с ПТ

- **Сравним со свойствами абелевой группы**
  - Замкнута относительно сложения ? **Да, но…**
    - Может выдавать бесконечность или нечисло
  - Коммутативно ? **Да**
  - Ассоциативно ? **Нет**
    - Переполнение и округление!
  - Ноль – нейтральный элемент ? **Да**
  - Каждый элемент имеет обратный ? **Почти**
    - Кроме бесконечности и нечисла
- **Монотонность**
  - $a \geq b \Rightarrow a+c \geq b+c$? **Почти**
    - Кроме бесконечности и нечисла

Математические свойства умножения с ПТ

- **Сравним со свойствами коммутативного кольца**
  - Замкнут относительно умножения ? **Да**
    - Может выдавать бесконечность или нечисло
  - Умножение коммутативно ? **Да**
  - Умножение ассоциативно ? **Нет**
    - Переполнение и округление!
  - Единица нейтральный элемент умножения ? **Да**
  - Умножение дистрибутивно относительно сложения ? **Нет**
    - Переполнение и округление!

- **Монотонность**
  - $a \geq b$ & $c \geq 0 \Rightarrow a * c \geq b * c$? **Почти**
    - Кроме бесконечностей и нечисел

Плавающая точка

- Основы: Двоичные дроби
- Стандарт «плавающей точки» IEEE : Определение
- Примеры и свойства
- Округление, сложение, умножение
- **Плавающая точка в Си**
- Сводка

Плавающая точка в Си

- **Си гарантирует как минимум два уровня точности**
  - `float` одиночная
  - `double` двойная
- **Преобразования**
  - Преобразование между `int`, `float`, и `double` меняет набор битов
  - `double`/`float` → `int`
    - Отсекает дробную часть
    - Подобно округлению к нулю
    - Не определено для запредельных или NaN: обычно TMin
  - `int` → `double`
    - Точное преобразование, т.к. `int` имеет разрядность ≤ 53 бит
  - `int` → `float`
    - Округляется в соответствии в режимом
- **x86: вычисления выражения в 80-битном представлении**

Головоломки плавающей точки

- **Для каждого следующего выражения Си объясните:**
  - либо почему оно верно для любых значений аргументов
  - либо почему неверно

```
int x = …;
float f = …;
double d = …;
```

Принять, что ни `d` ни `f` не есть NaN

- `x == (int)(float) x`
- `x == (int)(double) x`
- `f == (float)(double) f`
- `d == (float) d`
- `f == -(-f);`
- `2/3 == 2/3.0`
- `d < 0.0 ⇒ ((d*2) < 0.0)`
- `d > f ⇒ -f > -d`
- `d * d >= 0.0`
- `(d+f)-d == f`

Плавающая точка

- Основы: Двоичные дроби
- Стандарт «плавающей точки» IEEE : Определение
- Примеры и свойства
- Округление, сложение, умножение
- Плавающая точка в Си
- **Сводка**

Сводка

- **Плавающая точка IEEE 754 -1985 имеет ясные математические свойства. Почти!**
- **Представимы числа вида $(-1)^S$ M x 2^E**
- **Операции [не]зависимы от реализации**
  - Если все выполнены с двойной точностью и затем округлены
- **Не совпадают с вещественной арифметикой**
  - Нарушены свойства ассоциативности/дистрибутивности
  - Создаёт сложности для разработчиков компиляторов и ответственных численных приложений

Дополнения

- bfloat, малобитовый float для нейросетей
- IEEE 754-2019 для произвольных основания, в т.ч. 10
- IEEE std 1788.1-2017, для интервалов
- Unums: http://www.johngustafson.net/unums.html
- Логарифмические системы счисления
- …

Создание числа с плав. точкой

- **Шаги**
  - Нормализовать до получения старшей 1
  - Округлить до укладывания в мантиссу
  - Перенормализовать для учёта округления

