Лекция №3

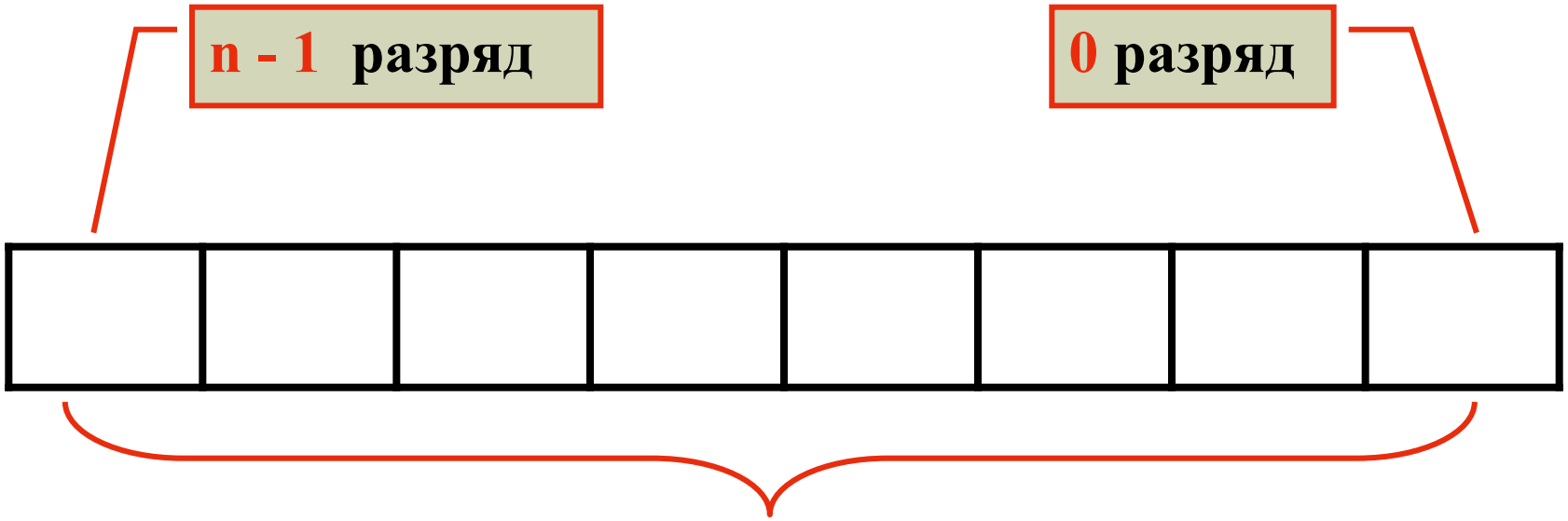
Представление информации

Вся информация в памяти ЭВМ представляется в форме *цифрового двоичного кода*

1	1	0	0	1	1	1	1	0	0	0	1	0	1	0	0
0	1	0	0	1	1	1	0	1	0	0	0	1	1	1	1
1	1	0	1	1	0	1	1	0	1	1	0	0	1	1	1
1	1	0	0	1	0	1	0	0	0	1	1	0	1	1	1
1	1	0	0	0	1	1	0	1	0	0	0	1	0	1	0
1	1	1	0	0	1	0	1	0	1	1	1	0	0	0	1
1	0	1	0	1	0	1	0	1	1	1	1	1	0	0	0
1	1	0	1	0	1	0	1	1	0	0	1	1	1	0	0
0	1	0	1	0	0	1	1	1	1	1	0	0	0	1	1
1	0	0	1	1	0	0	0	1	1	1	0	1	0	1	0
1	0	0	0	1	1	0	1	0	1	0	1	0	1	0	0
1	1	0	0	1	1	1	0	0	1	1	0	1	0	1	1
0	0	1	1	1	0	0	1	1	0	1	0	1	0	1	0
0	1	1	1	1	0	1	0	1	0	1	0	0	1	1	1
0	0	1	1	1	0	1	0	1	0	1	0	1	1	1	0
1	1	1	1	1	0	1	0	1	1	1	0	0	1	1	1

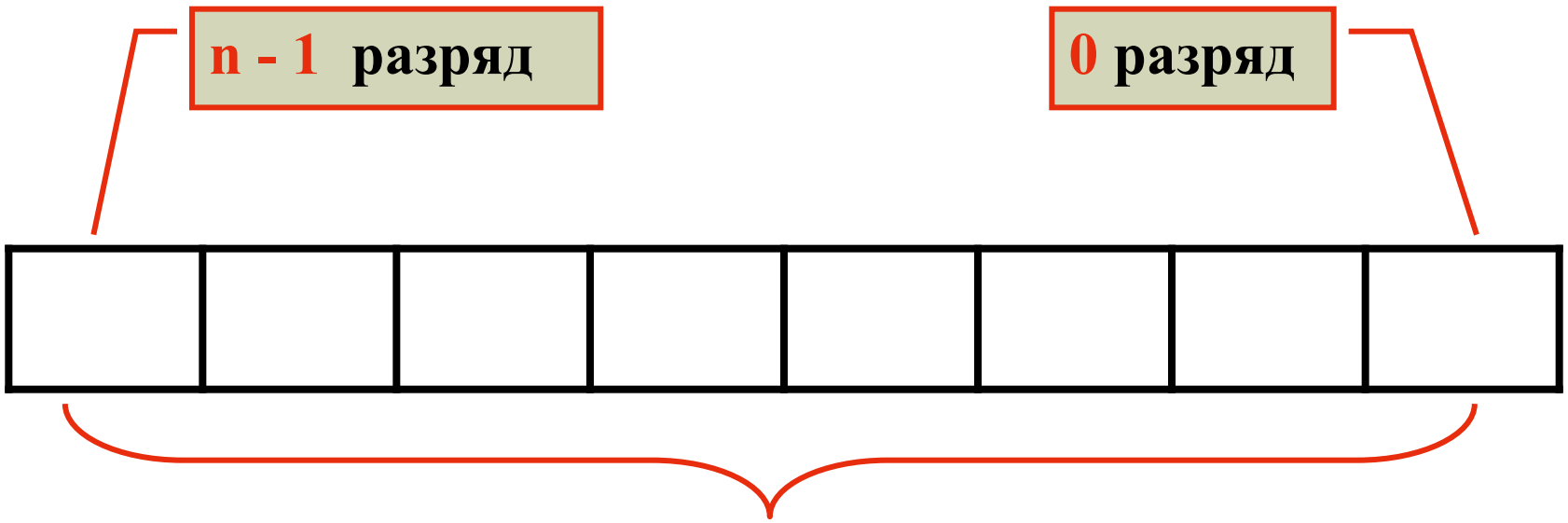
Образ компьютерной памяти

Ячейка *– это часть памяти компьютера, вмещающая в себя информацию, доступную для обработки* ***отдельной командой*** *процессора.*



Содержимое ячейки памяти называется ***машинным словом****.*

Ячейка памяти разделяется на ***разряды****, в каждом из которых хранится разряд числа.*

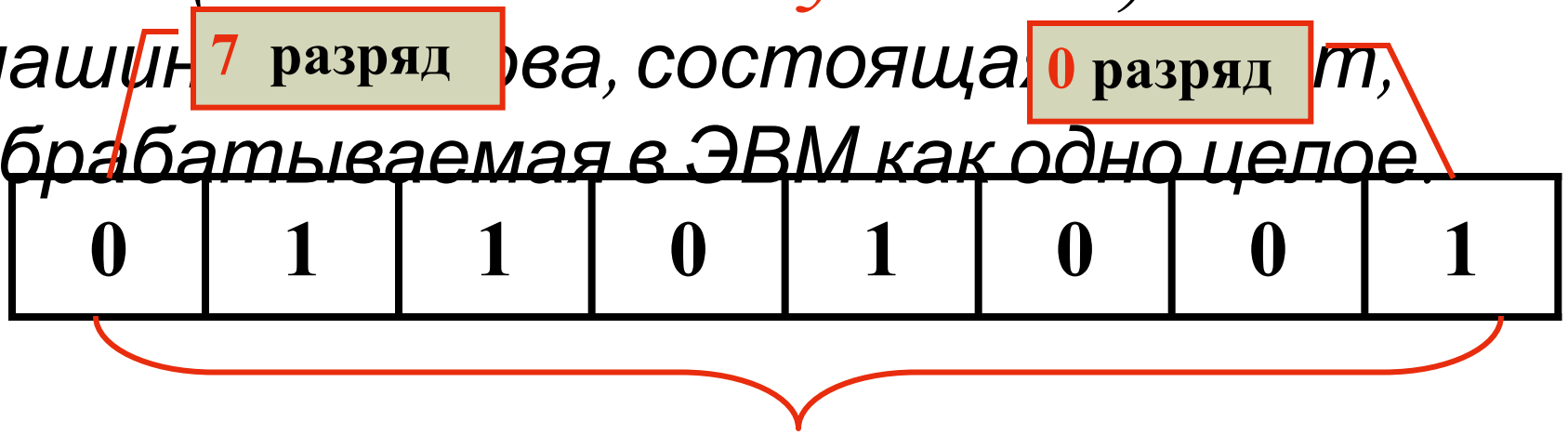


Бит (от английского *binary digit* — двоичная цифра) - *минимальная единица измерения информации. Каждый бит может принимать значение 0 или 1.*

Битом *также называют* **разряд** *ячейки памяти ЭВМ.*

8 бит = 1 байт

Байт (от английского *byte* – слог) – *часть машинного слова, состоящая из 8 бит, обрабатываемая в ЭВМ как одно целое.*



Целые числа в компьютере

Целые беззнаковые числа

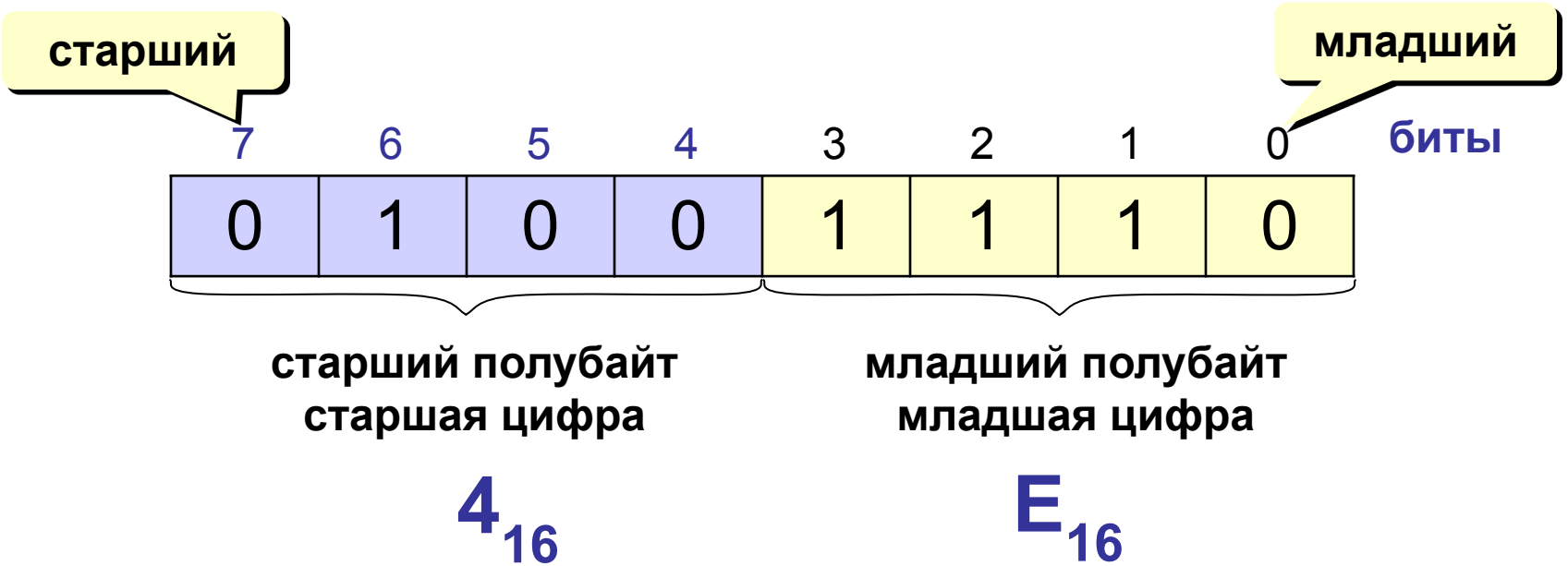
Беззнаковые данные – не могут быть отрицательными.

