

Дисциплина

«Арифметические и логические основы вычислительной техники»

Лектор: Калиниченко Евгений Иванович, а.7а-321
Л/р: Калиниченко Е.И. и Никишин Кирилл Игоревич

Аттестация по дисциплине – экзамен.

Допуск на экзамен – выполнение **всех 14 л.р.**

Промежуточная аттестация (допуск на экзамен):

минимум – 36 баллов;

максимум – 60 баллов.

1 контр. точка – л.р. №1-№6, мин.-18б, макс -30б.

1 контр. точка – л.р. №7-№14, мин.-18б, макс -30б.

Для каждой л.р. есть:

- методические указания (файл - docx);
- дополнительный (теоретический) материал (файл - pptx);
- пример выполнения и пример отчета по л.р. (файл - jpeg)

Задания для л.р. с №1 по №6 индивидуальны **для каждого студента**. Вариант задания привязан к порядковому номеру студента в журнале группы.

Задания для л.р. с №7 по №14 индивидуальны **для каждой бригады**. Вариант задания привязан к номеру бригады.

Любая цифровая вычислительная система обязательно включает в себя, по крайней мере, один цифровой процессор, который является её «мозгом».

Поэтому, прежде чем изучать сложные вычислительные системы, нужно хорошо представлять себе работу цифрового процессора изнутри, а именно знать:

- арифметические основы цифровых процессоров;
- логические основы цифровых процессоров.

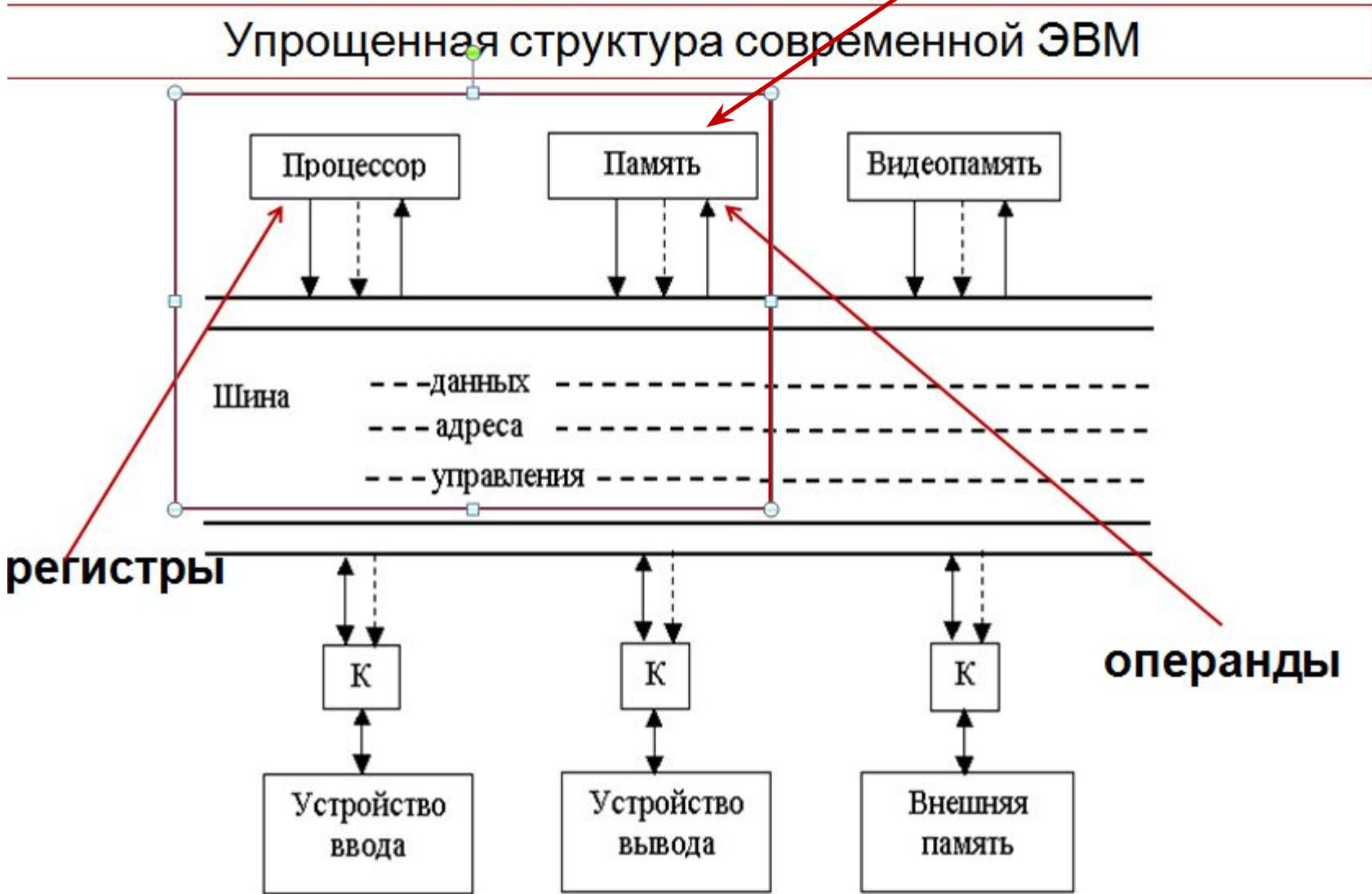
Арифметические основы цифровых процессоров

Содержание лекции 1:

- Системы счисления, используемые при работе с цифровыми процессорами и способы перевода
- Форматы представления чисел в цифровых процессорах
- Кодирование чисел в цифровых процессорах
- Кодирование символьной информации
- Пример цифрового процессора (i8086), обзор системы его команд

Системы счисления, используемые при работе с цифровыми процессорами и способы перевода

Информация, которую обрабатывает цифровой процессор хранится в оперативной памяти и в его регистрах. Эта информация **кодируется**.



Большинство кодов, применяемых в цифровых процессорах, основано на системах счисления, использующих позиционный принцип образования числа, при котором значение каждой цифры зависит от ее положения в числе.

Например, для десятичного числа 345 его значение рассчитывается по формуле:

$$345 = 3 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0$$

 ↑ ↑ ↑
 сотни десятки единицы

При работе с цифровыми процессорами используются четыре позиционные системы счисления.

- ▣ двоичная – представление информации (в памяти, в регистрах процессора) и её обработка в процессоре;
- ▣ восьмеричная, шестнадцатеричная (как промежуточные между процессором и человеком);
- ▣ десятичная (используемая человеком).

Каждая позиционная система счисления имеет базу.

База системы счисления это количество символов, используемых в ней.

В двоичной – база равна 2 (0,1)

В восьмеричной -- база равна 8 (0,1,2,3,4,5,6,7)

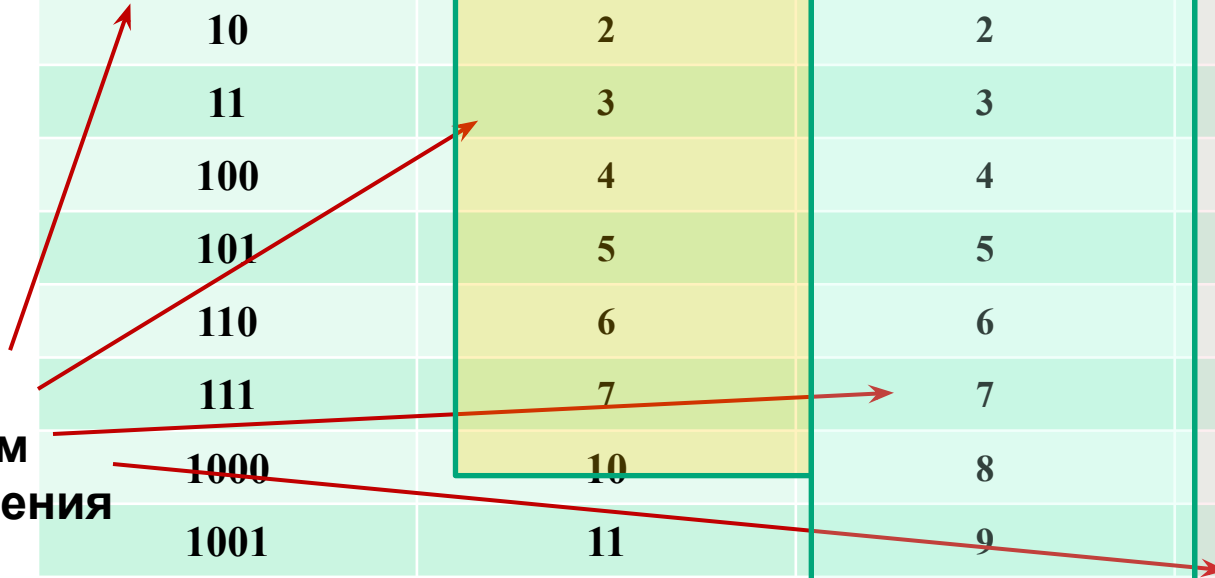
В шестнадцатеричной -- база равна 16
(0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)

В десятичной -- база равна 10 (0,1,2,3,4,5,6,7,8,9)

Системы счисления

Двоичная	Восьмеричная	Шестнадцатеричная	Десятичная
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

Базы
систем
счисления



В IT-литературе используются следующие обозначения, чтобы указать систему счисления, в которой представлены числа:

• для двоичной системы:

- использование нижнего индекса справа от числа.

Например: $101000_{(2)}$

- использование буквы В (либо b) (binary – двоичный) как нижнего индекса справа от числа или в одну строку с числом. Например:

$$101000_b = 101000_B = 101000В = 101000b;$$

• для восьмеричной системы:

- нижний индекс справа от числа в виде цифры 8 в скобках или букв O, o (octal – восьмеричный), как нижнего индекса или в одну строку справа от числа. Например:

$$367_{(8)} = 367_{\text{O}} = 367_{\text{o}} = 367\text{O} = 367\text{o};$$

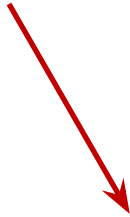
• для шестнадцатеричной системы:

- нижний индекс справа от числа в виде числа 16 в скобках или букв H, h (hexadecimal – шестнадцатеричный), как нижнего индекса или в одну строку справа от числа.