- **Пример**
  - Преобразовать 8 битовые беззнаковые числа в формат с плавающей точкой

s	exp	frac
1	4 бита	3 бита

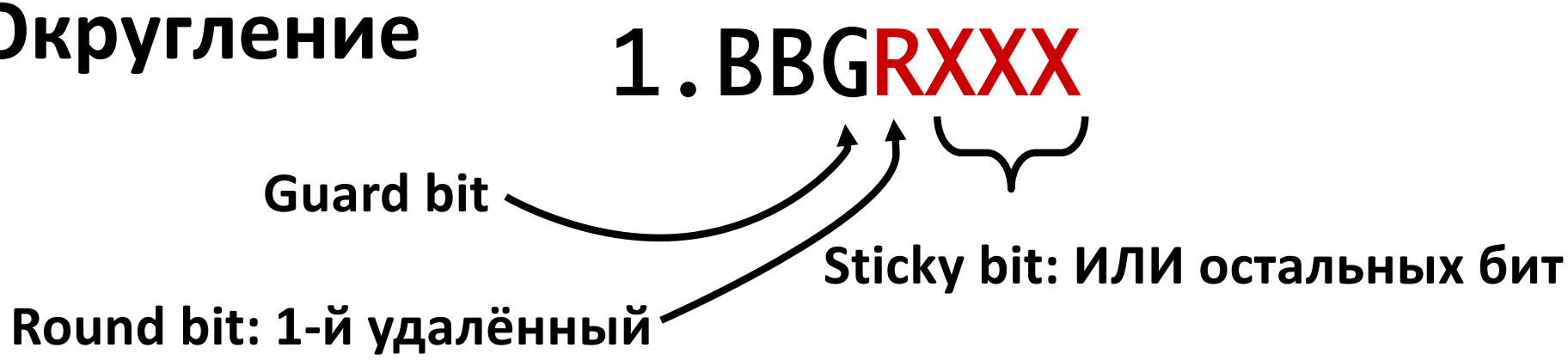
128	10000000
13	00001101
15	00001111
33	00010001
35	00010011
138	10001010
63	00111111

Нормализация

s	exp	frac
1	4 бита	3 бита

Значение	Двоичное	Дробь	Порядок
128	10000000	1.0000000	7
15	00001111	1.1110000	3
17	00010001	1.0001000	4
19	00010011	1.0011000	4
138	10001010	1.0001010	7
63	00111111	1.1111100	5

Округление



- **Правила округления**
  - Round = 1, Sticky = 1 → > 0.5
  - Guard = 1, Round = 1, Sticky = 0 → округление к четному

Значение	Дробь	GRS	Добав.	Округлённое
128	1.0000000	000	N	1.000
15	1.1110000	100	N	1.111
17	1.0001000	010	N	1.000
19	1.0011000	110	Y	1.010
138	1.0001010	011	Y	1.001
63	1.1111100	111	Y	10.000

Перенормализация

- **Проблема**
  - Округление может вызвать переполнение
  - Решается одиночным сдвигом точки вправо и увеличением порядка

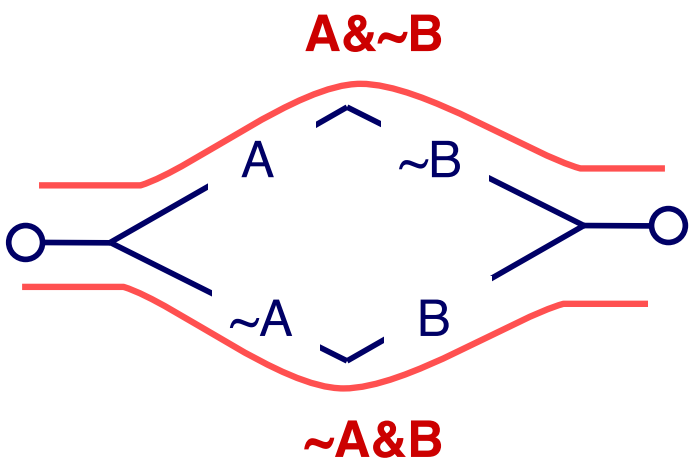
Значение	Округленное	Пор.	Уточнённое	Результат
128	1.000	7		128
15	1.111	3		15
17	1.000	4		16
19	1.010	4		20
138	1.001	7		134
63	10.000	5	1.000	64 (пор. 6)