Байт (символ)

память: **1 байт = 8 бит**

диапазон значений **0…255,** **$0…FF_{16} = 2^8 - 1$**

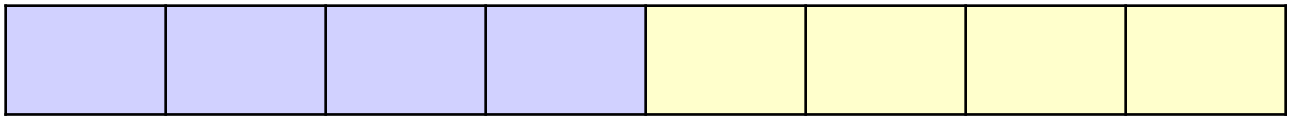
Си: ***unsigned char*** **Паскаль:** ***byte***



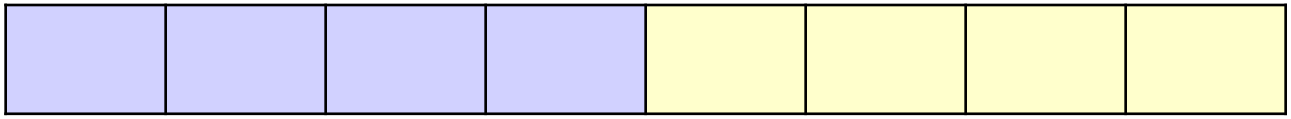
$$1001110_2 = 4E_{16} = \text{‘N’}$$

Примеры

78 =



115 =



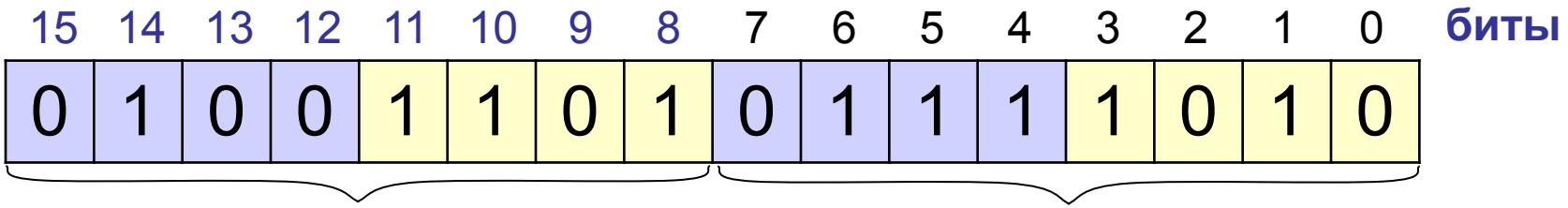
Целые беззнаковые числа

Целое без знака

память: **2 байта = 16 бит**

диапазон значений **0…65535, $0…FFFF_{16} = 2^{16}-1$**

Си: ***unsigned short int*** **Паскаль:** ***word***



$4D_{16}$ $7A_{16}$

$$100110101111010_2 = 4D7A_{16}$$

Длинное целое без знака

память: **4 байта = 32 бита**

диапазон значений **$0…FFFFFFFF_{16} = 2^{32}-1$**

Си: ***unsigned int*** **Паскаль:** ***dword, longword***

Целые числа со знаком

Сколько места требуется для хранения знака?

Старший (знаковый) бит числа определяет его знак. Если он равен 0, число положительное, если 1, то отрицательное.

«-1» – это такое число, которое при сложении с 1 даст 0.

1 байт: $FF_{16} + 1 = 1\,0\,0_{16}$ (−1)

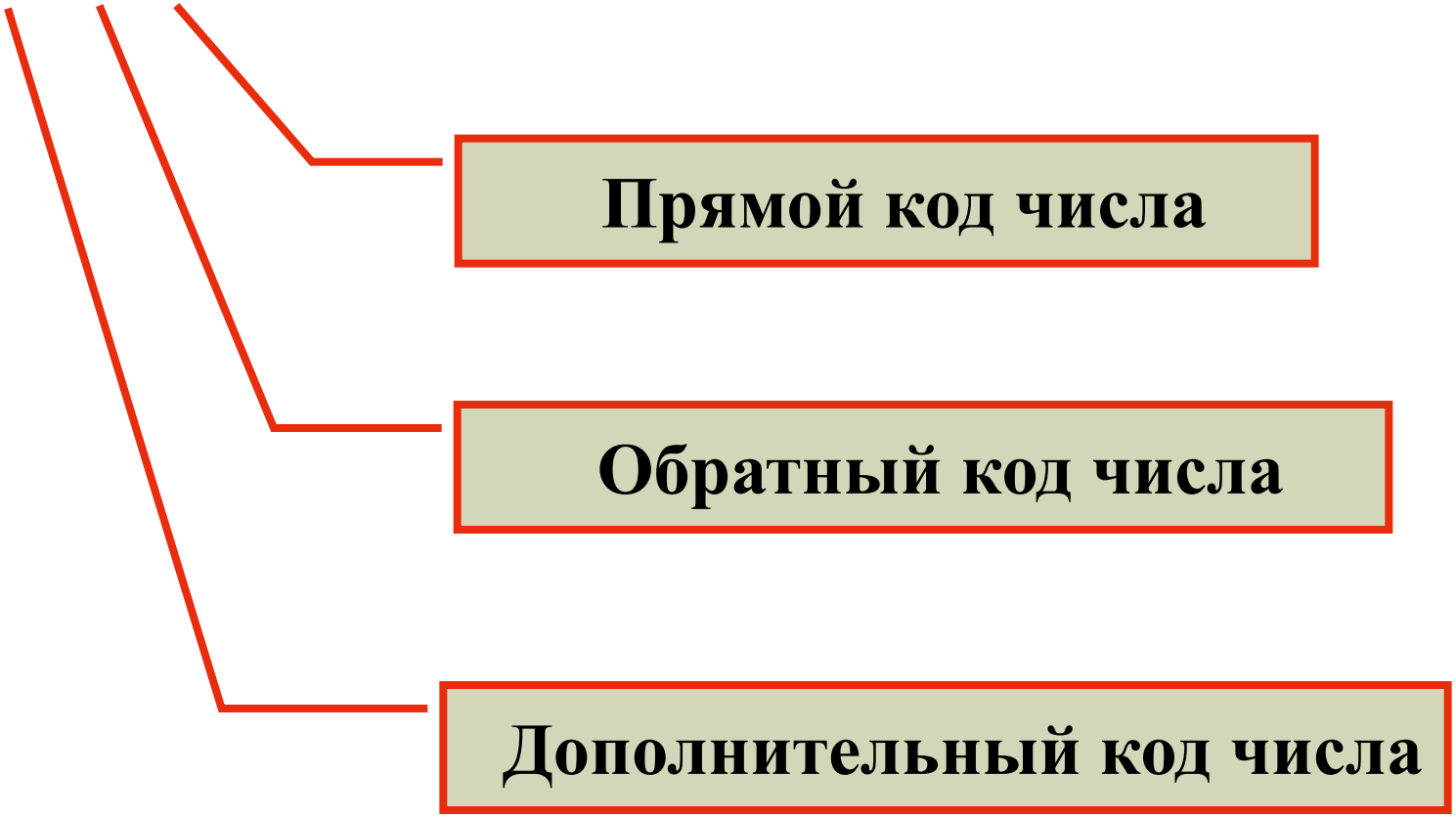
не помещается в 1 байт!

2 байта: $FFFF_{16} + 1 = 1\,0\,0\,0\,0_{16}$

4 байта: $FFFFFFFF_{16} + 1 = 1\,0\,0\,0\,0\,0\,0\,0\,0_{16}$

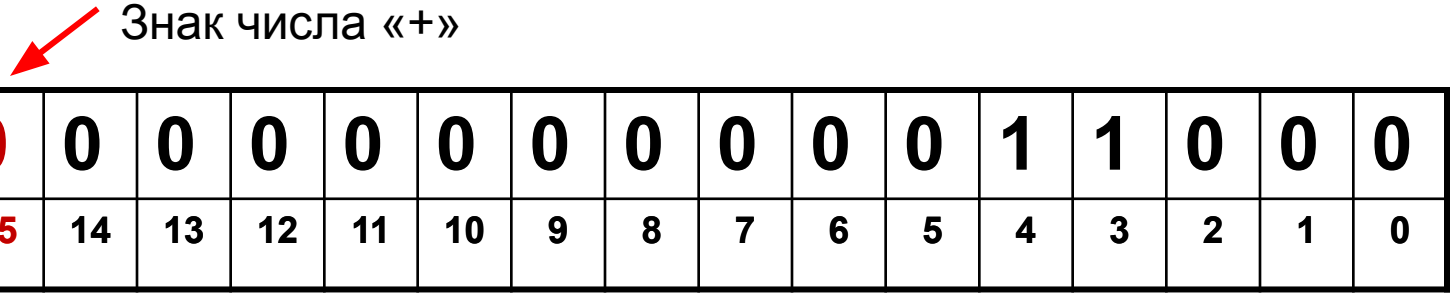
−1

В ЭВМ в целях упрощения выполнения арифметических операций применяют ***специальные коды*** *для представления* ***целых чисел***



Прямой код – это представление числа в двоичной системе счисления, при этом первый разряд отводится под знак числа. Если число положительное, то в первом разряде находится 0, если число отрицательное, в первом разряде указывается 1.

Положительное десятичное число **24** представляется



Обратный код для положительного числа в двоичной системе счисления совпадает с прямым кодом.

Для отрицательного числа все цифры числа заменяются на противоположные (1 на 0, 0 на 1), а в знаковый разряд заносится единица.

Отрицательное десятичное число **-24** представляется

1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Знак числа «-»

Дополнительный код используют в основном для представления в компьютере отрицательных чисел.