Например:

$$3AB_{(16)} = 3AB_H = 3AB_h = 3ABH = 3ABh.$$

Внимание: В отладчике (Turbodebugger), если 16-ричное число начинается с буквы, то запись его начинается с нуля:


$$AE1_{(16)} = \underline{0}AE1h$$

Перевод чисел

Рассмотрим алгоритмы перевода чисел из одной системы счисления в другую.

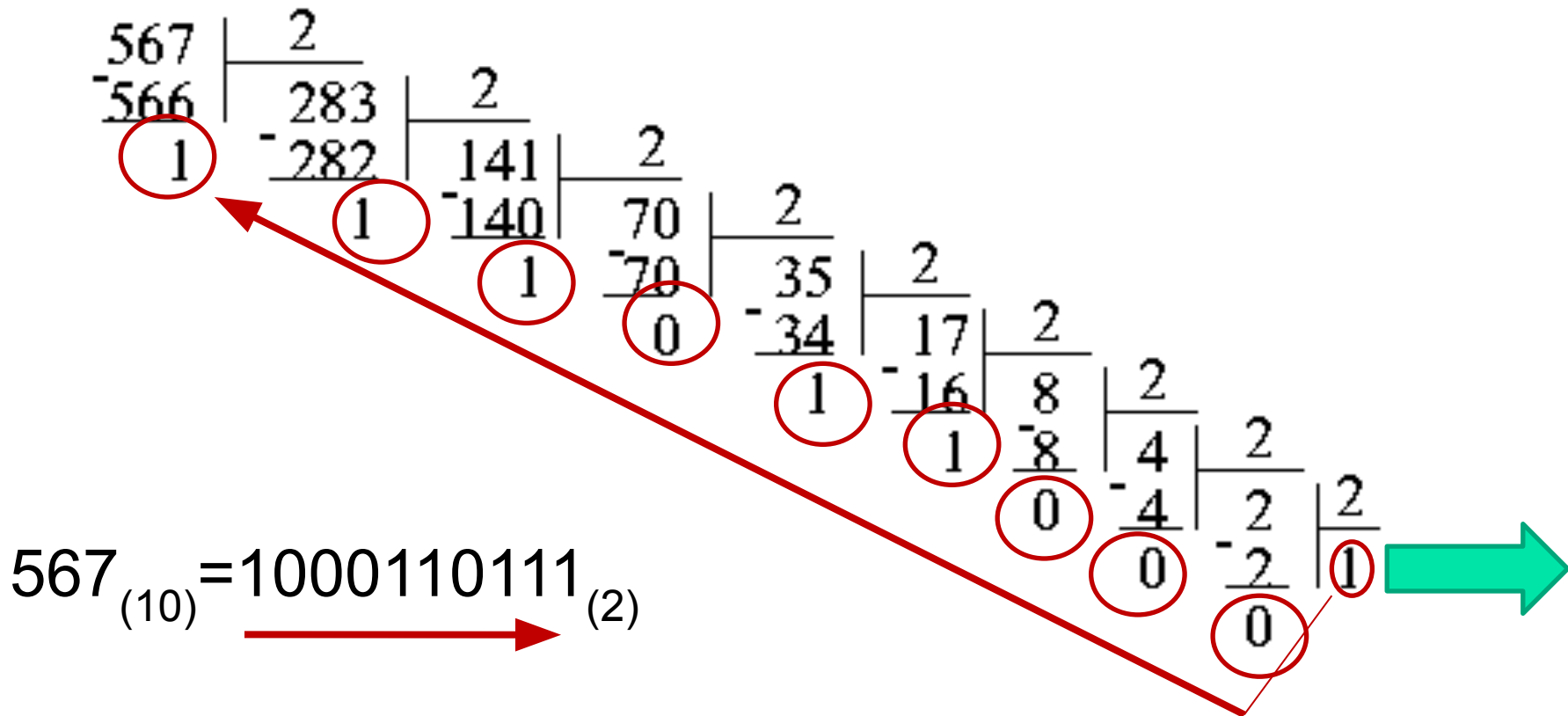
Эти алгоритмы различаются в зависимости от переводимого числа – целое или правильная дробь.

Целые числа переводятся с использованием операции деления, а правильные дроби – умножения.

Вещественные числа, имеющие целую и дробную часть переводятся по алгоритмам отдельно для целой части и дробной части.

Алгоритм перевода целых чисел

Перевести число 567 из десятичной в двоичную систему.



Внимание: не переводить по степеням двойки в лабораторных работах!

Почему используются 8-чная и 16-чная системы?

При переводе из десятичной (или же в десятичную) системы счисления в систему с основанием 8 или 16 результат получается быстрее, чем при переводе в двоичную систему. Это происходит за счет сокращения числа операций деления (умножения). И запись чисел в этих системах компактнее, чем в двоичной

А из этих систем очень просто осуществляется перевод чисел в двоичную систему (или из двоичной системы в 8, 16).

Это потому что основание этих систем, **кратно степеням двойки** ($8=2^3$ и $16=2^4$).

Перевод из 8,16-чной в 2-чную

Для того чтобы целое двоичное число перевести из системы счисления с основанием 2^n в двоичную систему, нужно каждую цифру числа в исходной системе заменить

n-разрядным

двоичным эквивалентом по ранее показанной таблице.

Например:

$$3_{(8)} = 011_{(2)} \quad (\text{двоичная триада})$$

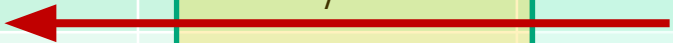
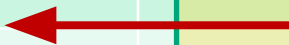
$$7_{(16)} = 0111_{(2)} \quad (\text{двоичная тетрада})$$

Системы счисления

Двоичная	Восьмеричная	Шестнадцатеричная	Десятичная
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

$$3_{(8)} = 011_{(2)}$$

$$7_{(16)} = 0111_{(2)}$$



Перевести $4A3F_{(16)}$ в двоичную систему счисления:

$4_{(16)} = 0100_{(2)}$ – тетрада;

$A_{(16)} = 1010_{(2)}$ – тетрада;

$3_{(16)} = 0011_{(2)}$ – тетрада;

$F_{(16)} = 1111_{(2)}$ – тетрада.

Тогда, располагая каждую тетраду с учетом ее веса, получаем в двоичной системе:

$4A3F_{(16)} = 0100101000111111_{(2)} = 0100101000111111_{(2)}$.

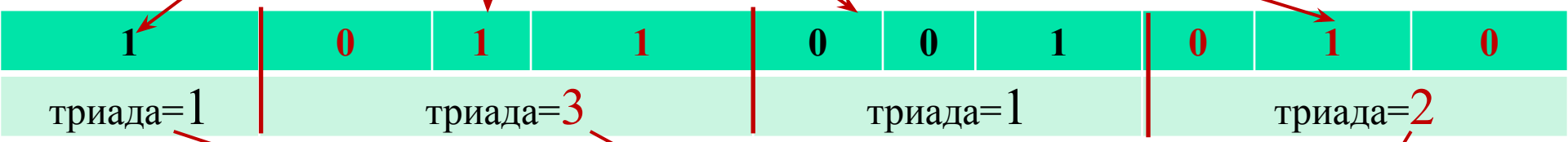
незначащий 0, можно не писать!

Перевод из 2-чной в 8, 16-чную:

для того чтобы целое двоичное число перевести в систему счисления с основанием 2^n , нужно:

- 1) двоичное число разбить справа налево на группы по n цифр в каждой; если в последней левой группе окажется меньше n разрядов, то дополнить ее нулями слева до нужного числа разрядов;
- 2) рассмотреть каждую группу как n -разрядное двоичное число и заменить ее соответствующей цифрой в системе счисления с основанием 2^n .

Перевести $1011001010_{(2)}$ в восьмеричную систему счисления:



Тогда, располагая восьмеричные цифры по весам разрядов, получаем: $1011001010_{(2)} = 1312_{(8)}$

Рациональный алгоритм перевода чисел такой (обязателен при выполнении лабораторных работ):

от человека к процессору --- 10 -> 16 -> 2

от процессора к человеку --- 2 -> 16 -> 10.

Пример: перевести $567_{(10)}$ в двоичную систему

1) переводим в 16-чную систему

567	16	
560	35	16
7	32	2
		3

$567_{(10)} = 237_{(16)}$

$567_{(10)} = 1000110111_{(2)}$

2) переводим из 16-чной в 2-чную

$001000110111_{(2)} = 1000110111_{(2)}$

Алгоритм перевода правильной дроби

Правильная дробь имеет целую часть, равную 0.

Перевод дроби в другую систему счисления выполняется по следующему алгоритму:

- 1) дробь умножается на основание системы счисления, в которую переводится (например, на 2, 8 или 16);
- 2) в первом полученном произведении выделяется целая часть и записывается цифрой системы счисления, в которую осуществляется перевод (на первом шаге она является старшей цифрой получаемой дроби в новой системе счисления);

3) полученная целая часть произведения отбрасывается, и оставшаяся дробная часть (это правильная дробь) вновь умножается на основание системы счисления, в которую осуществляется перевод, с последующей обработкой полученного произведения в соответствии с шагами 1) и 2);

Внимание! Если целые числа из одной системы счисления в другую всегда переводятся точно, то дробные числа, как правило приближенно.

Поэтому при переводе дробных чисел задается точность перевода (количество точных разрядов, которые нужно получить).

Если надо получить три точных разряда, то при переводе получают четыре разряда и последний четвертый разряд используется для округления.

Будем использовать правило округления: если отбрасываемый разряд равен или больше половины от основания системы счисления, в которую осуществляется перевод, то к остающемуся младшему разряду добавляется "1", в противном случае оставшаяся цифра не меняется.

$$0,3454_{(8)} \approx 0,346_{(8)} \quad (+1)$$

$$0,5632_{(8)} \approx 0,563_{(8)} \quad (+0)$$

Пример перевода правильной дроби:

Выполнить перевод числа $0,847_{(10)}$ в двоичную систему счисления. Перевод выполнить с точностью восемь значащих цифр после запятой.

~~Эффективным путем достижения цели будет перевод числа вначале в шестнадцатеричную (или восьмеричную систему), а затем – в двоичную.~~

Тогда, чтобы получить требуемую точность (8 двоичных разрядов) нужно получить три 16-ричных разряда.