Интересные числа {одинарные , двойные}

Описание	*exp*	*frac*	*Численное значение*
Ноль	**00…00**	**00…00**	**0.0**
Наименьшее денормализ.	**00…00**	**00…01**	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
▪ Одинарное ≈ 1.4×10^{-45}			
▪ Двойное ≈ 4.9×10^{-324}			
Наибольшее денормализ.	**00…00**	**11…11**	$(1.0 - \varepsilon) \times 2^{-\{126,1022\}}$
▪ Одинарное ≈ 1.18×10^{-38}			
▪ Двойное ≈ 2.2×10^{-308}			
Наименьшее нормализ.	**00…01**	**00…00**	$1.0 \times 2^{-\{126,1022\}}$
▪ Ближайшее большее наибольшего денормализованного			
Единица	**01…11**	**00…00**	**1.0**
Наибольшее нормализ.	**11…10**	**11…11**	$(2.0 - \varepsilon) \times 2^{\{127,1023\}}$
▪ Одинарное ≈ 3.4×10^{38}			
▪ Двойное ≈ 1.8×10^{308}			

Приложение булевой алгебры

- **Клодом Шенноном применена к цифровым системам**
  - Диплом MIT 1937 год
  - Рассмотрены схемы реле
    - Замкнутый контакт кодируется как 1, разомкнутый как 0



Контакт есть если

A&~B I ~A&B

= A^B

Пример небезопасного кода

```
/* Участок памяти ядра содержит данные доступные пользователю */
#define KSIZE 1024
char kbuf[KSIZE];

/* Копируем не более maxlen байт из области ядра в буфер пользователя */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Счётчик байт len – минимальное из размера буфера и maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

- **Аналог кода найденного в реализации getpeername FreeBSD's**
- **Легионы способных людей пытаются находить и находят уязвимости в давно и массово работающих программах**

Типичное применение

```
/* Участок памяти ядра содержит данные доступные пользователю */
#define KSIZE 1024
char kbuf[KSIZE];

/* Копируем не более maxlen байт из области ядра в буфер пользователя */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Счётчик байт len – минимальное из размера буфера и maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, MSIZE);
    printf("%s\n", mybuf);
}
```

Вредоносное применение

```
/* Определение библиотечной функции memcpy */
void *memcpy(void *dest, void *src, size_t n);
```

```
/* Участок памяти ядра содержит данные доступные пользователю */
#define KSIZE 1024
char kbuf[KSIZE];