Алгоритм получения дополнительного кода для отрицательного числа

1. Найти прямой код числа (перевести число в двоичную систему счисления число без знака)

2. Получить обратный код. Поменять каждый ноль на единицу, а единицу на ноль (инвертировать число)

3. К обратному коду прибавить 1

Найдем дополнительный код десятичного числа - 47

1. *Найдем двоичную запись числа 47 (прямой код)*

0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

2. *Инвертируем это число (обратный код)*

1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

3. *Прибавим 1 к обратному коду и получим запись этого числа в оперативной памяти*

1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Двоичный дополнительный код

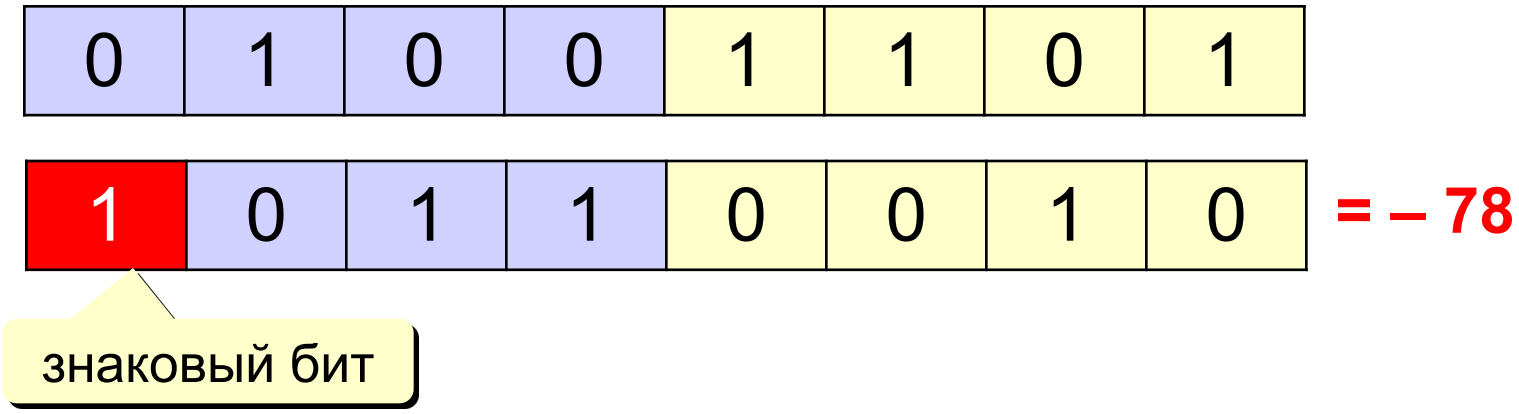
Задача: представить отрицательное число **(–a)** в двоичном дополнительном коде.

Решение:

1. Перевести число **a–1** в **двоичную систему**.
2. Записать результат **в разрядную сетку** с нужным числом разрядов.
3. Заменить все «0» на «1» и наоборот (***инверсия***).

Пример: **(– a) = – 78**, сетка 8 бит

4. **a – 1 = 77 = 1001101_2**
5.
6.



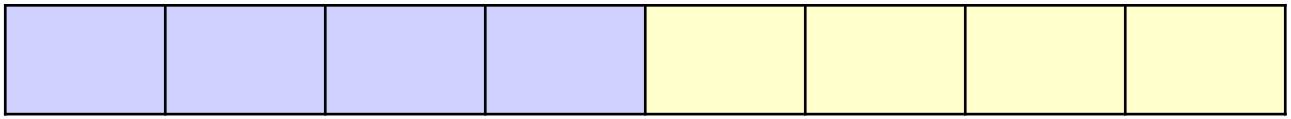
Двоичный дополнительный код

Проверка: 78 + (– 78) = ?

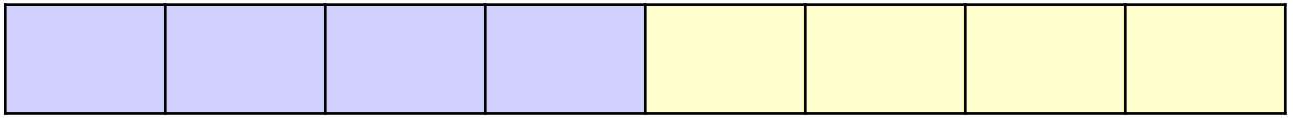
78 =	0	1	0	0	1	1	1	0
+								
– 78 =	1	0	1	1	0	0	1	0

Пример

(– a) = – 123, сетка 8 бит



– 123 =

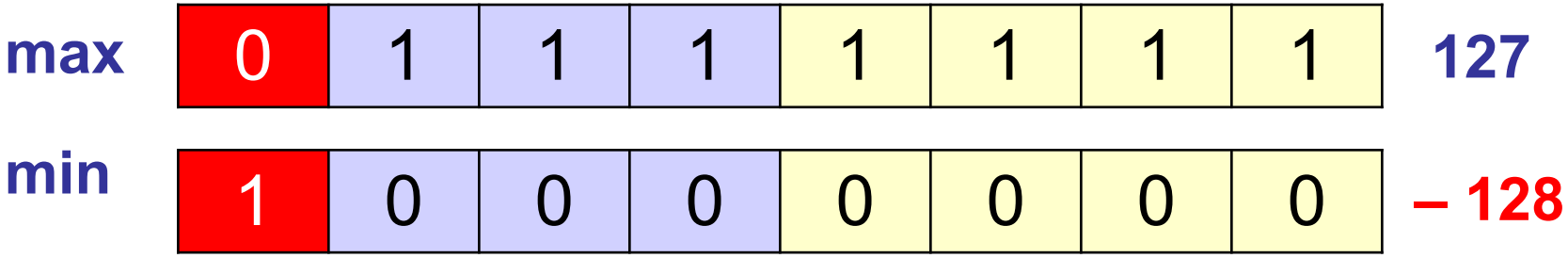


Целые числа со знаком

Байт (символ) со знаком

память: **1 байт = 8 бит**

диапазон значений:



– 128 = – 2^7 … 127 = 2^8 – 1

Си: ***char*** **Паскаль:** ***shortint***

можно работать с отрицательными числами

уменьшился диапазон положительных чисел

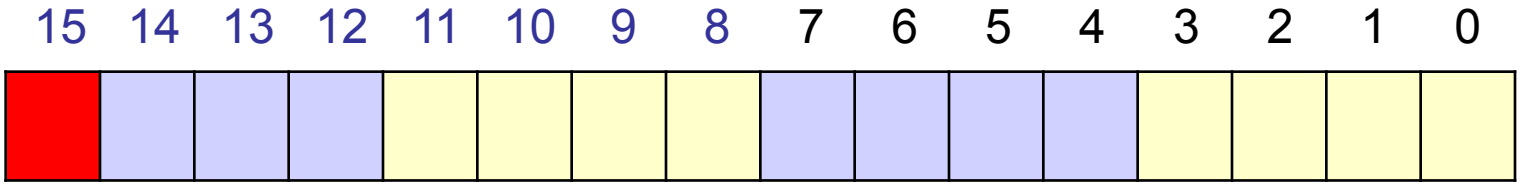
Целые числа со знаком

Слово со знаком

память: **2 байта = 16 бит**

диапазон значений

– 32768 … 32767



Си: ***short int*** **Паскаль:** ***smallint***

Двойное слово со знаком

память – **4 байта**

диапазон значений

– 2^{31} … 2^{31}-1

Си: ***int*** **Паскаль:** ***integer***

Ошибки

Переполнение разрядной сетки: в результате сложения больших положительных чисел получается отрицательное (**перенос в знаковый бит**).

	7	6	5	4	3	2	1	0	
	0	1	0	0	0	0	0	0	64
+	0	1	0	0	0	0	0	0	64
	1	0	0	0	0	0	0	0	– 128

Ошибки

Перенос: при сложении больших (по модулю) отрицательных чисел получается положительное (перенос **за границы разрядной сетки**).

	7	6	5	4	3	2	1	0	
	1	0	0	0	0	0	0	0	– 128
+	1	0	0	0	0	0	0	0	– 128
1	0	0	0	0	0	0	0	0	0

в специальный **бит переноса**

Вещественные числа в компьютере

Перевод дробных чисел

10 → 2

$0{,}375 = 0{,}011_2$

$\times \quad 2$

$0{,}750$

$\times \quad 2$

$1{,}500$

$\times \quad 2$

$1{,}0$

$0{,}7 = ?$

$0{,}7 = 0{,}1011\mathbf{0}0110\ldots$

$= 0{,}1(0110)_2$

Многие дробные числа нельзя представить в виде **конечных** двоичных дробей.

Для их точного хранения требуется **бесконечное** число разрядов.

Большинство дробных чисел хранится в памяти с ошибкой.

2 → 10

$2^{-2} = 1/2^2 = 0{,}25$

2 1 0 -1 -2 -3 **разряды**

$101{,}011_2 = 1\cdot2^2 + 1\cdot2^0 + 1\cdot2^{-2} + 1\cdot2^{-3}$

$= 4 + 1 + 0{,}25 + 0{,}125 = 5{,}375$

Нормализация двоичных чисел

$$X = s \cdot M \cdot 2^e$$

s – **знак** (1 или -1)

M – **мантисса**, $M = 0$ **или** $1 \le M < 2$

e – **порядок**

Пример:

знак | мантисса | порядок

$$15{,}625 = 1111{,}101_2 = 1 \cdot 1{,}111101_2 \cdot 2^3$$

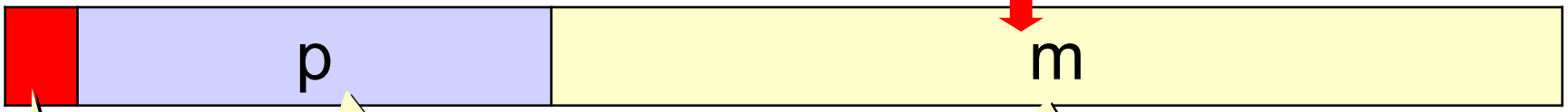
$$3{,}375 =$$

Нормализованные числа в памяти

IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754)

$$15{,}625 = 1 \cdot 1{,}111101_2 \cdot 2^3$$

$s = 1$ $e = 3$ $M = 1{,}111101_2$



Порядок со сдвигом:
p = e + E (сдвиг)

Дробная часть мантиссы:
m = M – 1

Знаковый бит:
0, если s = 1
1, если s = – 1

Целая часть M всегда 1, поэтому не хранится в памяти!