$0,847_{(10)} \approx 0,D9_{(16)} = 0,11011001_{(2)}$. Отметим, что третья шестнадцатеричная цифра после запятой использована для округления предыдущего разряда (второй после запятой) по правилу «половина веса отбрасываемого разряда».

Перевод дроби из 2-чной системы в 10-чную:

Выполнить перевод правильной дроби $0,001111011010001_{(2)}$ в десятичную систему счисления. Перевод выполнить с точностью три значащие цифры после запятой.

Эффективным путем достижения цели будет перевод числа вначале в шестнадцатеричную (или восьмеричную систему), а затем – в десятичную.

Внимание: разбиение двоичного числа на тетрады или триады для правильной дроби выполняется **слева направо**.

$$0,001111011010001 = 0,0011110110100010_{(2)} \approx 0,3DA_{(16)} = 3 \cdot 16^{-1} + 13 \cdot 16^{-2} + 10 \cdot 16^{-3} = 0,241_{(10)}$$

Алгоритм перевода вещественных чисел

Вещественные числа имеют целую часть и конечную или бесконечную (периодическую или непериодическую) дробную часть.

Например $1\frac{1}{3} = 1,33(3)$.

Такие числа переводятся в другую систему счисления следующим образом:

целая часть числа – по алгоритму перевода целых чисел;

дробная часть по алгоритму перевода дробных чисел.

Пример не приводится, поскольку все очевидно из ранее рассмотренного.

Форматы

**представления чисел в цифровых
процессорах**

Цифровой процессор – это устройство, осуществляющее обработку информации, представленной в цифровом виде (двоичная система счисления). Внешне это микросхема с высокой степенью интеграции (есть в любом компьютере, гаджете).

Одной из основных его характеристик является **«разрядность процессора»**.

Разрядность процессора – это **количество бит в регистре процессора**.

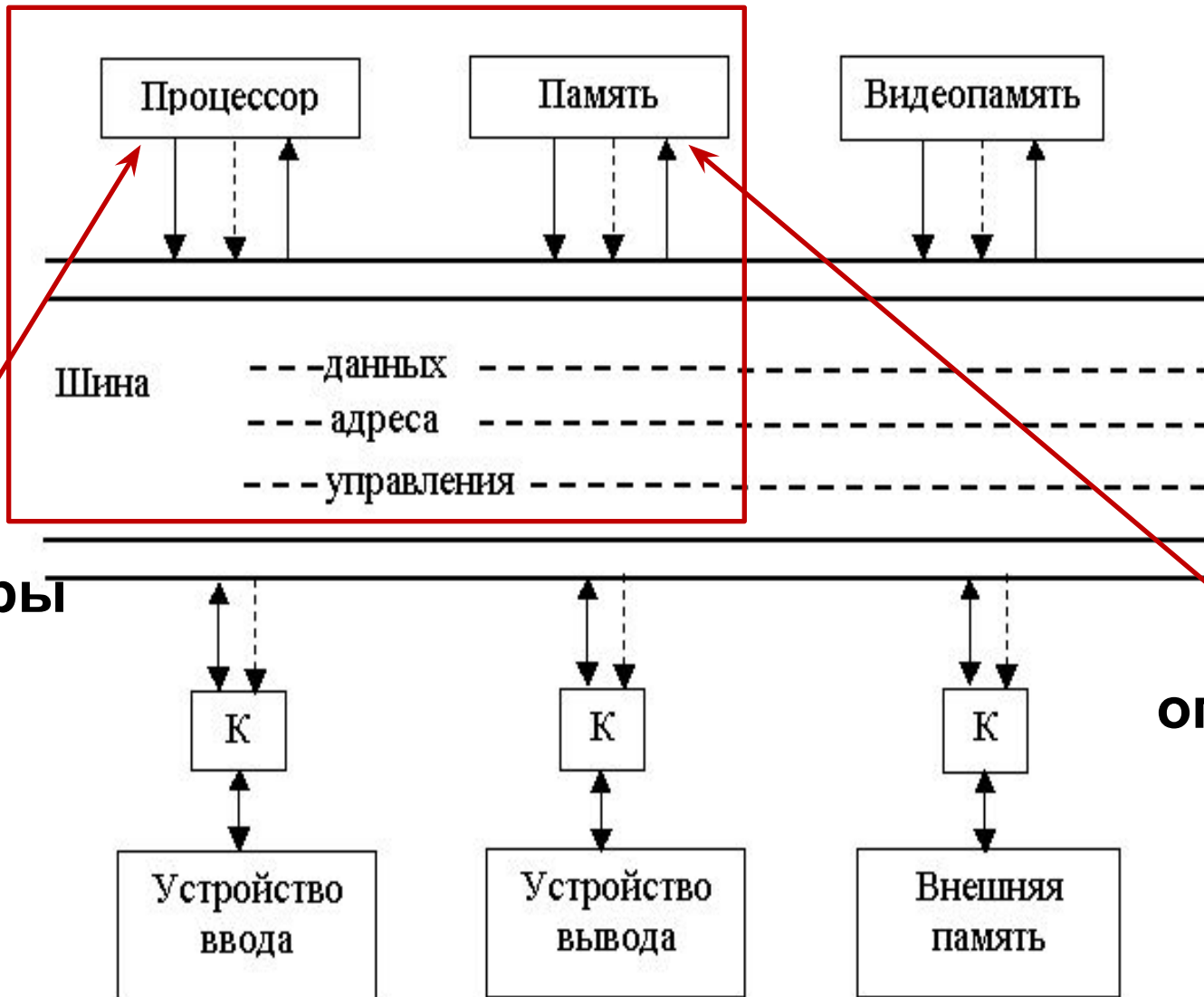
Регистры используются для хранения операндов, выполнения операций сложения, вычитания, сдвига или других команд **за один такт работы**.

Т.е. разрядность процессора определяет сколько бит используются для выполнения операций сложения, вычитания, сдвига или других команд за один такт работы.

При выполнении **арифметических операций** цифровой процессор использует представление **чисел** (называемых операндами) в определенных форматах.

В этих форматах операнды хранятся в оперативной памяти компьютера, затем пересылаются в регистры процессора и над ними выполняются арифметические операции.

Упрощенная структура современной ЭВМ



регистры

операнды

В цифровом процессоре для представления чисел используются три следующих формата :

- формат с фиксированной точкой;
- формат с плавающей точкой;
- формат двоично-десятичного кода.

Формат с фиксированной точкой (**sign**, **unsign**)

Формат с фиксированной точкой (ФТ), в зависимости от представляемых чисел, имеет три вида:

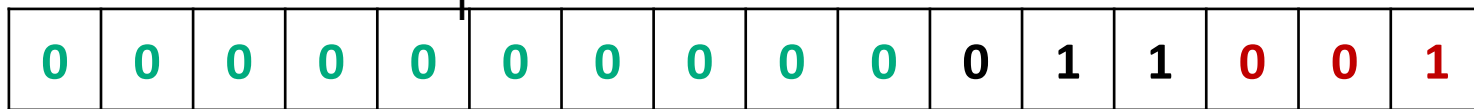
Вид ФТ	
Целые числа (integer)	без знака (unsign в языках программирования)
	со знаком (sign в языках программирования)
Дробные числа	без знака
	со знаком
Смешанные числа	без знака
	со знаком

Формат ФТ (sign) представления целых чисел

Целое число K_1 , заданное в восьмеричной системе счисления, представить в формате с ФТ в 16-разрядном процессоре (2 байта).

$$K_1 = 31_{(8)} = 11001_{(2)}$$

без знака K_1 :

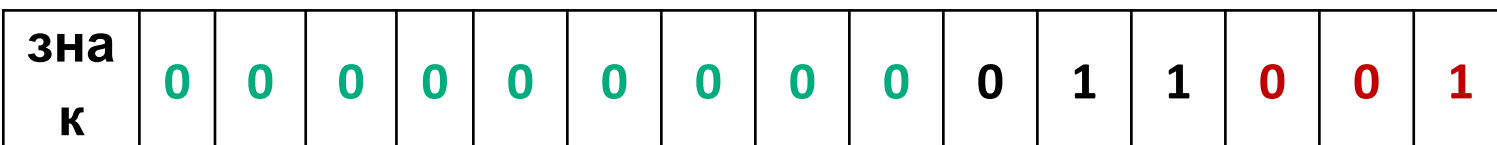


$$31_{(8)} = 19_{(16)}$$

заполнение разрядов



со знаком K_1 :



$$31_{(8)} = 19_{(16)}$$



Еще пример для целого числа:

Целое число K_2 , заданное в восьмеричной системе счисления, представить в формате с ФТ (2 байта).

$$K_2 = 1231_{(8)} = 1010011001_{(2)} = 299_{(16)}$$

без знака:

0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$= 1231_{(8)} = 299_{(16)}$

ds:0000 99 02

со знаком:

знак	0	0	0	0	0	1	0	1	0	0	1	1	0	0	1
------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$= 1231_{(8)} = 299_{(16)}$

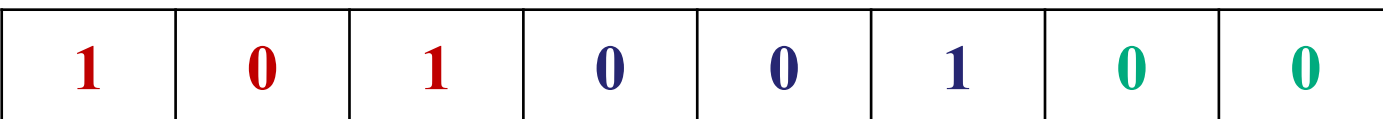
ds:0000 99 02

Формат представления правильных дробей

Дробное число M_1 , заданное в восьмеричной системе счисления, представить в формате с ФТ в 8-разрядном процессоре (1 байт).

$$M = 0,51_{(8)} = 0,101001_{(2)}$$

без знака:



$$= 0,51_{(8)} = 0,A4_{(16)}$$

заполнение разрядов



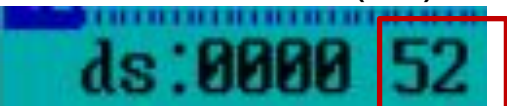
со знаком



$$= 0,51_{(8)} = 0,A4_{(16)}$$

$$5_{(16)}$$

$$2_{(16)}$$



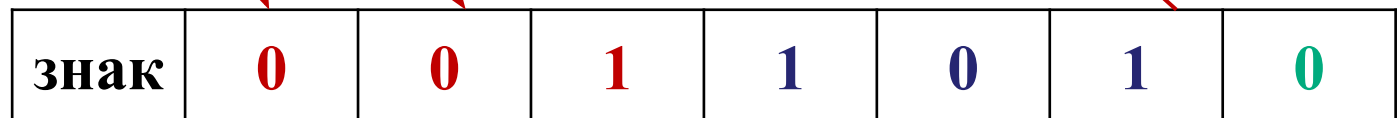
Дробное число M_2 , заданное в восьмеричной системе счисления, представить в формате с ФТ в 8-разрядном процессоре (1 байт).

$$M = 0,15_{(8)} = 0,001101_{(2)}$$

без знака:



со знаком:



ds:0000 34 1A

A screenshot of a debugger window showing a memory dump. The address ds:0000 contains the value 34 1A. Red boxes highlight the hex values 34 and 1A. A red arrow points from the 6th bit of the 'со знаком' row to the 34 hex value, and another red arrow points from the 7th bit of the 'со знаком' row to the 1A hex value.