/* Копируем не более maxlen байт из области ядра в буфер пользователя */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Счётчик байт len – минимальное из размера буфера и maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, -MSIZE);
    . . .
}
```

Математические свойства UAdd

- **Сложение по модулю образует абелеву группу**
  - **Замкнута** относительно сложения

    $0 \le \mathrm{UAdd}_w(u, v) \le 2^w - 1$
  - **Коммутативна**

    $\mathrm{UAdd}_w(u, v) = \mathrm{UAdd}_w(v, u)$
  - **Ассоциативна**

    $\mathrm{UAdd}_w(t, \mathrm{UAdd}_w(u, v)) = \mathrm{UAdd}_w(\mathrm{UAdd}_w(t, u), v)$
  - **0** является нейтральным элементом

    $\mathrm{UAdd}_w(u, 0) = u$
  - Любому элементу есть **обратный**
    - Let $\mathrm{UComp}_w(u) = 2^w - u$

      $\mathrm{UAdd}_w(u, \mathrm{UComp}_w(u)) = 0$

Математичские свойства TAdd

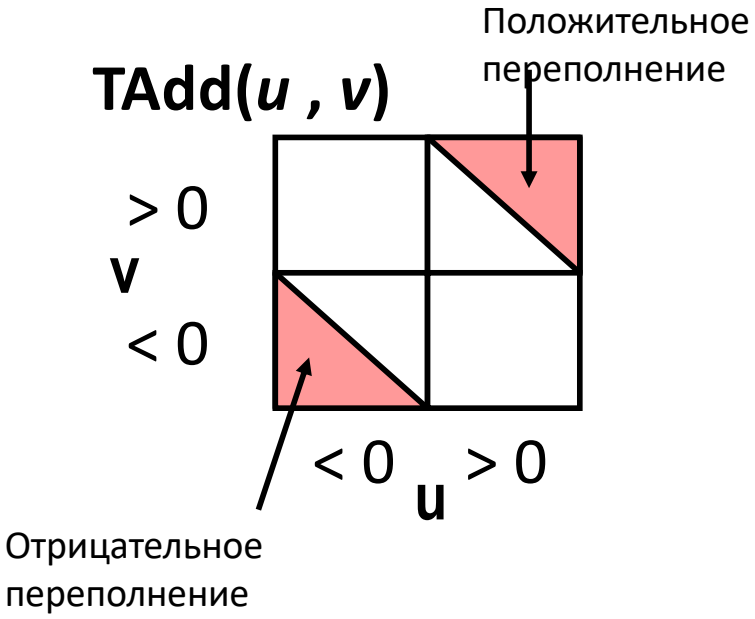
- **Группа изоморфная беззнаковым с UAdd**
  - $TAdd_w(u, v) = U2T(UAdd_w(T2U(u), T2U(v)))$
    - т.к. битовые преобразования идентичны

- **Дополнительны коды с TAdd образуют группу**
  - Замкнута, коммутативна, ассоциативна, 0 – нейтральный элемент
  - Для каждого элемента есть обратный

$$TComp_w(u) = \begin{cases} -u & u \neq TMin_w \\ TMin_w & u = TMin_w \end{cases}$$

Определение TAdd

- **Функциональность**
  - Полное суммирование требует *w*+1 бит
  - Отбрасывание MSB
  - Интерпретация оставшихся бит как целого числа в дополнительном коде



$$TAdd_w(u,v) = \begin{cases} u+v+2^w & u+v < TMin_w \\ u+v & TMin_w \le u+v \le TMax_w \\ u+v-2^w & TMax_w < u+v \end{cases}$$

Переполнение отрицательными

Переполнение положительными

Изменение знака: Инверсия и увеличение

- **Утверждение: для дополнительного кода верно**

  `~x + 1 == -x`

- **Инверсия**
  - Важно: `~x + x == 1111…111 == -1`

	x	1	0	0	1	1	1	0	1
+	~x	0	1	1	0	0	0	1	0
	-1	1	1	1	1	1	1	1	1

Примеры инверсии и увеличения

x = 15213

	Десятичное	Шестнадцатиричное	Двоичное
x	15213	3B 6D	00111011 01101101
~x	-15214	C4 92	11000100 10010010
~x+1	-15213	C4 93	11000100 10010011
y	-15213	C4 93	11000100 10010011

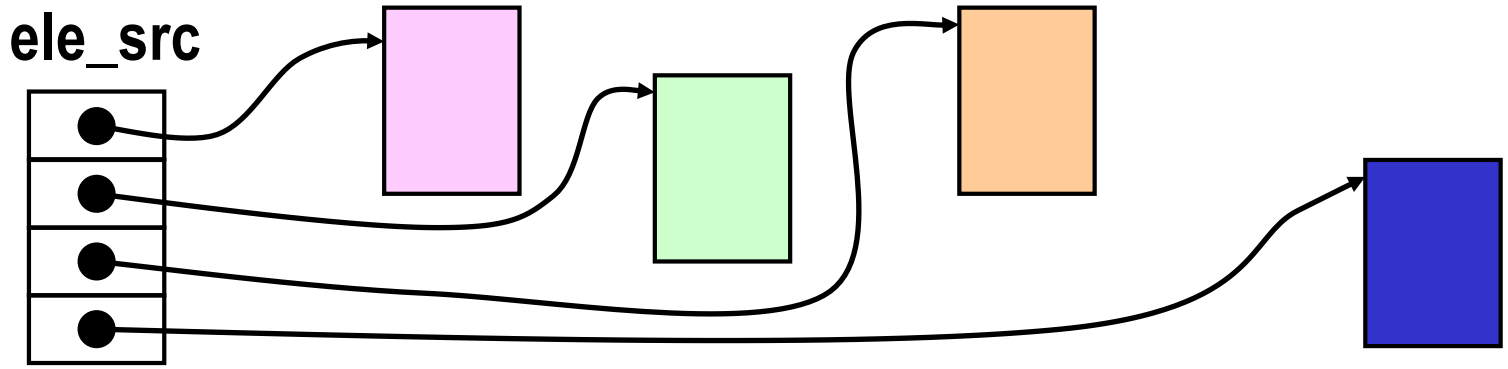
x = 0

	Десятичное	Шестнадцатиричное	Двоичное
0	0	00 00	00000000 00000000
~0	-1	FF FF	11111111 11111111
~0+1	0	00 00	00000000 00000000

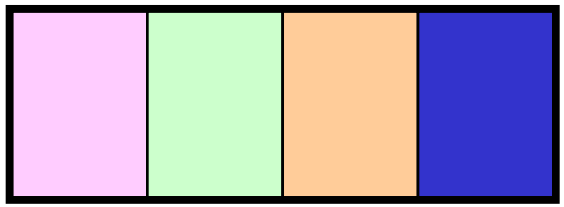
Пример небезопасного кода №2

- **Библиотека SUN XDR**
  - Широко применяется для межмашинного обмена данными