Нормализованные числа в памяти

Тип данных	Размер, байт	Мантисса, бит	Порядок, бит	Сдвиг порядка, **E**	Диапазон модулей	Точность, десятичн. цифр
float **single**	**4**	23	8	127	$3{,}4\cdot10^{-38}$ ... $3{,}4\cdot10^{38}$	**7**
double **double**	**8**	52	11	1023	$1{,}7\cdot10^{-308}$ ... $1{,}7\cdot10^{308}$	**15**
long double **extended**	**10**	64	15	16383	$3{,}4\cdot10^{-4932}$ ... $3{,}4\cdot10^{4932}$	**19**

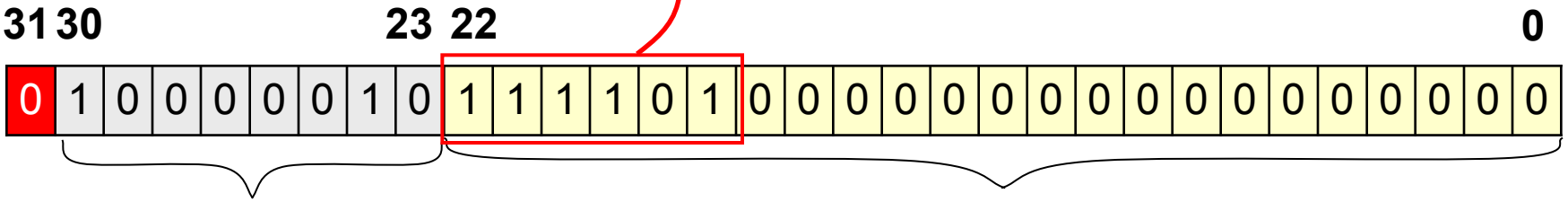
Типы данных для языков: **Си**

Паскаль

Вещественные числа в памяти

$15{,}625 = 1{,}111101_2 \cdot 2^3$

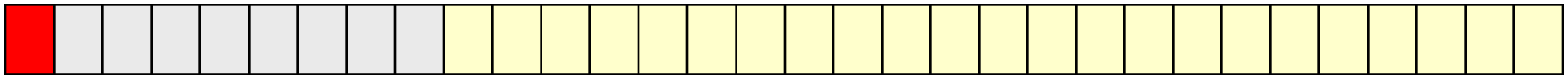
4 байта = 32 бита



$p = e + 127 = 130 = 10000010_2$

$m = M - 1 = 0{,}111101_2$

$3{,}375 =$



Арифметические операции

сложение

$$5{,}5 + 3 = 101{,}1_2 + 11_2 = 8{,}5 = 1000{,}1_2$$

1. Порядок выравнивается до большего

$$5{,}5 = 1{,}011_2 \cdot 2^2$$
$$3 = 1{,}1_2 \cdot 2^1 = 0{,}11_2 \cdot 2^2$$

2. Мантиссы складываются

$$\begin{array}{r} 1{,}011_2 \\ +\ 0{,}110_2 \\ \hline 10{,}001_2 \end{array}$$

3. Результат нормализуется (с учетом порядка)

$$10{,}001_2 \cdot 2^2 = 1{,}0001_2 \cdot 2^3 = 1000{,}1_2 = 8{,}5$$

Арифметические операции

вычитание

$$10{,}75 - 5{,}25 = 1010{,}11_2 - 101{,}01_2 = 101{,}1_2 = 5{,}5$$

1. Порядок выравнивается до большего

$$10{,}75 = 1{,}01011_2 \cdot 2^3$$
$$5{,}25 = 1{,}0101_2 \cdot 2^2 = 0{,}10101_2 \cdot 2^3$$

2. Мантиссы вычитаются

$$\begin{array}{r} 1{,}01011_2 \\ -\ 0{,}10101_2 \\ \hline 0{,}10110_2 \end{array}$$

3. Результат нормализуется (с учетом порядка)

$$0{,}1011_2 \cdot 2^3 = 1{,}011_2 \cdot 2^2 = 101{,}1_2 = 5{,}5$$

Арифметические операции

умножение

$$7 \cdot 3 = 111_2 \cdot 11_2 = 21 = 10101_2$$

1. Мантиссы умножаются

$$7 = 1{,}11_2 \cdot 2^2$$
$$3 = 1{,}1_2 \cdot 2^1$$

$$\begin{array}{r} 1{,}1\,1_2 \\ \times\quad 1{,}1_2 \\ \hline 1\,1\,1_2 \\ 1\,1\,1_2 \\ \hline 1\,0{,}1\,0\,1_2 \end{array}$$

2. Порядки складываются: $2 + 1 = 3$
3. Результат нормализуется (с учетом порядка)

$$10{,}101_2 \cdot 2^3 = 1{,}0101_2 \cdot 2^4 = 10101_2 = 21$$

Арифметические операции

деление

$$17{,}25 : 3 = 10001{,}01_2 : 11_2 = 5{,}75 = 101{,}11_2$$

1. Мантиссы делятся

$$17{,}25 = 1{,}000101_2 \cdot 2^4$$

$$3 = 1{,}1_2 \cdot 2^1$$

$$1{,}000101_2 : 1{,}1_2 = 0{,}10111_2$$

2. Порядки вычитаются: $4 - 1 = 3$

3. Результат нормализуется (с учетом порядка)

$$0{,}10111_2 \cdot 2^3 = 1{,}0111_2 \cdot 2^2 = 101{,}11_2 = 5{,}75$$

Кодирование символов

Текстовый файл

- на экране (символы)
- в памяти – коды

1000001_2	1000010_2	1000011_2	1000100_2
65	66	67	68

В файле хранятся не изображения символов, а их числовые коды!

Файлы со шрифтами: ***.fon**, ***.ttf**, ***.otf**

Кодировка ASCII (7-битная)

ASCII = *American Standard Code for Information Interchange*

Коды 0-127:

0-31 **управляющие символы**:
7 – звонок, 10 – новая строка,
13 – возврат каретки, 27 – Esc.

32 пробел

знаки препинания: **. , : ; ! ?**

специальные знаки: **+ – * / () {} []**

48-57 цифры **0..9**

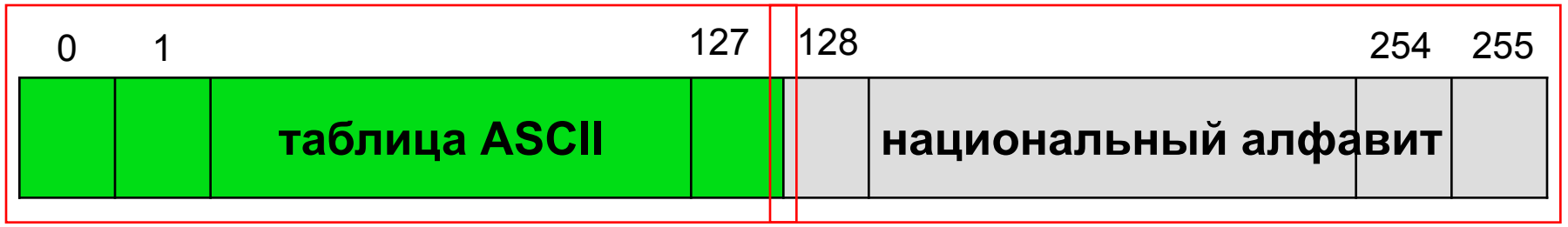
65-90 заглавные латинские буквы **A-Z**

97-122 строчные латинские буквы **a-z**

Где русские буквы?

8-битные кодировки

Кодовые страницы (расширения ASCII):



Для русского языка:

CP-866	для *MS DOS*
CP-1251	для *Windows* (Интернет)
KOI8-R	для *UNIX* (Интернет)
MacCyrillic	для компьютеров *Apple*

Проблема:

Windows-1251	*KOI8-R*
Привет, Вася!	оПХБЕР, бЮЯЪ!
рТЙЧЕФ, чБУС!	Привет, Вася!

8-битные кодировки

- 1 байт на символ – файлы небольшого размера!
- просто обрабатывать в программах

- нельзя использовать символы разных кодовых страниц одновременно (русские и французские буквы, и т.п.)
- неясно, в какой кодировке текст (перебор вариантов!)
- для каждой кодировки нужен свой шрифт (изображения символов)

Стандарт UNICODE

1 112 064 знаков, используются около **100 000**

Windows: **UTF-16**

16 битов на распространённые символы,
32 бита на редко встречающиеся

Linux: **UTF-8**

8 битов на символ для ASCII,
от 16 до 48 бита на остальные

- совместимость с ASCII
- более экономична, чем UTF-16, если много символов ASCII

2010 г. – 50% сайтов использовали UTF-8!

Кодирование графической информации

Растровое кодирование

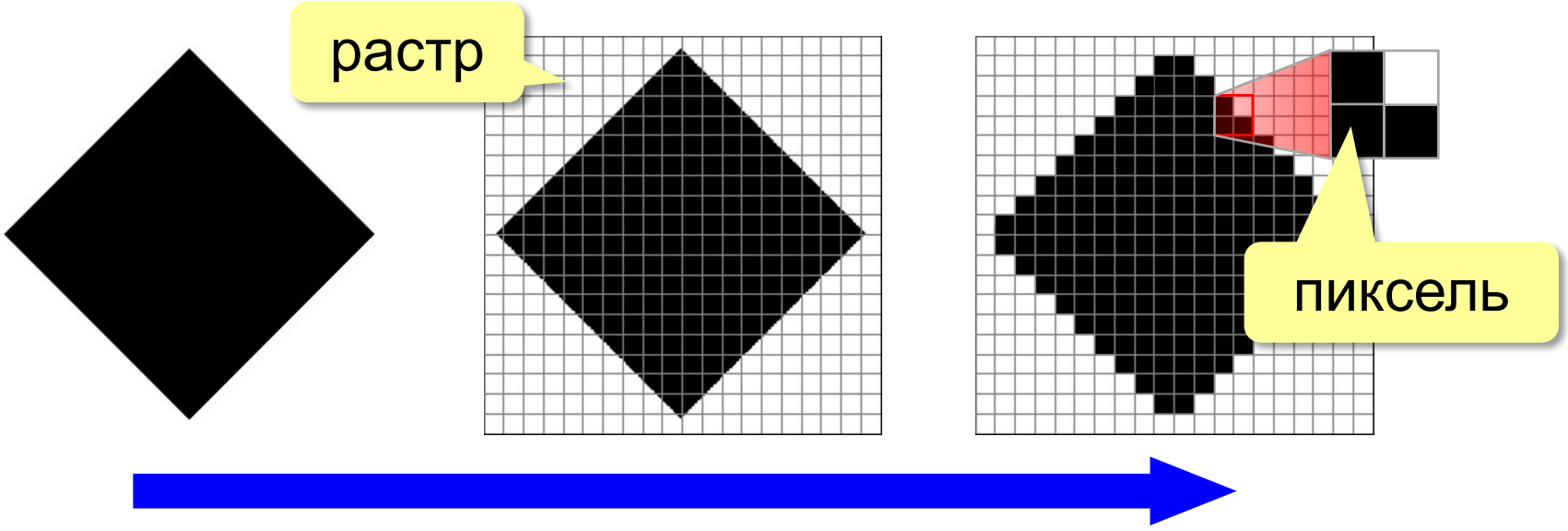
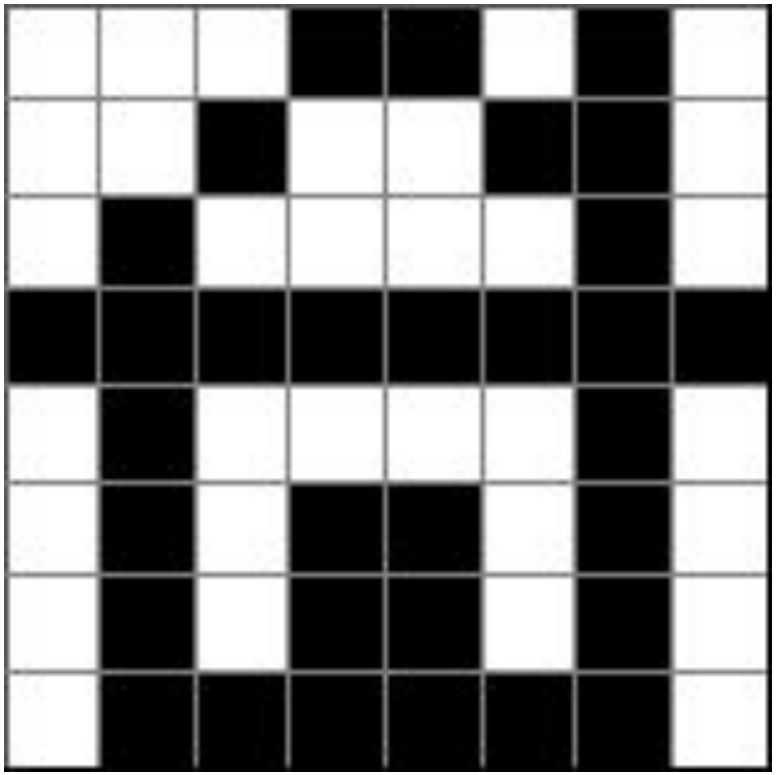


Рисунок искажается!