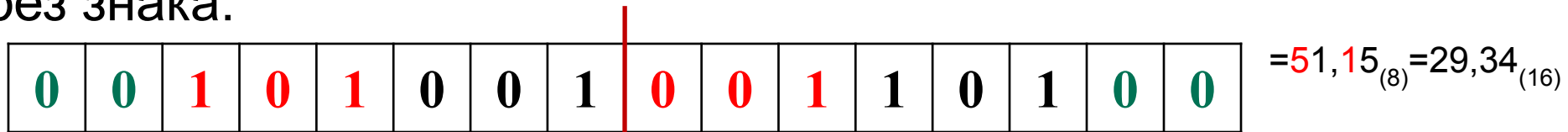
Смешанные числа (неправильные дроби)

Для представления таких чисел в формате с ФТ разрядная сетка процессора делится на две части: одна (левая) для целой части числа, другая (правая) для дробной части (специального разделителя нет).

Смешанное число L , заданное в восьмеричной системе счисления, представить в формате с ФТ в шестнадцатиразрядном процессоре, где левые 8 бит отведены под целую часть числа, включая знак, а остальные 8 – под дробную часть.

$$L = 51,15_{(8)} = 101001,001101_{(2)}$$

без знака:



ds:0000 34 29

со знаком:



Запятая для разделения целой и дробной части для смешанных чисел с ФТ в явном виде отсутствует, её положение учитывается в алгоритмах арифметических операций.

В формате ФТ (целые, дробные, смешанные) арифметические операции выполняются **ТОЧНО**.

Диапазон вычисления с ФТ определяется разрядностью процессора. Например в 16-разрядном процессоре диапазон вычисления для целых чисел со знаком от $-32768_{(10)}$ до $+32767_{(10)}$.

Это не значит, что на таком процессоре нельзя выполнить арифметические операции над числами с ФТ в большем диапазоне. Можно, но в этом случае нужно будет использовать так называемую "n-разрядную арифметику".

Формат с плавающей точкой (float)

Одной из форм записи вещественных чисел является их представление в экспоненциальном виде, в котором отдельно записывают **мантиссу** числа и **порядок** числа.

Пример экспоненциальной формы числа $2008_{(10)}$:

$$2008_{(10)} = 20,08 * 10^2 = 200800 * 10^{-2} = 0,2008 * 10^4$$

Мантисса

Порядок

Мантисса равна 20,08 или 200800, или 0,2008.
Порядок равен соответственно 2 или -2, или 4.

Любое число в экспоненциальной форме имеет множество представлений.

$$1 = 0,00001 * 10^5 = 1000 * 10^{-3} \text{ и т.д.}$$

Среди этих представлений выделили нормализованное представление числа:

$$2008_{(10)} = 0,2008 * 10^4$$

Для каждого числа это представление – единственное.

$$1 = 0,1 * 10^1$$

При нормализованном представлении числа в экспоненциальной форме мантисса (M) должна быть в интервале

$0,1_{(d)} \leq M < 1_{(d)}$, где d – основание системы счисления.

Операнды в цифровом процессоре в формате с плавающей точкой (ПТ/float) представляют числа в экспоненциальной форме.

Такой формат чисел в компьютерах используется для научно-технических расчетов, когда в вычислениях диапазон чисел может варьироваться от очень малых величин до очень больших, т.е. нужно обеспечить большой диапазон вычислений. Это плюс.

Но в отличие от формата с ФТ, в котором выполняются абсолютно точные вычисления, операции в таком формате выполняются **с приближением**, определяемым разрядной сеткой процессора.

В истории IT- технологий существовало много форматов чисел в формате с ПТ.

В настоящее время общепринятым стандартом представления операндов в формате с плавающей точкой в цифровом процессоре является стандарт **IEEE 754**.

В IEEE 754:

- мантисса представляется в прямом коде;
- порядок «смещен» (увеличен) на константу.

Смещение порядка на константу позволяет обойтись без явного бита знака порядка. Если значение смещенного порядка больше константы смещения, он – положительный, если меньше – отрицательный. Такое решение позволило реализовать более простые алгоритмы операций над порядками.

Если константа смещения равна 127, то:

$0,1 \cdot 10^{-3}$ будет записано, как $0,1 \cdot 10^{124}$;

$0,2008 \cdot 10^4$ будет записано, как $0,2008 \cdot 10^{131}$

Есть два формата представления чисел с плавающей точкой стандарта **IEEE 754** в оперативной памяти процессора :

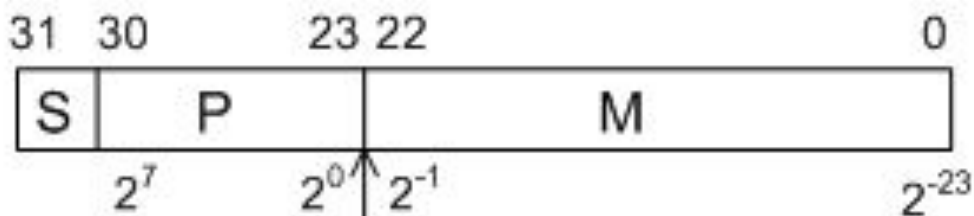
- «короткое вещественное (КВ)»,
- «длинное вещественное (ДВ)».

Они отличаются диапазоном представимых в них чисел.

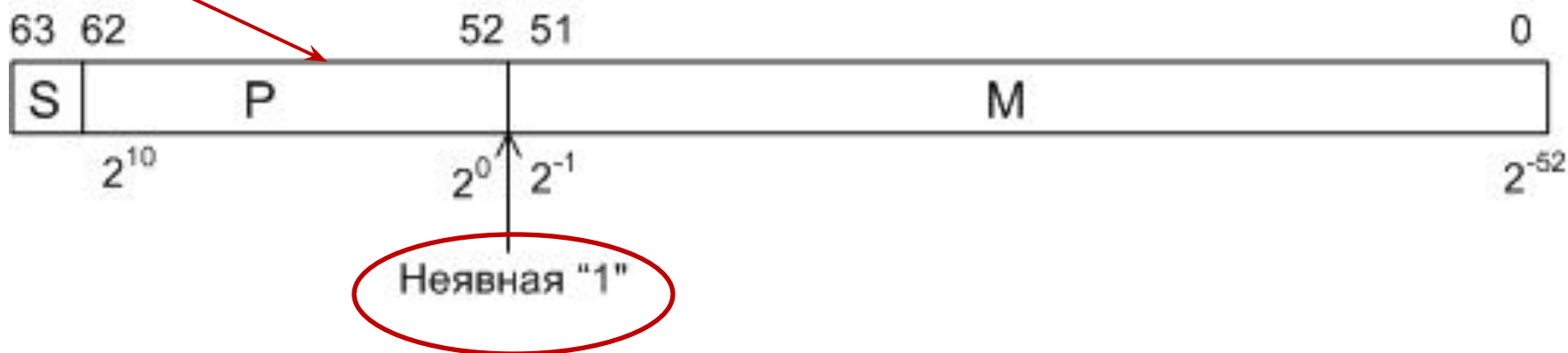
В самом же процессоре арифметические операции выполняются всегда в формате: «внутреннее (расширенное) вещественное (ВВ)».

Т.е. при передаче из оперативной памяти в процессор форматы КВ и ДВ преобразуются в ВВ и наоборот.

КВ (4 байта)



ДВ (8 байт)



Используются следующие обозначения: М – мантисса числа; S – знак мантиссы; Р – порядок числа.

В формате КВ под мантиссу отводится 24 бит, а под порядок – 8 бит. Величина порядка операнда смещена на $127_{(10)}$, т.е.

$$P = P_x + 127_{(10)}$$

В формате ДВ под мантиссу отводится 53 бит, а под порядок – 11 бит. Величина порядка операнда смещена на $1023_{(10)}$, т.е.

$$P = P_x + 1023_{(10)}$$

Форматы вещественных чисел в IEEE-754

КВ

ДВ

ВВ

	Короткий	Длинный	Расширенный
Длина числа, бит	32	64	80
Размерность мантиссы, бит	24	53	64
Диапазон представляемых значений	$10^{-38} .. 10^{+38}$	$10^{-308} .. 10^{+308}$	$10^{-4932} .. 10^{+4932}$
Размерность характеристики q, бит	8	11	16
Фиксированное смещение f	+127	+1023	+16383
Диапазон порядка p	-126 .. 127	-1022 .. +1023	-16382 .. +16383
Диапазон характеристики q	0 .. 255	0 .. 2047	0 .. 32767

здесь термин характеристика = порядок



«Скрытый» бит мантиссы

Поскольку при нормализованном представлении операнда $(0,1_{(2)} \leq M < 1_{(2)})$ в двоичной системе счисления первая цифра мантиссы после запятой всегда будет равна «1», т.е.

0,1.....

это можно использовать для увеличения диапазона представимых чисел в оперативной памяти, для чего диапазон представления мантиссы нормализованного числа в стандарте IEEE 754 меняется на диапазон $1_{(2)} \leq M < 2_{(2)}$.