```
void* copy_elements(void *ele_src[], int ele_cnt, size_t ele_size);
```



malloc(ele_cnt * ele_size)



Код XDR

```
void* copy_elements(void *ele_src[], int ele_cnt, size_t ele_size) {
    /*
     * Занимаем буфер для ele_cnt объектов, каждый ele_size байт
     * и копируем с места указанного ele_src
     */
    void *result = malloc(ele_cnt * ele_size);
    if (result == NULL)
        /* Ошибка malloc */
        return NULL;
    void *next = result;
    int i;
    for (i = 0; i < ele_cnt; i++) {
        /* Копирование объектов по назначению */
        memcpy(next, ele_src[i], ele_size);
        /* Перемещение указателя на следующую позицию в памяти */
        next += ele_size;
    }
    return result;
}
```

Уязвимость XDR

malloc(ele_cnt * ele_size)

- **Что если:**
  - **`ele_cnt`** = $2^{20} + 1$
  - **`ele_size`** = $4096 = 2^{12}$
  - Выделится = ??

- **Как сделать функцию безопасной?**

Компилированный код умножения

Функция Си

```
int mul12(int x)
{
  return x*12;
}
```

Арифметические операции в результате компиляции

```
  leal (%eax,%eax,2), %eax
  sall $2, %eax
```

Пояснение

```
  t <- x+x*2
  return t << 2;
```

- **Компилятор Си автоматически создаёт код сдвига и сложения при умножении на константу**

Компилированный код деления беззнакового

Функция Си

```
unsigned udiv8(unsigned x)
{
  return x/8;
}
```

Арифметические операции в результате компиляции