Пиксель – это наименьший элемент рисунка, для которого можно задать свой цвет.

Растровое изображение – это изображение, которое кодируется как множество пикселей.

Растровое кодирование

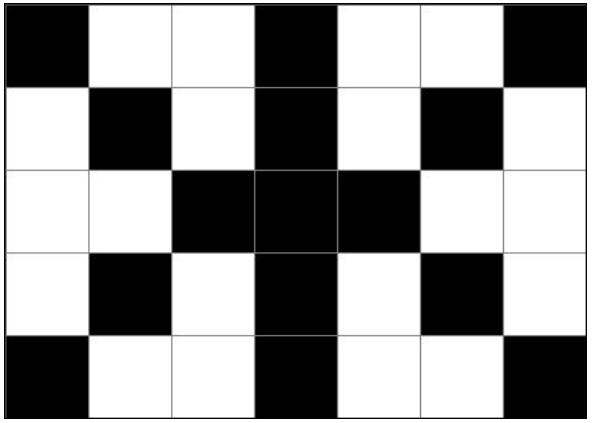


0	0	0	1	1	0	1	0	1A
0	0	1	0	0	1	1	0	26
0	1	0	0	0	0	1	0	42
1	1	1	1	1	1	1	1	FF
0	1	0	0	0	0	1	0	42
0	1	0	1	1	0	1	0	5A
0	1	0	1	1	0	1	0	5A
0	1	1	1	1	1	1	0	7E

$1A2642FF425A5A7E_{16}$

Задача

Закодируйте рисунок с помощью шестнадцатеричного кода:



Разрешение

Разрешение – это количество пикселей, приходящихся на дюйм размера изображения.

ppi = *pixels per inch*, пикселей на дюйм

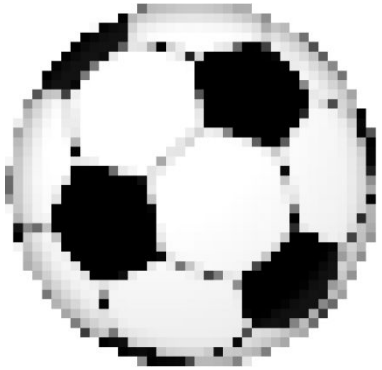
1 дюйм = 2,54 см

300 ppi

печать

96 ppi

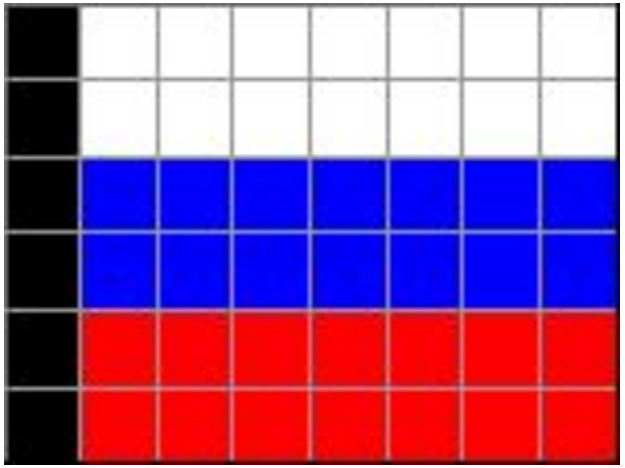
экран



48 ppi

24 ppi

Кодирование цвета



00	11	11	11	11	11	11	11
00	11	11	11	11	11	11	11
00	01	01	01	01	01	01	01
00	01	01	01	01	01	01	01
00	10	10	10	10	10	10	10
00	10	10	10	10	10	10	10

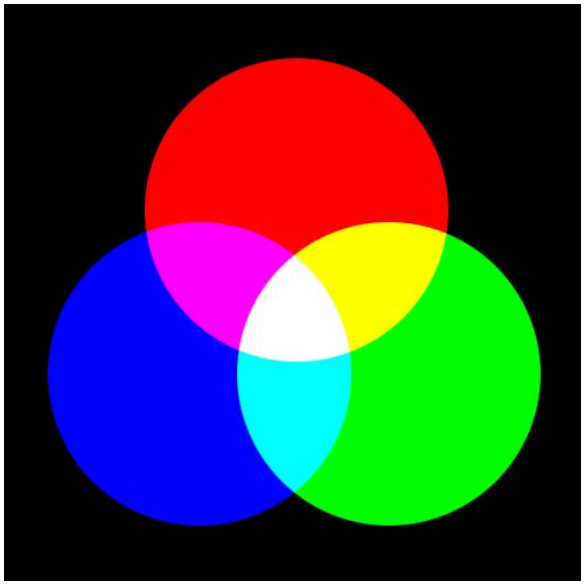
? Как выводить на монитор цвет с кодом 00?

? Как закодировать цвет в виде чисел?

Цветовая модель RGB

Д. Максвелл, 1860 **цвет = (R, G, B)**

red	***green***	***blue***
красный	зеленый	синий
0..255	0..255	0..255



(0, 0, 0)
(255, 255, 255)
(255, 0, 0)
(255, 150, 150)
(0, 255, 0)
(255, 255, 0)
(0, 0, 255)
(100, 0, 0)

? Сколько разных цветов можно кодировать?

256·256·256 = **16 777 216** (*True Color,* «истинный цвет»)

! RGB – цветовая модель для устройств, излучающих свет (мониторов)!

Цветовая модель RGB

(**255**, **255**, **0**) → #**FFFF00**

	RGB	Веб-страница
	(0, 0, 0)	#000000
	(255,255,255)	#FFFFFF
	(255, 0, 0)	#FF0000
	(0, 255, 0)	#00FF00
	(0, 0, 255)	#0000FF
	(255, 255, 0)	#FFFF00
	(204,204,204)	#CCCCCC

Задачи

Постройте шестнадцатеричные коды:

RGB (100, 200, 200) →

RGB (30, 50, 200) →

RGB (60, 180, 20) →

RGB (220, 150, 30) →

Глубина цвета

Глубина цвета — это количество битов, используемое для кодирования цвета пикселя.

Сколько памяти нужно для хранения цвета 1 пикселя в режиме *True Color*?

R (0..255) 256 = 2^8 вариантов 8 битов = 1 байт

R G B: 24 бита = 3 байта

True Color (истинный цвет)

Кодирование с палитрой

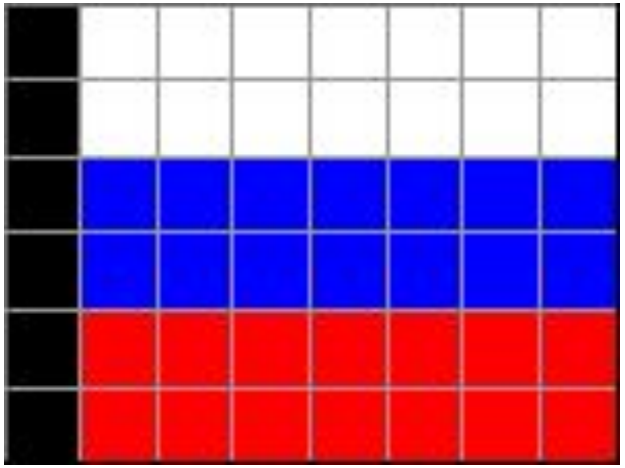
Как уменьшить размер файла?

- уменьшить разрешение
- уменьшить глубину цвета

снижается качество

Цветовая палитра – это таблица, в которой каждому цвету, заданному в виде составляющих в модели RGB, сопоставляется числовой код.

Кодирование с палитрой



00	11	11	11	11	11	11	11
00	11	11	11	11	11	11	11
00	01	01	01	01	01	01	01
00	01	01	01	01	01	01	01
00	10	10	10	10	10	10	10
00	10	10	10	10	10	10	10

Палитра:

0	0	0	0	0	255	255	0	0	255	255	255
цвет 00_2			цвет 01_2			цвет 10_2			цвет 11_2		

Какая глубина цвета?

2 бита на пиксель

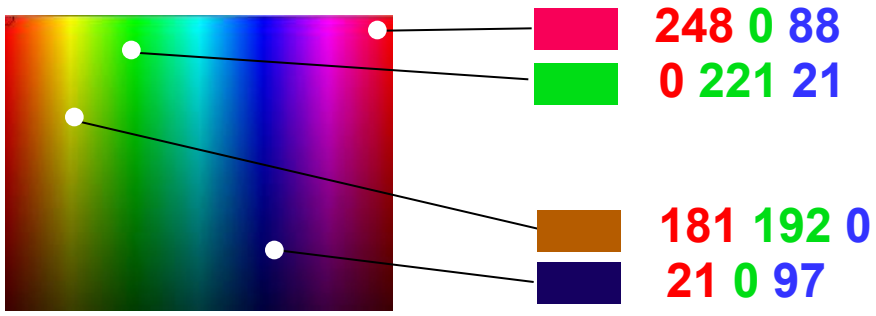
Сколько занимает палитра?