Причем единица целой части мантиссы учитывается неявно (**неявная единица**), т.е. под нее не отводится бит. В таком виде операнд хранится в оперативной памяти процессора. При выполнении арифметических операций над операндом, при его извлечении из памяти в регистр процессора (формат ВВ) этот скрытый бит восстанавливается, т.е. присутствует в явном виде

Алгоритм преобразования вещественного десятичного числа в двоичное число с плавающей точкой формата IEEE 754 (на примере числа $= 8,125_{(10)}$)

1. Перевести целую часть вещественного числа в двоичную систему и поставить после нее десятичную точку (для заданного примера: 1000).
2. Перевести дробную часть вещественного числа в двоичную систему с точностью для определения значений всех битов мантииссы, предусмотренных форматом (KB, DB, VB) (для заданного примера: 0,0010...0)

3. Записать полученное значение дробной части после десятичной точки. Если количество разрядов мантиссы получилось меньше выделенного под нее, то дополнить дробную часть незначащими нулями справа до предусмотренного форматом размера. Для заданного примера:

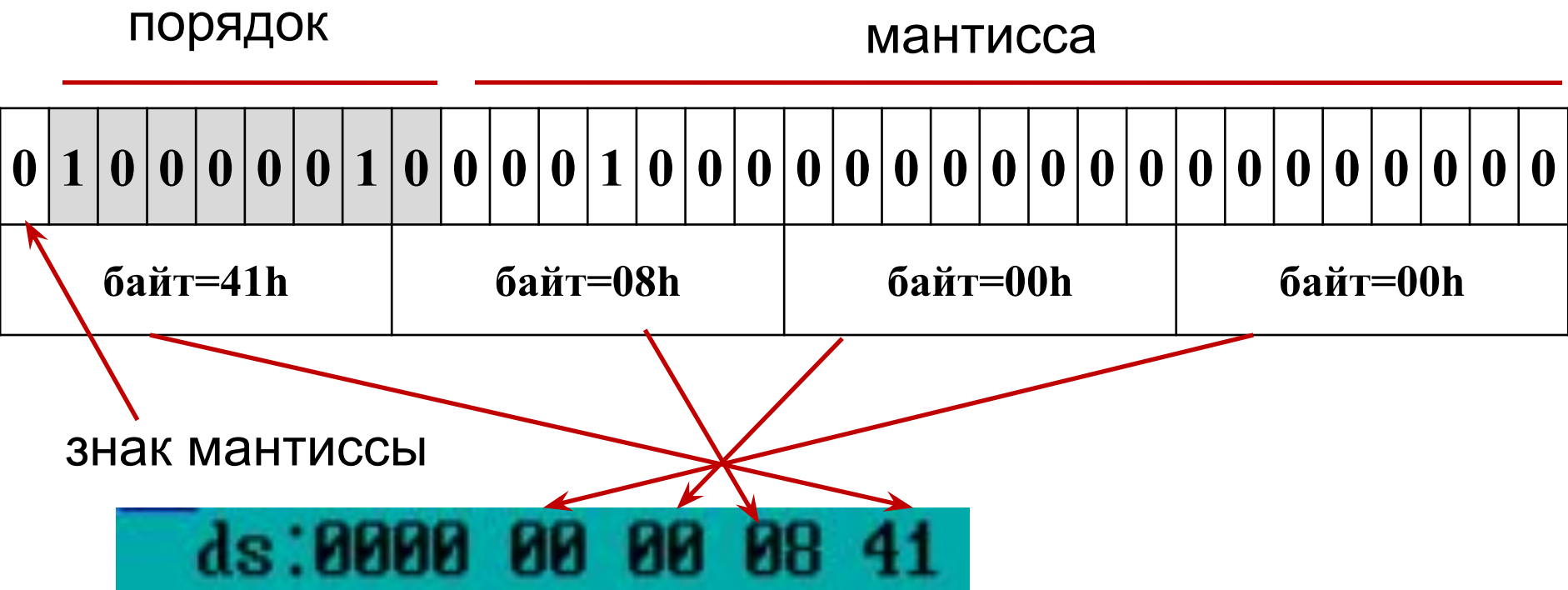
1000,00100000000000000000000000000000 (всего 31 бит)

4. Представить число в экспоненциальной форме. Для заданного примера:

0,100000100000000000000000000000000000*10¹⁰⁰

7. Записать значение порядка и значение мантииссы в соответствующие биты формата КВ или ДВ (у мантииссы, скрывается единица целой части).

8. Если число положительное, то в самый старший разряд (знак мантииссы) следует записать 0, если отрицательное, то 1.



Рассмотрим другие примеры представления операндов в формате КВ.

Напоминание:

- мантисса – M ($1_{(2)} \leq M < 2_{(2)}$);
- порядок – P (смещен на $127_{(10)}$).

Пример. Представить число $16,AC_{(16)}$ в формате КВ.

Перевод в двоичную систему:

$$16,AC_{(16)} = 10110,10101100_{(2)} = 1,011010101100_{(2)} * 10^{100}_{(2)}$$

$$M = 1,011010101100_{(2)}$$

$$P = 100_{(2)} + (127_{(10)} = 1111111_{(2)}) = 10000011_{(2)}$$

Тогда формат КВ этого числа (красным цветом выделены биты порядка) будет (целая часть мантиссы «скрыта»):

0	1	0	0	0	0	0	1	1	0	1	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
байт=41h								байт=0B5h								байт=60h								байт=00h													

Еще пример:

Представить число $-7,С8_{(16)}$ в формате КВ.

Перевод:

$$-7,С8_{(16)} = -111,11001000_{(2)} = -1,1111001000_{(2)} * 10^{10}.$$

$$M = -1,1111001000_{(2)};$$

$$P = 10_{(2)} + (127_{(10)} = 1111111_{(2)}) = 10000001_{(2)}.$$

Тогда формат КВ этого числа (красным цветом выделены биты порядка) будет (целая часть мантииссы «скрыта»):

1	1	0	0	0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
байт								байт								байт								байт										

Еще пример:

Представить число $-0,0C8_{(16)}$ в формате КВ.

Перевод:

$$-0,0C8_{(16)} = -0,000011001000_{(2)} = -1,1001000_{(2)} * 10^{-101}.$$

$$M = -1,1001000_{(2)};$$

$$P = -101_{(2)} + (127_{(10)} = 1111111_{(2)}) = 01111010_{(2)}.$$

Тогда формат КВ этого числа (красным цветом выделены биты порядка) будет (целая часть мантиссы «скрыта»):

1	0	1	1	1	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
байт								байт								байт								байт												

Специальные значения

Несмотря на большой диапазон вещественных чисел, представимых в формате с ПТ, бесконечное количество чисел находится за рамками представления в нём.

Для обработки тех случаев, в которых при вычислениях в форматах КВ и ДВ могут быть получены значения из непредставимого диапазона, в стандарте IEEE 754 предусмотрены специальные комбинации бит, называемые *специальными численными значениями*.

К специальным значениям относятся:

- денормализованные вещественные числа,
- отрицательная и положительная бесконечности,
- не числа;
- неопределенности,
- значения в неподдерживаемых форматах,
- ноль.

Для представления специальных значений зарезервированы порядки:

минимальный = 00000000 и

максимальный = 11111111

Ноль

Значение нуля относится к специальным.

Это делается из-за того, что ноль выделяется среди корректных вещественных значений, получаемых как результат какой-то арифметической операции.

Кроме того, ноль может формироваться как реакция процессора на определенную ситуацию.

Ноль представляется с нулевым порядком и нулевой мантисой. Например, в КВ:

00000000 00000000 00000000 00000000

Денормализованные вещественные числа

Денормализованные числа – числа, значение которых меньше минимально представимого в нормализованном виде в формате.

По мере приближения к нулю значение мантиссы уменьшается и наступает момент, когда разрядной сетки, отведенной под порядок, становится мало. В этот момент значение порядка обращается в нуль.

Число, при достижении которого это происходит на самом деле отлично от нуля, так как между истинным нулем и минимальным нормализованным находится еще бесконечное множество чисел.

Именно эти числа и представляют собой денормализованные числа.

Однако диапазон денормализованных чисел не безграничен и определяется количеством разрядов, отведенных под мантиссу.

Денормализованное число имеет нулевой порядок и ненулевую мантиссу, например для формата КВ:

00000000 00000100 00000111 00000000

Бесконечности

Среди причин, приводящих к формированию значения бесконечности, в первую очередь следует назвать переполнение и деление на нуль.

Бесконечность кодируется с максимальным значением порядка =11111111 и мантиссой равной 1.

Соответственно различают положительную и отрицательную бесконечности:

$+\infty$	->	0	11111111	1	00000000	00000000	00000000
$-\infty$	->	1	11111111	1	00000000	00000000	00000000

Не числа

К не числам относятся такие битовые последовательности, которые не совпадают ни с одним из рассмотренных ранее форматов.

Не числа представляются с любым знаком, максимальным порядком и любой мантиссой кроме 1.

Например:

01111111 00000**1**00 00000000 00000000

Формат двоично-десятичного кода (BCD - код)

Формат двоично-десятичного кода (BCD – binary coded decimal) в компьютерах используется для экономических расчетов.

Так же как и в формате с ФТ, вычисления, выполняемые процессором в этом формате, точные.

Представление числа в BCD-коде – это такое представление, когда каждая десятичная цифра кодируется четырёхбитовым двоичным кодом.

В настоящее время общепринятым стандартом представления операндов в BCD коде является стандарт **IEEE 754**. В его основе лежит использование кода 8421:

Десятичная цифра	Код 8421
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

В соответствии со стандартом IEEE 754 под операнд отводится 10 байт, из них крайний левый байт – знаковый.

В остальных (числовых байтах) записывается по две десятичные цифры (тетрадами).

Это позволяет обрабатывать 18-разрядные десятичные числа, что является достаточным для любых экономических расчетов.

Формат BCD-кодирования по стандарту IEEE 754

байт 9	байт 8	байт 7	байт 6	байт 5	байт 4	байт 3	байт 2	байт 1	байт 0
знак	$d_{17}d_{16}$	$d_{15}d_{14}$	$d_{13}d_{12}$	$d_{11}d_{10}$	d_9d_8	d_7d_6	d_5d_4	d_3d_2	d_1d_0

Пример:

представить $-1231,05_{(10)}$ в VCD коде стандарта IEEE 754.