```
  shrl $3, %eax
```

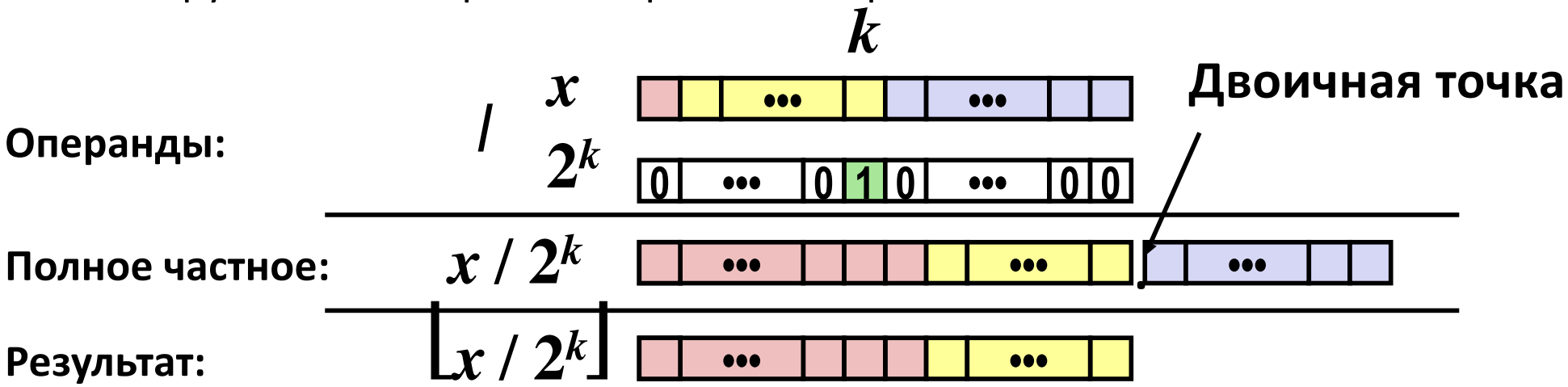
Пояснение

```
  # Логический сдвиг
  return x >> 3;
```

- **Для беззнаковых используется логический сдвиг**
- **Для пользователей Явы**
  - Логический сдвиг пишется как >>>

Деление сдвигом знакового на степень двойки

- **Частное от деления знакового на степень двойки**
  - `x >> k` даёт $\lfloor x/2^k \rfloor$
  - Использует арифметический сдвиг
  - Округляет в неверном направлении при `u < 0`

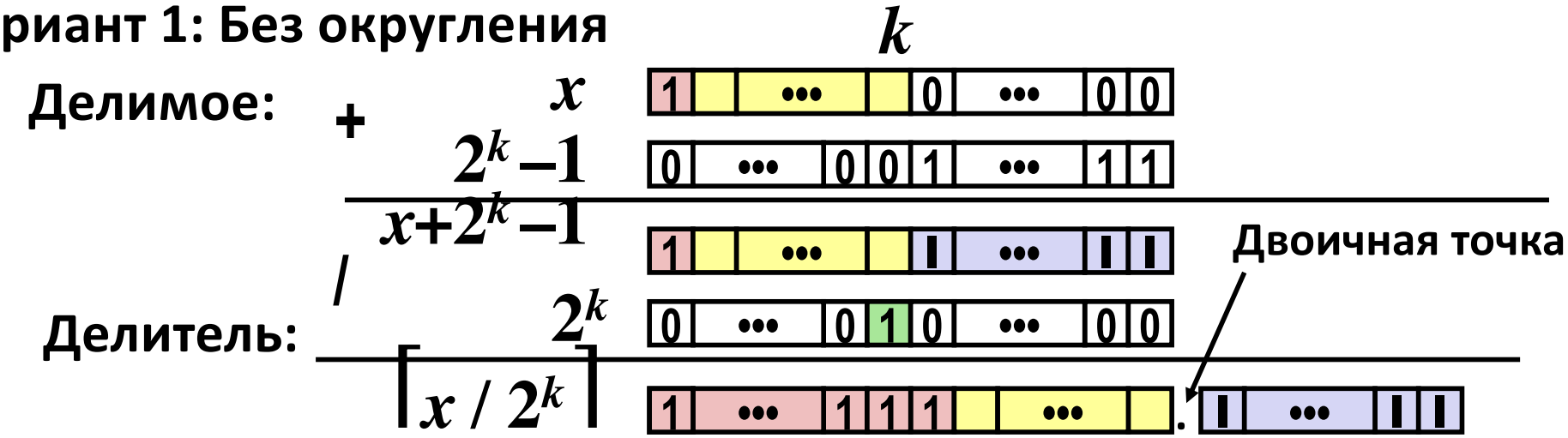


	Полное частное	Результат	Шестн.	Двоичный
`y`	-15213	-15213	`C4 93`	`11000100 10010011`
`y >> 1`	-7606.5	-7607	`E2 49`	`11100010 01001001`