$3 \cdot 4 = 12$ байтов

Кодирование с палитрой

Шаг 1. Выбрать количество цветов: 2, 4, … **256.**

Шаг 2. Выбрать 256 цветов из палитры:



Шаг 3. Составить палитру (каждому цвету – номер 0..255)

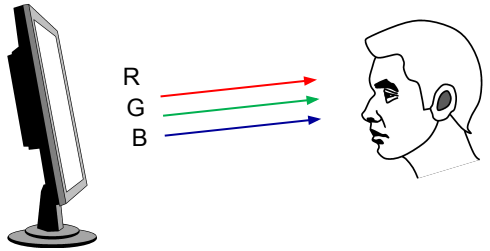
палитра хранится в начале файла

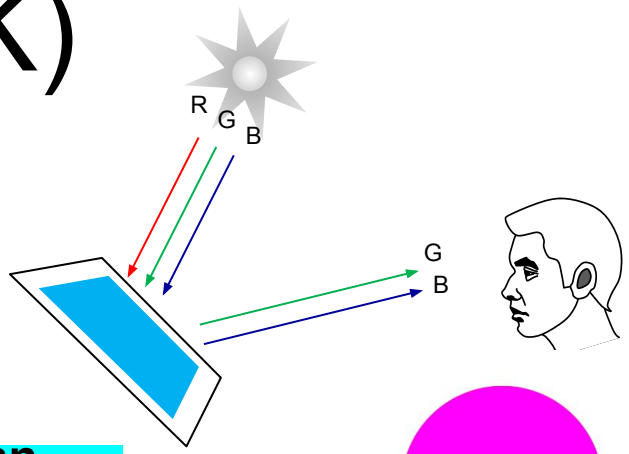
0	1		254	255
248 0 88	0 221 21	…	181 192 0	21 0 97

Шаг 4. Код пикселя = номеру его цвета в палитре

2	45	65	14	…	12	23

Кодирование цвета при печати (CMYK)

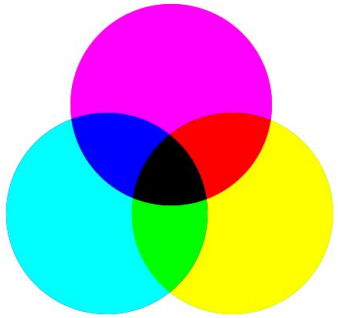




Белый – красный = голубой **C = Cyan**

Белый – зелёный = пурпурный **M = Magenta**

Белый – синий = желтый **Y = Yellow**



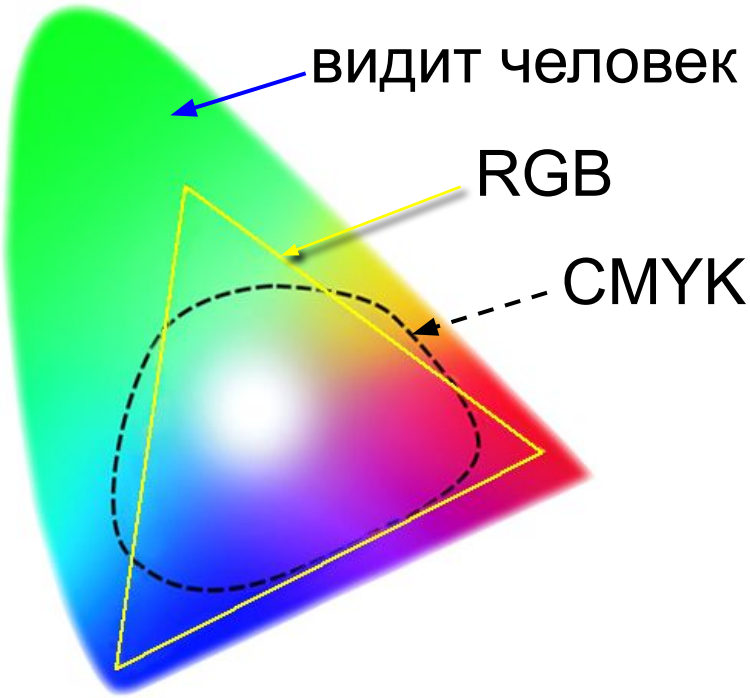
Модель CMY

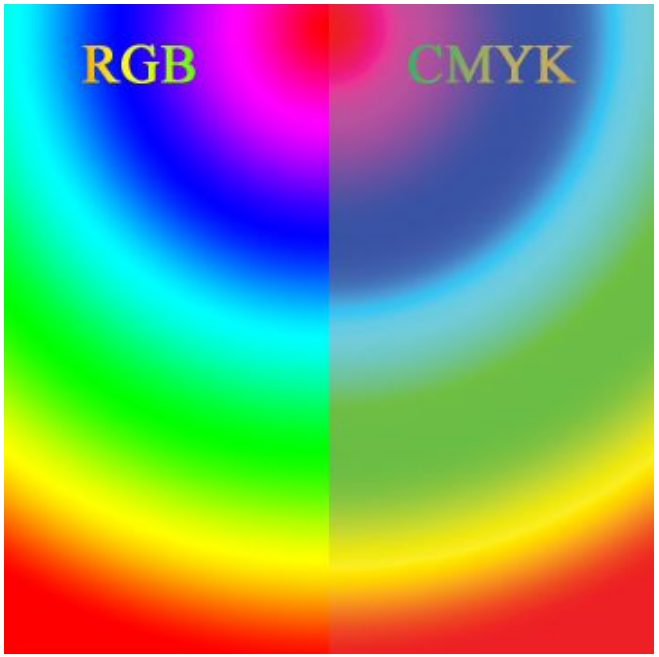
C	M	Y	
0	0	0	
255	255	0	
255	0	255	
0	255	255	
255	255	255	

Модель CMYK: + *Key color*

- меньший расход краски и лучшее качество для чёрного и серого цветов

RGB и CMYK





- не все цвета, которые показывает монитор (RGB), можно напечатать (CMYK)
- при переводе кода цвета из RGB в CMYK цвет искажается

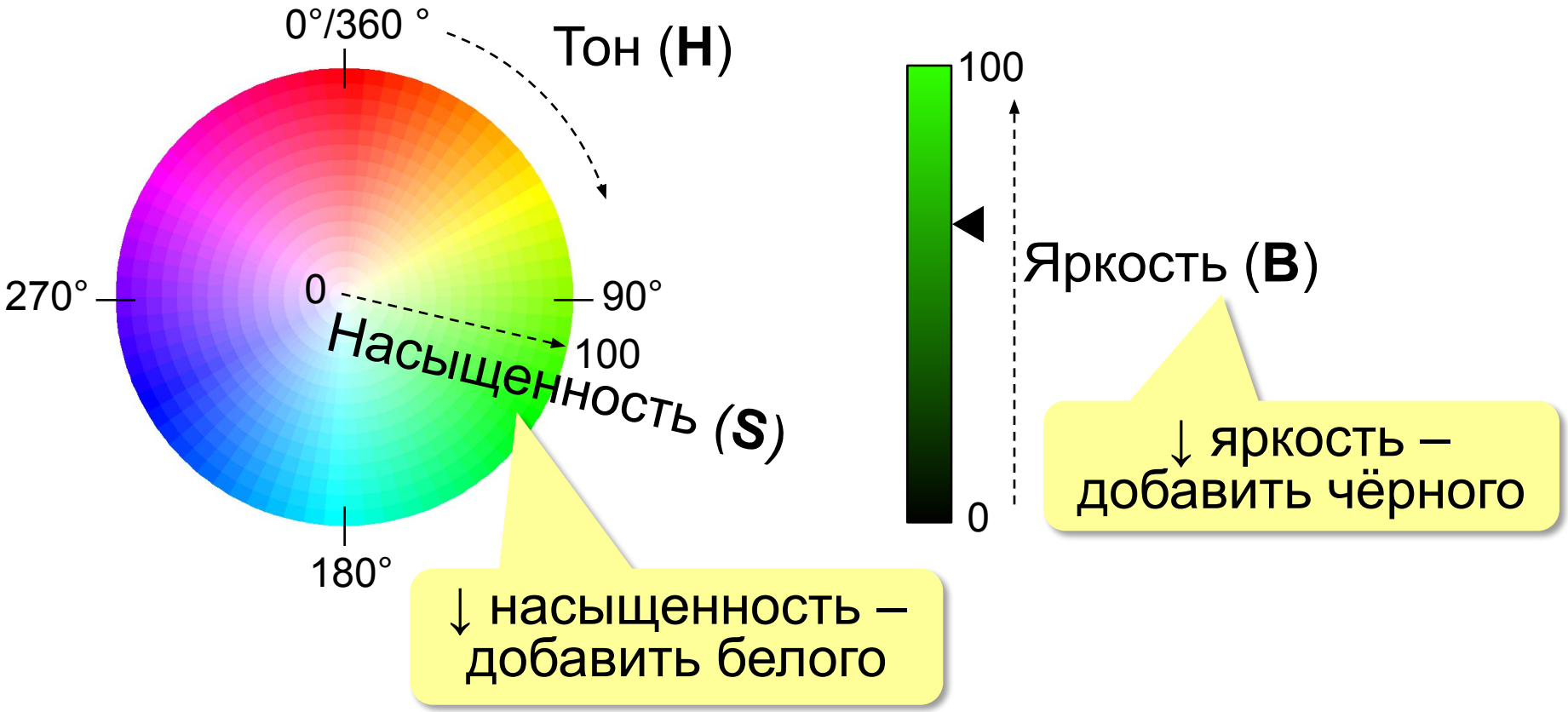
RGB(0,255,0)
CMYK(65,0,100,0)
→ RGB(104,175,35)

Цветовая модель HSB (HSV)

HSB = ***H****ue* (тон, оттенок)

S*aturation* (насыщенность)

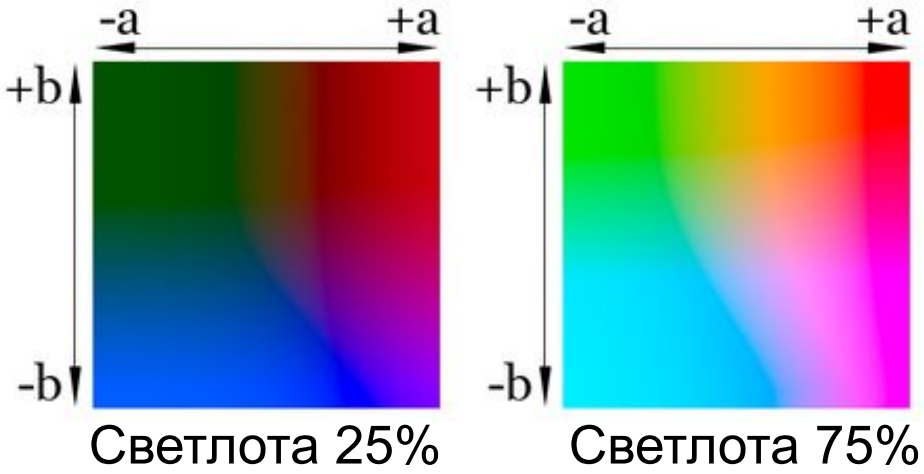
B*rightness* (яркость) или ***V****alue* (величина)



Цветовая модель Lab

Международный стандарт кодирования цвета, независимого от устройства (1976 г.)
Основана на модели восприятия цвета человеком.

Lab = *L*ightness (светлота)
a, b (задают цветовой тон)



Светлота 25% Светлота 75%

- для перевода между цветовыми моделями: RGB → Lab → CMYK
- для цветокоррекции фотографий

Растровое кодирование: итоги

- универсальный метод (можно закодировать любое изображение)
- единственный метод для кодирования и обработки размытых изображений, не имеющих чётких границ (фотографий)

- есть **потеря информации** (почему?)
- при изменении размеров цвет и форма объектов на рисунке **искажается**
- **размер файла** не зависит от сложности рисунка (а от чего зависит?)