байт 9	байт 8	байт 7	байт 6	байт 5	байт 4	байт 3	байт 2	байт 1	байт 0
11111111	00000000	00000000	00000000	00000000	00000000	00000000	00010010	00110001	00000101
—	00	00	00	00	00	00	12	31	05

байт 2	байт 1	байт 0
00010010	00110001	00000101

Кодирование чисел в цифровых процессорах

Кодирование чисел в цифровых процессорах

В цифровых процессорах в каждом из ранее рассмотренных форматов операнды представляются в одном из следующих кодов:

- прямом;
- дополнительном;
- обратном.

Введем следующую систему обозначений:

- $[X]_1$ – представление операнда X в прямом коде;
- $[X]_2$ – представление операнда X в дополнительном коде;
- $[X]_3$ – представление операнда X в обратном коде.

Рассмотрим представление операндов в прямом, обратном и дополнительном кодах в формате с фиксированной точкой.

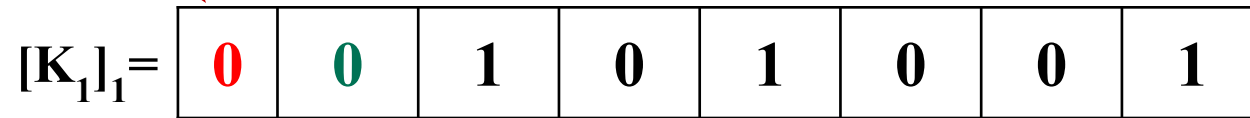
Прямой код

Представление операнда в прямом коде состоит из двух частей: знака числа, под него отводится крайний левый бит (старший) и модуля числа.

Если число положительное – знак кодируется 0, а если отрицательное, то 1.

Рассмотрим примеры представления целых операндов в этом коде:

$$K_1 = +51_{(8)} = +101001_{(2)}$$



$$K_2 = -51_{(8)} = -101001_{(2)}$$



$$M_1 = 0,15_{(8)} = +0,001101_{(2)}$$

$[M_1]_1 =$	0	0	0	1	1	0	1	0
-------------	----------	----------	----------	----------	----------	----------	----------	----------



$$M_2 = -0,15(8) = +0,001101_{(2)}$$

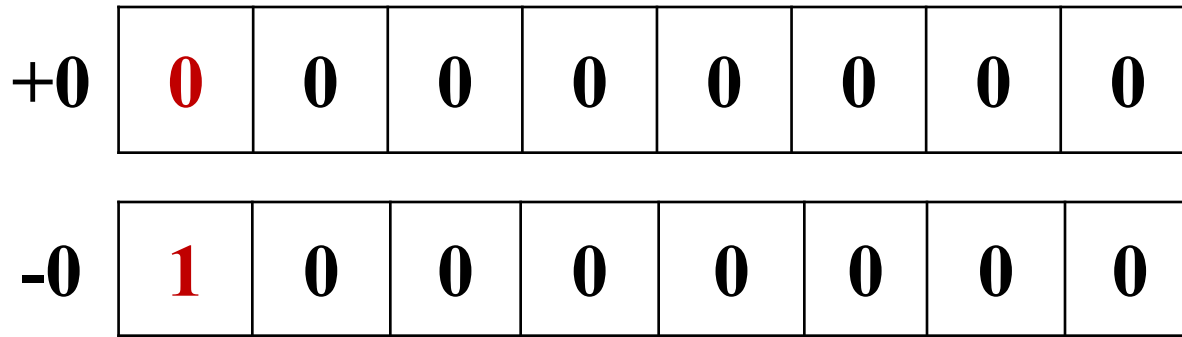
$[M_2]_1 =$	1	0	0	1	1	0	1	0
-------------	----------	----------	----------	----------	----------	----------	----------	----------



Достоинство прямого кода состоит в том, что в нем более просто реализуются алгоритмы выполнения «длинных» (по времени выполнения) операций – умножения и деления.

Недостатками прямого кода являются:

- 1) двойное представление нуля, которое должно быть учтено либо аппаратными, либо программными средствами;



- 2) представление операнда в виде двух частей (знака и модуля) приводит к достаточно сложным алгоритмам операций сложения/вычитания в цифровом процессоре.

Обратный код

Обратный код в настоящее время практически не используется.

Обратный код положительного числа совпадает с самим числом, а знаковый разряд кодируется 0.

Обратный код отрицательного числа получается по следующему алгоритму: в знаковый бит записывается 1, а остальные биты получают инверсией разрядов исходного числа. Например:

$$K_1 = 51_{(8)} = 101001_{(2)}$$

$[K_1]_3 =$	0	0	1	0	1	0	0	1
-------------	----------	----------	----------	----------	----------	----------	----------	----------

$$K_2 = -51_{(8)} = -101001_{(2)}$$

$[K_2]_3 =$	1	1	0	1	0	1	1	0
-------------	----------	----------	----------	----------	----------	----------	----------	----------

Достоинствами обратного кода являются:

- знаковый и числовые биты операнда составляют единый код, поэтому просто реализуются алгоритмы сложения и вычитания (но следует отметить, что они немного сложнее, чем в дополнительном коде);
- симметричный диапазон представления положительных и отрицательных чисел в цифровом процессоре.

Недостатками обратного кода являются:

- двойное представление нуля:

+0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

-0

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

- более сложные по сравнению с прямым кодом алгоритмы операций умножения и деления.

Дополнительный код

Дополнительный код положительного числа совпадает с самим числом, представленным в заданном формате, а знаковый разряд кодируется 0.

Дополнительный код отрицательного числа получается:

- для целых чисел – дополнением до модуля, по которому работает процессор (отсюда и название кода);
- для правильных дробей дополнением до "1".

Этот способ получения дополнительного кода следует из теории чисел.

Например, для 16-разрядного процессора, который работает по модулю $2^{16}=65536_{(10)}$ дополнением для $-36_{(10)}$ будет:

$65530_{(10)}$.

Дополнительный код (как и прямой и обратный) можно использовать в любой системе счисления.

Рассмотрим пример использования дополнительного кода в десятичной системе счисления.

Допустим есть трехразрядный **десятичный** процессор (тогда он работает по модулю $1000_{(10)}$).

Выполнить сложение в нем в дополнительном коде:

$$456 + (-126) = 330$$

Представляем слагаемые в дополнительном коде:

$$[456]_2 = 456$$

$$[-126]_2 = 874 \text{ т.е. дополнение } 126 \text{ до } 1000_{(10)}$$

	4	5	6
+	8	7	4
=	1 3	3	0

В цифровом процессоре, дополнительный код двоичного числа (с учетом формата представления) получается более простым способом.

Алгоритм следующий:


если число положительное, то в знаковый бит записывается 0, а само число – в остальную разрядную сетку (с учетом формата);

если число отрицательное, то в знаковый бит записывается 1, остальные биты получают инверсией разрядов числа с добавлением 1 в младший разряд. Т.е. каждый разряд дополняется до 2 (основания системы счисления) и плюс 1 в младший.

По этому же алгоритму можно получить дополнительный код числа в любой системе счисления. Рассмотрим десятичную систему счисления и предыдущий пример.

456 - число положительное, поэтому $[456]_2 = 456$

-126 - число отрицательное, поэтому дополняем каждый разряд до 9 и в младший разряд прибавляем 1.

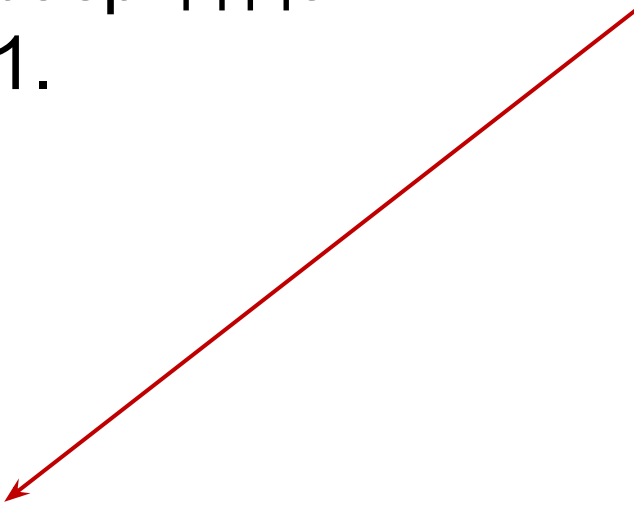

$$\begin{array}{r} 999 \\ [-126]_2 = 874 \quad 126 \\ \underline{873} + 1 = 874 \end{array}$$

Рассмотрим пример получения дополнительного кода в шестнадцатеричной системе счисления.

Представить $(-6AF9_{(16)})$ в дополнительном коде.

Дополняем каждый разряд до F и к младшему разряду добавляем 1.

6	A	F	9
9	5	0	6
+			1
9	5	0	7



$$(-6AF9_{(16)}) = [9507]_2$$

Примеры для 8-разрядного процессора:

$$K_1 = 51_{(8)} = 101001_{(2)}$$

0	0	1	0	1	0	0	1
----------	----------	----------	----------	----------	----------	----------	----------

$$M_1 = -0,15_{(8)} = -0,0011010_{(2)}$$

0	0	0	1	1	0	1	0
----------	----------	----------	----------	----------	----------	----------	----------

ИНВЕРСИЯ

1	1	1	0	0	1	0	1
----------	----------	----------	----------	----------	----------	----------	----------

+ 1

1	1	1	0	0	1	1	0
----------	----------	----------	----------	----------	----------	----------	----------

01+1=10

1	1	1	0	0	1	1	0
----------	----------	----------	----------	----------	----------	----------	----------

$$L_1 = 51,15_{(8)} = 101001,001101_{(2)}$$

$$[L_1]_2 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ \hline \end{array}$$

$$L_2 = -51,15_{(8)} = -101001,001101_{(2)}$$

$$|L_2| = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ \hline \end{array}$$

$$\text{ИНВЕРСИЯ} \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline \end{array}$$

$$+1 \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array}$$

$$[L_2]_2 = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array}$$

целая часть

дробная часть

Перевод числа из дополнительного кода

Перевод в двоичную систему счисления из дополнительного кода выполняется (с учетом формата представления) **по тому же алгоритму**, что и в дополнительный:

если знаковый бит равен 0 (число положительное), то знак числа – «плюс», а само число берется из остальных битов;

если знаковый бит равен 1 (число отрицательное), то знак числа – «минус», а остальные разряды числа получают инверсией битов с **добавлением** 1 в младший бит.