`y >> 4`	-950.8125	-951	`FC 49`	`11111100 01001001`
`y >> 8`	-59.4257813	-60	`FF C4`	`11111111 11000100`

Правильное деление на степень двойки(1)

- **Частное от деления знакового на степень двойки**
  - Необходимо $\lfloor x/2^k \rfloor$(Округление к 0)
  - Вычисляется как $\lfloor (x+2^k-1)/2^k \rfloor$
    - В Си: `(x + (1<<k)-1) >> k`
    - Смещение делимого к 0

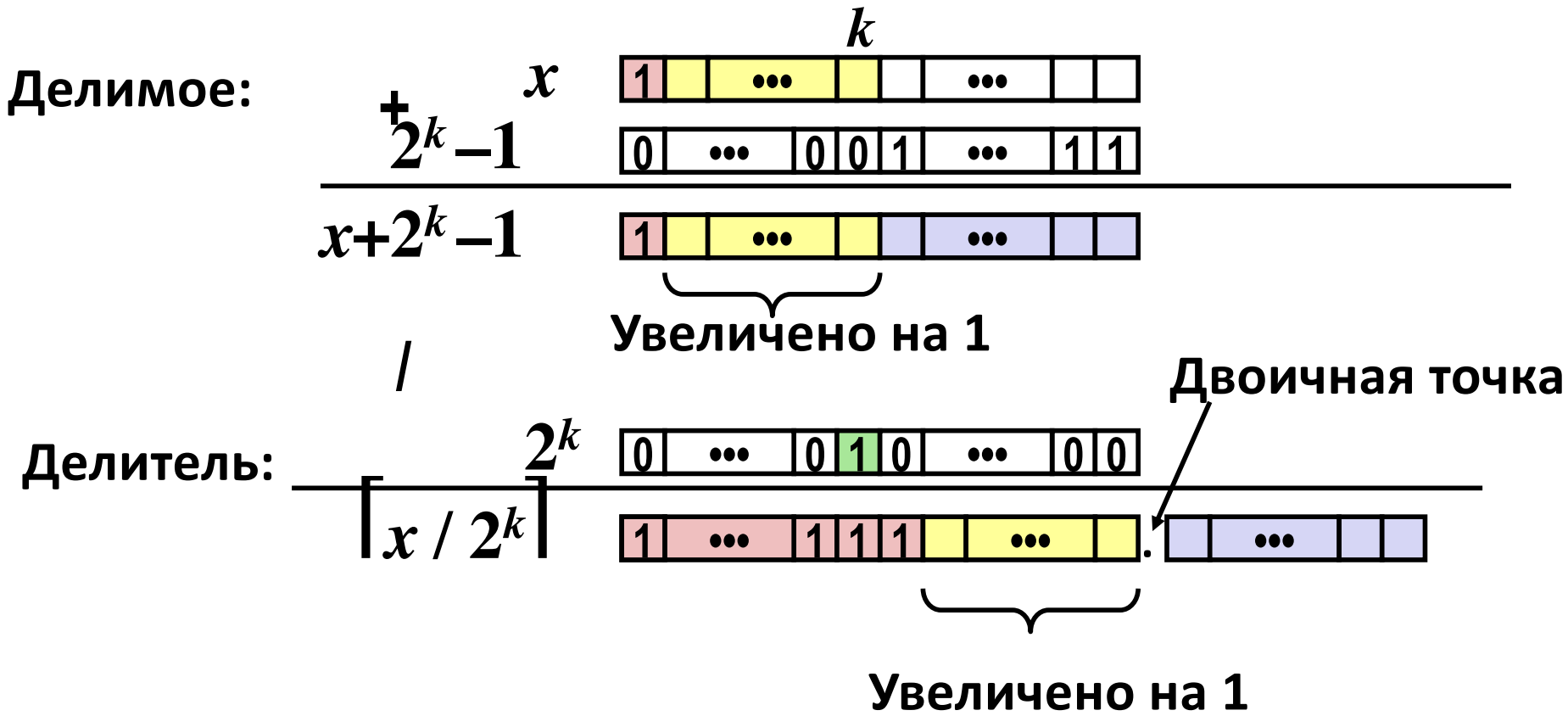
Вариант 1: Без округления



Смещение не имеет эффекта

Правильное деление на степень двойки(2)

Вариант 2: округление



Смещение добавляет 1 к результату

Компилированный код деления знакового

Функция Си