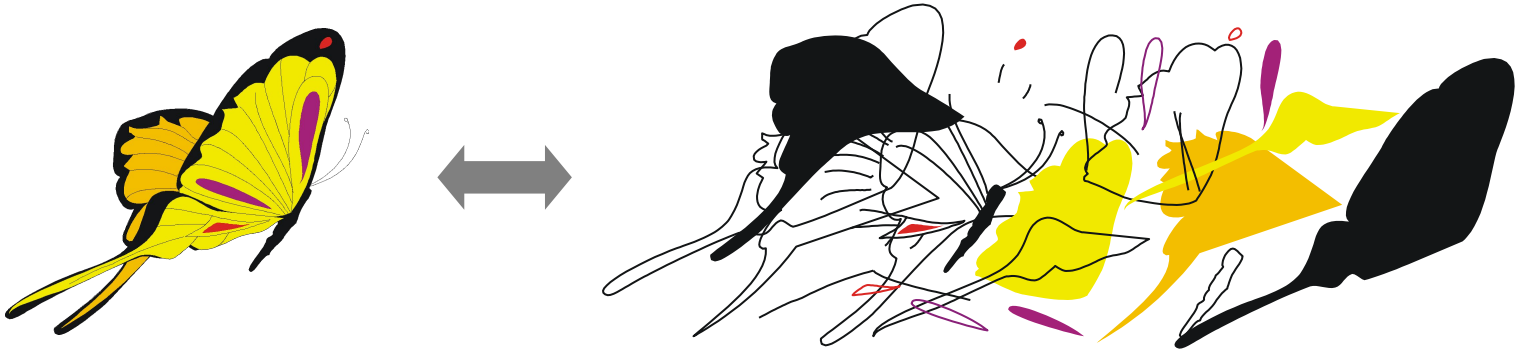
Векторное кодирование

Рисунки из геометрических фигур:

- отрезки, ломаные, прямоугольники
- окружности, эллипсы, дуги
- сглаженные линии (кривые Безье)

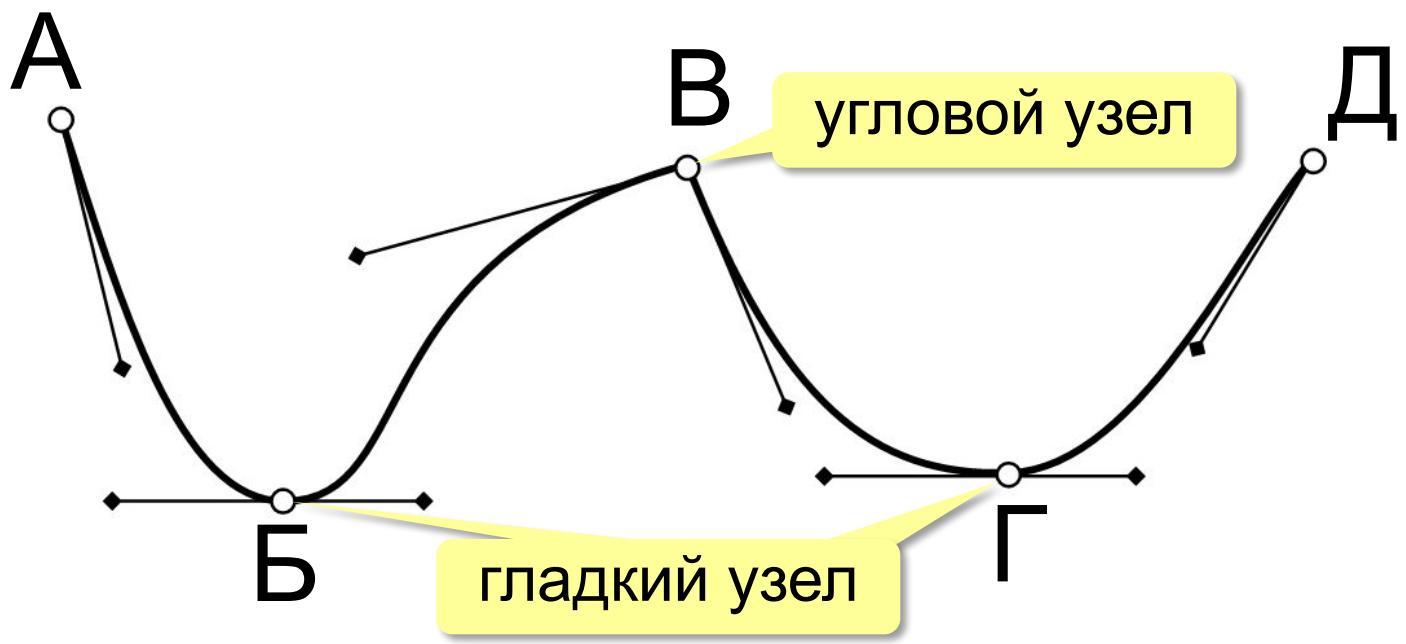
Для каждой фигуры в памяти хранятся:

- размеры и координаты на рисунке
- цвет и стиль границы
- цвет и стиль заливки (для замкнутых фигур)



Векторное кодирование

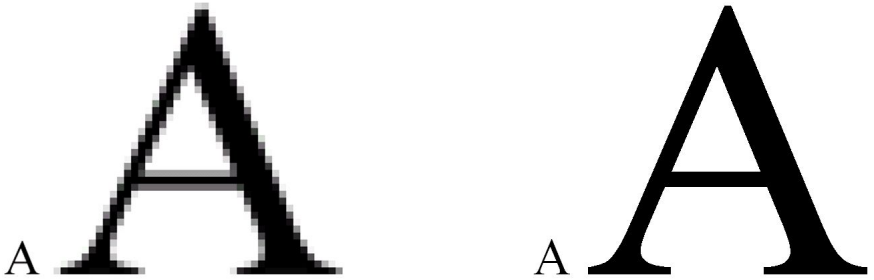
Кривые Безье:



Хранятся координаты узлов и концов «рычагов» (3 точки для каждого узла, кривые 3-го порядка).

Векторное кодирование (итоги)

- лучший способ для хранения **чертежей, схем, карт**
- при кодировании **нет потери информации**
- при изменении размера **нет искажений**



растровый рисунок

векторный рисунок

- меньше **размер файла**, зависит от сложности рисунка

- неэффективно использовать для **фотографий** и размытых изображений

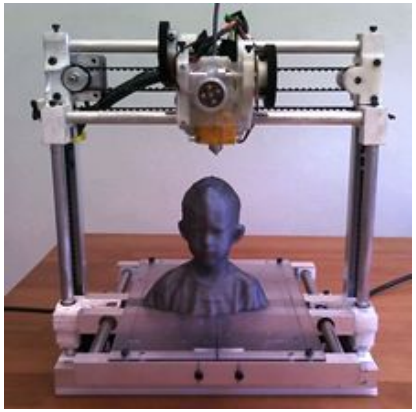
3D-графика

Трёхмерная графика (3D-графика) – это раздел компьютерной графики, который занимается созданием моделей и изображений *трёхмерных* объектов.

3D-модели: каждая точка имеет 3 координаты

3D-модели

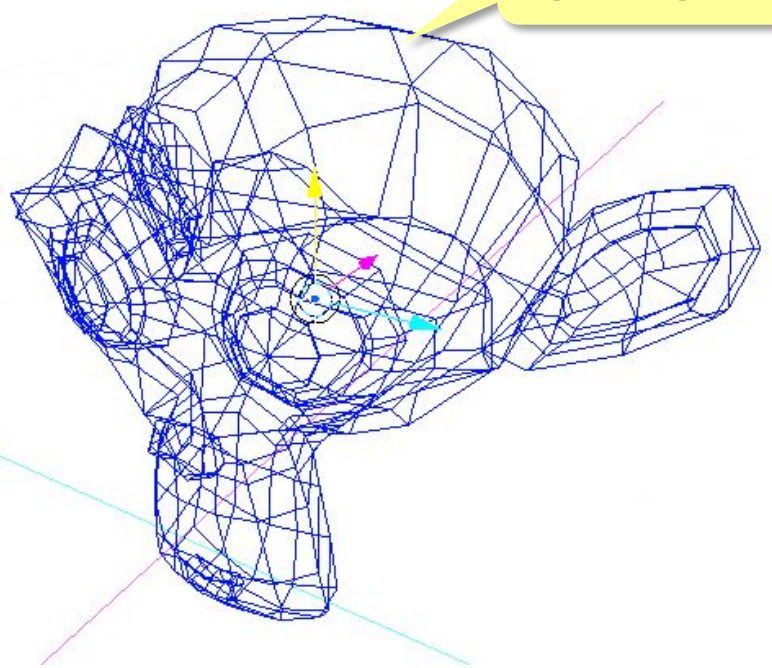
- проекции (на плоскость)
- анимация
- расчёты (на прочность и т.п.)
- 3D-печать



Построение каркаса (рёбер)

узлы (вершины)

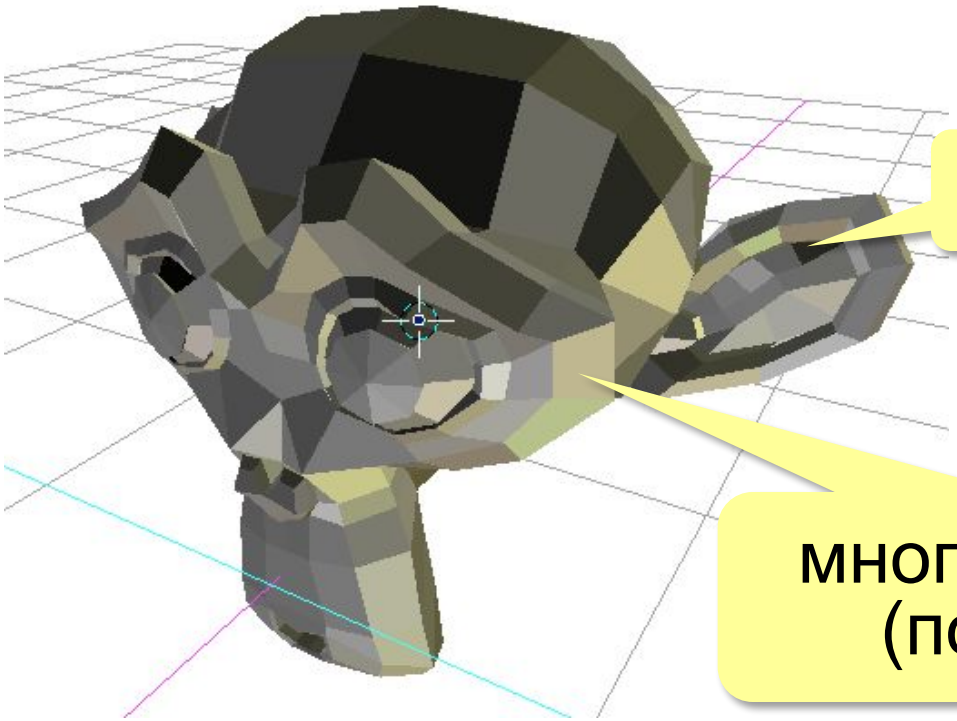
рёбра



Хранятся координаты точек (x, y, z)!

Растровая или векторная?