```
int idiv8(int x)
{
  return x/8;
}
```

Арифметические операции в результате компиляции

```
  testl %eax, %eax
  js   L4
L3:
  sarl $3, %eax
  ret
L4:
  addl $7, %eax
  jmp  L3
```

Пояснения

```
  if x < 0
    x += 7;
  # Арифметический сдвиг
  return x >> 3;
```

- **Арифметический сдвиг int**
- **Для пользователей Ява**
  - Арифм. сдвиг пишется как >>

Арифметика: основные правила(2)

- **Беззнаковые и знаковые в доп.коде – изоморфные кольца: изоморфизм = преобразование типов**

- **Сдвиг влево**
  - Беззнаковые: умножение на 2^k
  - Всегда логический сдвиг

- **Сдвиг вправо**
  - Беззнаковые: логический сдвиг, деление на 2^k с округлением к 0
  - Знаковые: арифметический сдвиг
    - Положительные: деление на 2^k с округлением к 0
    - Отрицательные: деление на 2^k с округлением от 0 используется смещение для исправления

Свойства арифметики беззнаковых

- **Умножение и сложение беззнаковых образуют коммутативное кольцо**
  - Сложение – коммутативная группа
  - Замкнутая относительно умножения

    $0 \le \mathrm{UMult}_w(u, v) \le 2^w - 1$
  - Умножение – коммутативно

    $\mathrm{UMult}_w(u, v) = \mathrm{UMult}_w(v, u)$
  - Умножение – ассоциативно

    $\mathrm{UMult}_w(t, \mathrm{UMult}_w(u, v)) = \mathrm{UMult}_w(\mathrm{UMult}_w(t, u), v)$
  - 1 – нейтральный элемент умножения

    $\mathrm{UMult}_w(u, 1) = u$
  - Умножение дистрибутивно относительно сложения

    $\mathrm{UMult}_w(t, \mathrm{UAdd}_w(u, v)) = \mathrm{UAdd}_w(\mathrm{UMult}_w(t, u), \mathrm{UMult}_w(t, v))$

Свойства арифметики знаковых в доп.коде

- **Изморфные алгебры**
  - Беззнаковые умножение и сложения
    - Сокращаются до w
  - Умножения и сложения знаковых в дополнительном коде
    - Сокращаются до w
- **Обе образуют кольца**
  - Изоморфные кольцу вычетов по модулю 2^w
- **Сравнение с арифметикой (математических) целых**
  - И там, и там – кольца
  - Целые обладают свойствами упорядоченности, т.е.,

    $u > 0 \quad \Rightarrow \quad u + v > v$

    $u > 0, v > 0 \quad \Rightarrow \quad u \cdot v > 0$
  - Этими свойствами не обладают знаковые в доп.коде

    TMax `+ 1 == ` *TMin*

    `15213 * 30426 == -10030` (для 16-битных слов)

Чтение байт в обратном порядке

- **Результат дизассемблирования**
  - Текстовое представление машинного кода
  - Выдаётся программой читающей машинный код
- **Пример фрагмента**

Адрес	Машинный код	Представление на Ассемблере
8048365:	5b	pop %ebx
8048366:	81 c3 ab 12 00 00	add $0x12ab,%ebx
804836c:	83 bb 28 00 00 00 00	cmpl $0x0,0x28(%ebx)

- **«Расшифровка» значений**
  - Значение: 0x12ab
  - Размещение в слове (32 бита): 0x000012ab
  - Разделение на байты: 00 00 12 ab
  - Переупорядочение: ab 12 00 00