Поверхность

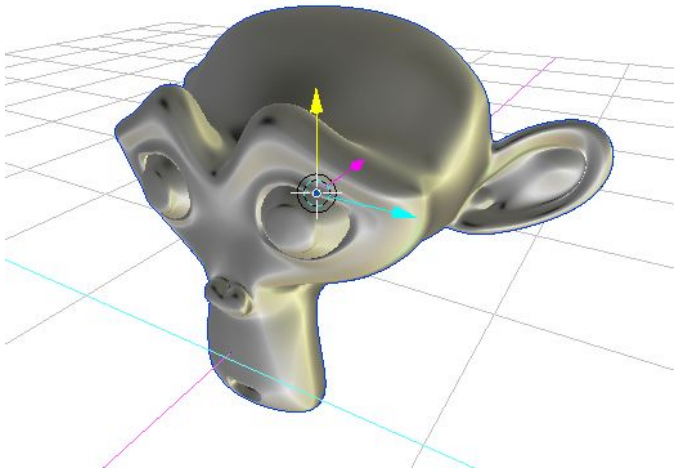


треугольники

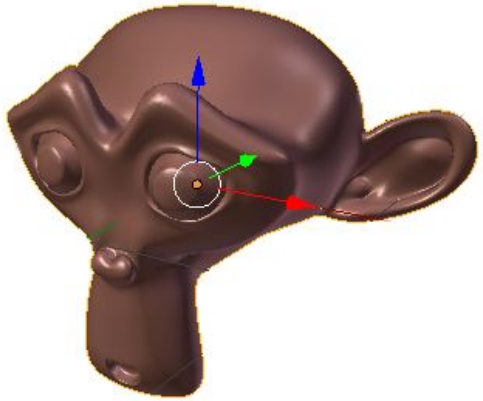
многоугольники
(полигоны)

Завершение модели

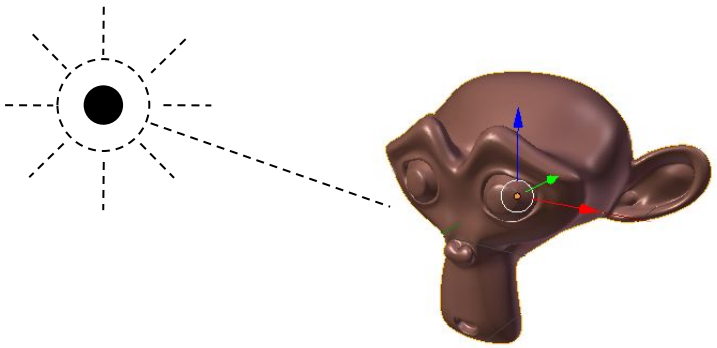
сглаживание



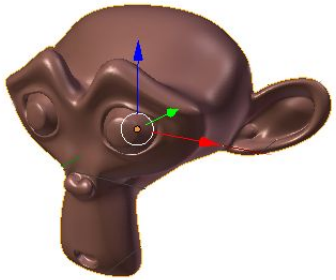
материал



установка света

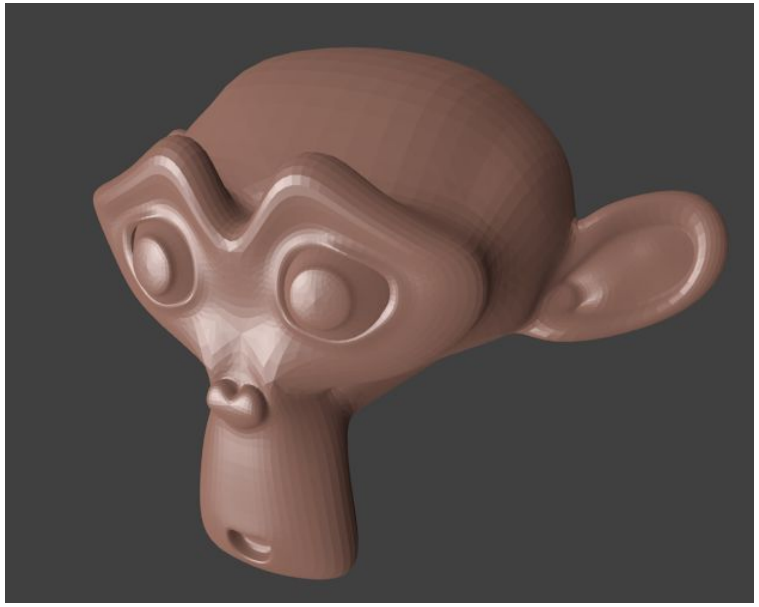


установка камеры



Результат

рендеринг

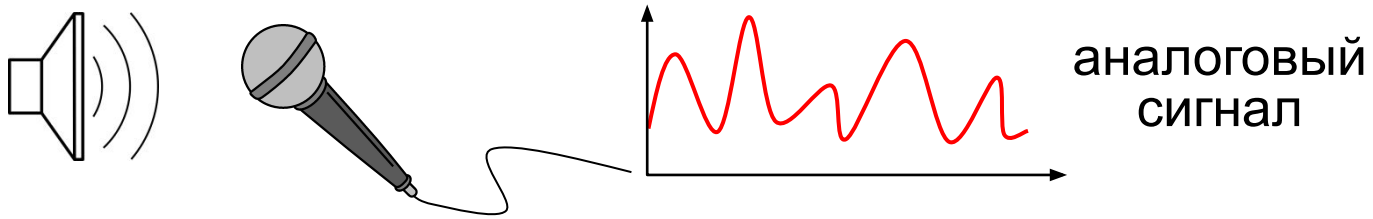


3D-печать

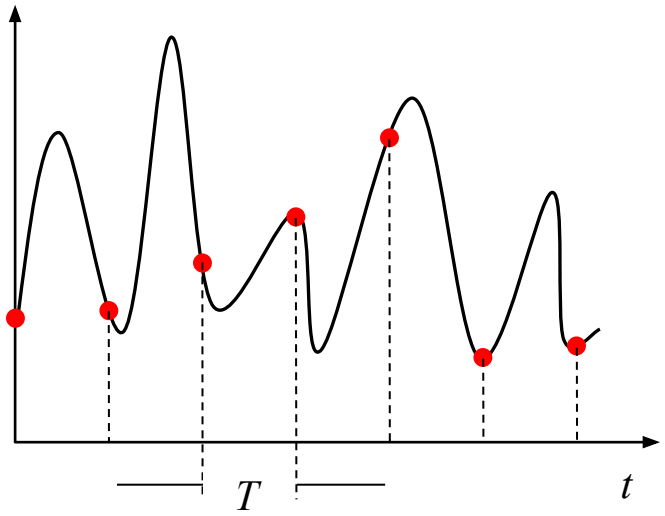
Рендеринг (визуализация) — построение двухмерного изображения по 3D-модели.

Кодирование звуковой и видеоинформации

Оцифровка звука



Оцифровка – это преобразование аналогового сигнала в цифровой код (дискретизация).



T – интервал дискретизации (с)

$f = \frac{1}{T}$ – частота дискретизации (Гц, кГц)

8 кГц – минимальная частота для распознавания речи

11 кГц, 22 кГц,

44,1 кГц – качество CD-дисков

48 кГц – фильмы на DVD

96 кГц, 192 кГц

Человек слышит

16 Гц … 20 кГц

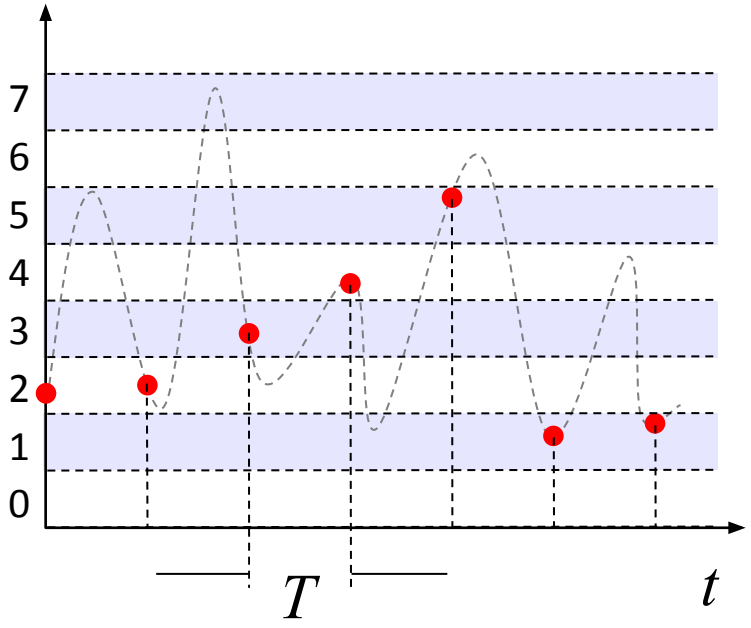
Оцифровка звука: квантование

Сколько битов нужно, чтобы записать число 0,6?

Квантование (дискретизация по уровню) – это представление числа в виде цифрового кода конечной длины.

АЦП = **А**налого-**Ц**ифровой **П**реобразователь

3-битное кодирование:



8 битов = 256 уровней

16 битов = 65536 уровней

24 бита = 2^{24} уровней

Разрядность кодирования — это число битов, используемое для хранения одного отсчёта.

Оцифровка звука

Задача. Определите информационный объем данных, полученных при оцифровке звука длительностью 1 минута с частотой 44 кГц с помощью 16-битной звуковой карты. Запись выполнена в режиме «стерео».

За 1 сек *каждый канал* записывает 44000 значений,
каждое занимает 16 битов = 2 байта
всего 44000 · 2 байта = 88000 байтов

С учётом «стерео»
всего 88000 · 2 = 176000 байтов

За 1 минуту
176000 · 60 = 10560000 байтов
≈ 10313 Кбайт ≈ 10 Мбайт

Оцифровка – итог

можно закодировать **любой звук** (в т.ч. голос, свист, шорох, …)

- есть **потеря информации**
- большой **объем файлов**

Какие свойства оцифрованного звука определяют качество звучания?

Форматы файлов:

WAV (*Waveform audio format),* часто без сжатия (размер!)

MP3 (*MPEG-1 Audio Layer 3*, сжатие с учётом восприятия человеком)

AAC (*Advanced Audio Coding*, 48 каналов, сжатие)

WMA (*Windows Media Audio,* потоковый звук, сжатие)

OGG (*Ogg Vorbis,* открытый формат, сжатие)

Инструментальное кодирование

MIDI (*Musical Instrument Digital Interface* — цифровой интерфейс музыкальных инструментов).

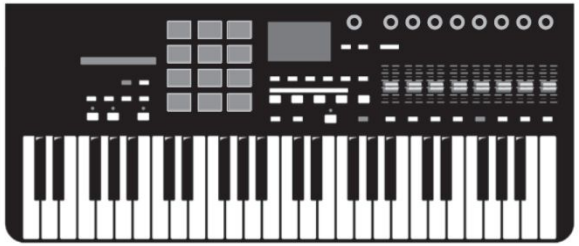
в файле **`.mid`**:
- нота (высота, длительность)
- музыкальный инструмент
- параметры звука (громкость, тембр)
- до 1024 каналов

128 мелодических и 47 ударных

программа для звуковой карты!

в памяти звуковой карты:
- образцы звуков (волновые таблицы)

MIDI-клавиатура:



- **нет потери информации** при кодировании инструментальной музыки
- небольшой **размер файлов**

невозможно закодировать нестандартный звук, голос