$[L_1]_2 =$

0	0	1	0	1	0	0	1	0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$L_1 = +101001,001101_{(2)} = 51,15_{(8)}$

$[L_2]_2 =$

1	1	0	1	0	1	1	0	1	1	0	0	1	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

ИНВЕРСИЯ

1	0	1	0	1	0	0	1	0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

+1

1	0	1	0	1	0	0	1	0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$L_2 = -101001,001101_{(2)} = -51,15_{(8)}$

Достоинство дополнительного кода в том, что знаковый и числовые биты операнда составляют единый код, а не две части, как в прямом. Как следствие этого, алгоритмы сложения и вычитания реализуются более просто.

- Недостатками дополнительного кода** являются:
- несимметричный диапазон представления положительных и отрицательных чисел в цифровом процессоре (диапазон отрицательных больше на одно число, чем положительных);
 - более сложные по сравнению с прямым кодом алгоритмы операций умножения и деления.

Несимметричный диапазон представления положительных и отрицательных чисел в цифровом процессоре в дополнительном коде приводит к тому, что для минимально отрицательного числа используется особый алгоритм перевода в дополнительный код и из кода в двоичную систему.

Перевод в дополнительный код для минимально отрицательного числа (-1000...00) выполняется по следующему алгоритму:

единица в знаковом разряде кодирует «минус», а числовые разряды равны 000...00, т.е.

1000...00

Перевод из дополнительного кода для минимально отрицательного числа (1000...00)

выполняется по следующему алгоритму:

единица в знаковом разряде дает «минус», а модуль числа равен 1000...00, т.е. "1" в знаке дает и "-" и числовую единицу:

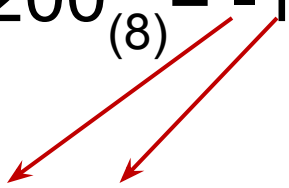
– 1000...00

Например, для 8-разрядного процессора
максимально представимые числа:

□ максимальное положительное число равно
 $127_{(10)} = 177_{(8)} = 1111111_{(2)}$ и его дополнительный код
равен:

$$127_{(10)} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline \end{array}$$

□ минимальное отрицательное число равно
 $-128_{(10)} = -200_{(8)} = -10000000_{(2)}$ и его дополнительный
код равен:

$$-128_{(10)} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \end{array}$$


Кодирование символьной информации

Первые цифровые процессоры обрабатывали только числовые данные. Сейчас процессоры в большей степени обрабатывает не числа, а текстовую информацию, расположенную в базе данных. Но и этом случае каждый символ используемого алфавита имеет свой числовой код.

Для представления символьной информации в компьютере используются кодовые таблицы.

В кодовой таблице каждому символу соответствует код этого символа (целое число).

Таким образом при вводе, выводе, хранении и обработке символьной информации все операции выполняются над соответствующими кодами (числами).

До недавнего времени наиболее широко использовался код ASCII (American Standard Code for Interchange Information) – американский стандартный код для обмена информацией. Символы в этом коде кодируются одним байтом. Таким образом, кодовая таблица содержит 256 различных символов.

Половина кодов является постоянными (управляющие символы, символы английского алфавита, цифровые символы, знаки пунктуации и др.), а другая половина может быть изменена (например, часть символов может быть заменена на символы русского или любого другого алфавита).

Фрагмент таблицы 8-битного кода ASCII

American Standard Code for Information Interchange. Коды символов даны в 16-ричной системе счисления.

Число	Символ	Число	Символ	Число	Символ	Число	Символ	Число	Символ	Число	Символ
20	пробел	30	.	40	@	50	P	60	'	70	p
21	!	31	0	41	A	51	Q	61	a	71	q
22	"	32	1	42	B	52	R	62	b	72	r
23	#	33	2	43	C	53	S	63	c	73	s
24	\$	34	3	44	D	54	T	64	d	74	t
25	%	35	4	45	E	55	U	65	e	75	u
26	&	36	5	46	F	56	V	66	f	76	v
27	'	37	6	47	G	57	W	67	g	77	w
28	(38	7	48	H	58	X	68	h	78	x
29)	39	8	49	I	59	Y	69	i	79	y
2A	*	3A	9	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	:	4B	K	5B	[6B	k	7B	{
2C	,	3C	;	4C	L	5C	\	6C	l	7C	
2D	-	3D	<	4D	M	5D]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

Пример кодировки слов way и WAY:

77(w)

61(a)

79(y)

way=

0	1	1	1	0	1	1	1	0	1	1	0	0	0	0	1	0	1	1	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

57(W)

41(A)

59(Y)

WAY=

0	1	0	1	0	1	1	1	0	1	0	0	0	0	0	1	0	1	0	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

На основе этого кода, с целью совместимости, был создан отечественный код "КОИ-8" (код обмена информацией - восьмибитный).

КОИ-8 — восьмибитовая ASCII-совместимая кодовая таблица, созданная для кодирования букв кириллических алфавитов.

В КОИ-8 символы русского алфавита поместили в верхнюю часть кодовой таблицы так, что позиции кириллических символов соответствуют их фонетическим аналогам в английском алфавите в нижней части таблицы.

Это значит, что убрав в тексте, написанном в КОИ-8, левый восьмой бит каждого символа, то получится текст, написанный латинскими символами.

Недостатком ASCII явилось ограниченное количество символов (256), что затрудняло его использование для представления многоязычных текстов.

Для решения проблем, связанных с унификацией символьной информации, был предложен стандарт Unicode (Юникод).

Этот код покрывает языки Америки, Европы, Среднего Востока, Африки, Индии, Азии и Океании и технические символы.

Эта система позволяет закодировать свыше миллиона символов (1 114 112).

Большинство символов, используемых в основных языках мира занимают 65 536 кодовых комбинаций.

Остальных (более миллиона) кодовых точек вполне достаточно для кодирования всех известных символов, включая даже исторические знаки и редкие языки.

Стандарт UNICODE имеет три формы:

- 32-битную (UTF-32);
- 16-битную (UTF-16);
- 8-битную (UTF-8).

Весьма распространенная восьмибитная форма UTF-8 была создана для удобной совместимости с ASCII-ориентированными системами кодирования.

Сейчас UNICODE содержит 96 382 символа, их более чем достаточно для общения на всех известных языках мира, а также для написания классических (исторических) шрифтов многих языков (например, европейский алфавит, среднеазиатское письмо направленное справа налево, шрифты Азии, и другие).

Спасибо за внимание !!

Лабораторная работа

"Форматы представления чисел в цифровых процессорах"

Целые числа

(на ассемблере для 16-разрядного процессора)

сегмент данных

```
data segment
;=====
        A dw 19h ← =19(16)
data ends
;=====
```

сегмент кода

```
code segment
assume cs:code, ds:data, ss:nothing
start:      mov ax, data ;load adress
            mov ds, ax ; data segment
            ;=====
            ; "тело" ВЫЧИСЛЕНИЯ
            ;=====
quit:      mov ax, 4c00h ; cod to finish 0
            int 21h ; exit to dos
code ends
end start
```

Целые числа = 19₍₁₆₎

ds – сегмент данных; cs – сегмент кода

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: TD

File View Run Breakpoints Data Options Window Help

[] CPU 80486

Address	Instruction	Register/Value
cs:0000	mov ax,465A	ax 465A
cs:0003	mov ds,ax	bx 0000
cs:0005	mov ax,4C00	cx 0000
cs:0008	int 21	dx 0000
cs:000A	add [bx+si],al	si 0000
cs:000C	add [bx+si],al	di 0000
cs:000E	add [bx+si],al	bp 0000
cs:0010	add [bx+si],al	sp 0000
cs:0012	add [bx+si],al	ds 465A
cs:0014	add [bx+si],al	es 464A
cs:0016	add [bx+si],al	ss 4659
cs:0018	add [bx+si],al	cs 465B
cs:001A	db FF	ip 0005
cs:001B	inc word ptr [bx+si]	
cs:001D	add bh,bh	

Address	Hex	ASCII
ds:0000	19 00 00 00 00 00 00 00	↓
ds:0008	00 00 00 00 00 00 00 00	
ds:0010	BB 5A 46 8E DB BB 00 4C	ZFA L
ds:0018	CD 21 00 00 00 00 00 00	=?
ds:0020	00 00 00 00 00 00 00 00	

память Сегмент данных

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

Чтобы отобразить сегмент данных нужно по клавише F8 выполнить первые две команды программы:

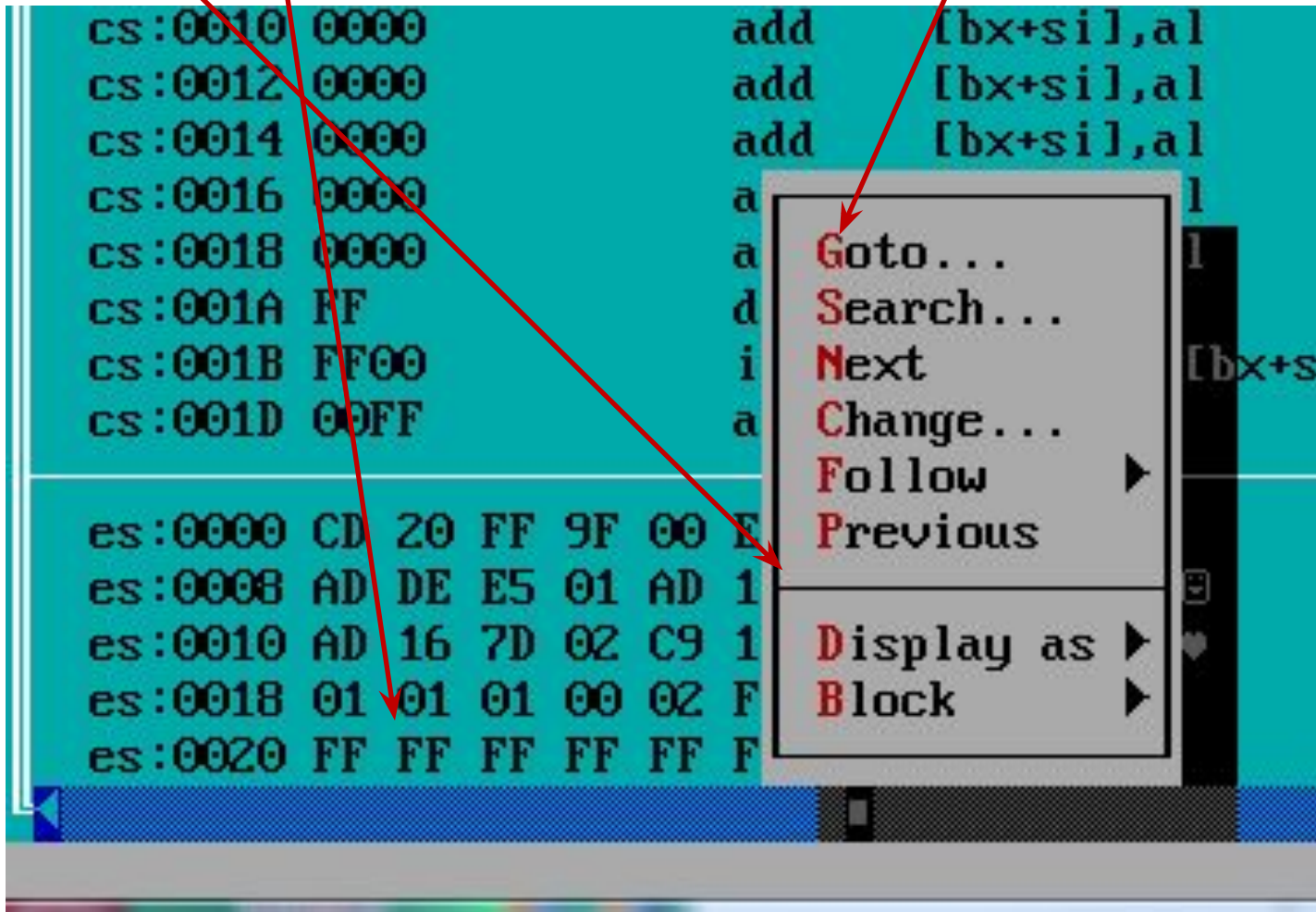
```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program:
File View Run Breakpoints Data Option
[ ]-CPU 80486
cs:0000 B85A46 mov ax,465A
cs:0003 8ED8 mov ds,ax
cs:0005 B8004C mov ax,4C00
cs:0008 CD21 int 21
cs:000A 0000 add [bx+si],al
cs:000C 0000 add [bx+si],al
cs:000E 0000 add [bx+si],al
cs:0010 0000 add [bx+si],al
cs:0012 0000 add [bx+si],al
cs:0014 0000 add [bx+si],al
cs:0016 0000 add [bx+si],al
cs:0018 0000 add [bx+si],al
cs:001A FF db FF
cs:001B FF00 inc word ptr [bx+s
cs:001D 00FF add bh,bh

ds:0000 CD 20 FF 9F 00 EA FF FF = f Ω
ds:0008 AD DE E5 01 AD 16 AF 01 ; |r|i->
ds:0010 AD 16 7D 02 C9 10 3F 03 i-}B|?♥
ds:0018 01 01 01 00 02 FF FF FF 000 0
ds:0020 FF FF FF FF FF FF FF FF
```

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program:
File View Run Breakpoints Data Opti
[ ]-CPU 80486
cs:0000 B85A46 mov ax,465A
cs:0003 8ED8 mov ds,ax
cs:0005 B8004C mov ax,4C00
cs:0008 CD21 int 21
cs:000A 0000 add [bx+si],al
cs:000C 0000 add [bx+si],al
cs:000E 0000 add [bx+si],al
cs:0010 0000 add [bx+si],al
cs:0012 0000 add [bx+si],al
cs:0014 0000 add [bx+si],al
cs:0016 0000 add [bx+si],al
cs:0018 0000 add [bx+si],al
cs:001A FF db FF
cs:001B FF00 inc word ptr [bx
cs:001D 00FF add bh,bh

es:0000 CD 20 FF 9F 00 EA FF FF = f Ω
es:0008 AD DE E5 01 AD 16 AF 01 ; |r|i->
es:0010 AD 16 7D 02 C9 10 3F 03 i-}B|?♥
es:0018 01 01 01 00 02 FF FF FF 000 0
es:0020 FF FF FF FF FF FF FF FF
```

Переключиться на сегмент данных (щелкнуть на нем правой кнопкой) и левой кнопкой, вызвать контекстное меню. Выбрать "Goto" нажать Enter:



```
cs:0010 0000 add [bx+si],al
cs:0012 0000 add [bx+si],al
cs:0014 0000 add [bx+si],al
cs:0016 0000 a
cs:0018 0000 a
cs:001A FF d
cs:001B FF00 i [bx+s
cs:001D 00FF a

es:0000 CD 20 FF 9F 00 E
es:0008 AD DE E5 01 AD 1
es:0010 AD 16 7D 02 C9 1
es:0018 01 01 01 00 02 F
es:0020 FF FF FF FF FF F
```

The image shows a debugger window with a teal background. The top section displays assembly code for the CS segment, and the bottom section displays memory data for the ES segment. A context menu is open over the data segment, listing options: Goto..., Search..., Next, Change..., Follow, Previous, Display as, and Block. Red arrows indicate the sequence of actions: right-clicking on the data segment and then clicking on the 'Goto...' option.

Задать отображение памяти начиная с адреса 0000 в сегменте данных, нажать Enter:

The screenshot shows a debugger window with a memory dump and assembly code. A dialog box is overlaid on the assembly code, prompting for an address. The dialog title is "Enter address to position to" and the input field contains "ds:0_". The dialog has "OK", "Cancel", and "Help" buttons. A red arrow points from the text above to the dialog title.

```
cs:0014 0000 add [bx+si],al
cs:0016 0000 add [bx+si],al
cs:0018 0000 add [bx+si],al
cs:001A FF db
cs:001B FF00 inc
cs:001D 00FF add
```

es:0000 CD 20 FF 9F 00 EA
es:0008 AD DE E5 01 AD 16
es:0010 AD 16 7D 02 C9 10
es:0018 01 01 01 00 02 FF FF FF 0000
es:0020 FF FF FF FF FF FF FF FF ss:0000

Enter item prompted for in dialog title

Побайтовое отображение памяти с ds:0

```
CS:001D  00FF          add      bh, bh
ds:0000  19 00 99 02 52 34 29 00  ↓ 00R4)
ds:0008  00 00 00 00 00 00 00 00
ds:0010  BB 5A 46 8E D8 BB 00 4C  7ZFÄ† L
ds:0018  CD 21 00 00 00 00 00 00  =!
ds:0020  00 00 00 00 00 00 00 00
```

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next

адреса
памяти

содержимое
по адресам памяти

СИМВОЛЬНЫЙ
код
байтов

Целые числа = $19_{(16)}$ и $299_{(16)}$

```
LEK1 - Блокнот
Файл  Правка  Формат  Вид  Справка
data segment
;=====
      A dw 19h
      B dw 299h
data ends
;=====
code segment
assume cs:code, ds:data, ss:nothing
start:      mov ax, data ;load adress
            mov ds, ax ; data segment
            ;=====
            ; "тело" ВЫЧИСЛЕНИЯ
            ;=====
quit:      mov ax, 4c00h ; cod to finish 0
            int 21h ; exit to dos
code ends
end start
```

сегмент данных

сегмент кода

Целые числа

19₍₁₆₎ и 229₍₁₆₎

```
DOSBox 0.74, Cpu speed: 300% cycles Frames in 0 Program: TD
File View Run Breakpoints Data Options
[ ]=CPU 80486
cs:0000 B85A46      mov     ax,465A
cs:0003 8ED8          mov     ds,ax
cs:0005 B8004C          mov     ax,4C00
cs:0008 CD21          int     21
cs:000A 0000          add     [bx+si],al
cs:000C 0000          add     [bx+si],al
cs:000E 0000          add     [bx+si],al
cs:0010 0000          add     [bx+si],al
cs:0012 0000          add     [bx+si],al
cs:0014 0000          add     [bx+si],al
cs:0016 0000          add     [bx+si],al
cs:0018 0000          add     [bx+si],al
cs:001A FF           db     FF
cs:001B FF00        inc     word ptr [bx+si]
cs:001D 00FF        add     bh,bh

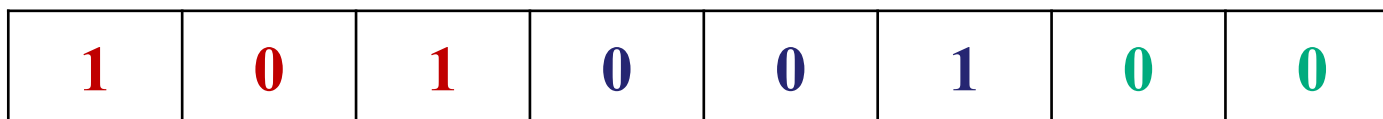
ds:0000 19 00 99 02 00 00 00 00 ↓ 00
ds:0008 00 00 00 00 00 00 00 00
ds:0010 BB 5A 46 8E D8 B8 00 4C ǀZǂǂǂ L
ds:0018 CD 21 00 00 00 00 00 00 =!
ds:0020 00 00 00 00 00 00 00 00
```

Формат представления правильных дробей

Дробное число M_1 , заданное в восьмеричной системе счисления, представить в формате с ФТ в 8-разрядном процессоре (1 байт).

$$M = 0,51_{(8)} = 0,101001_{(2)}$$

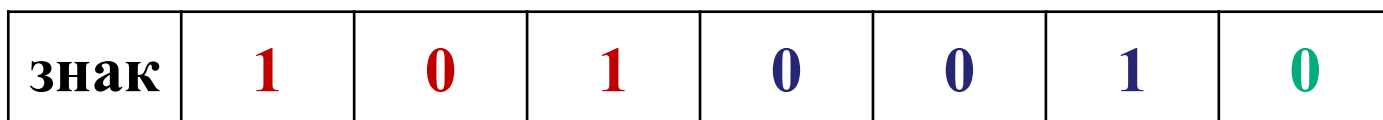
без знака:



заполнение разрядов



со знаком:



$\approx 0, A4_{(16)}$

5

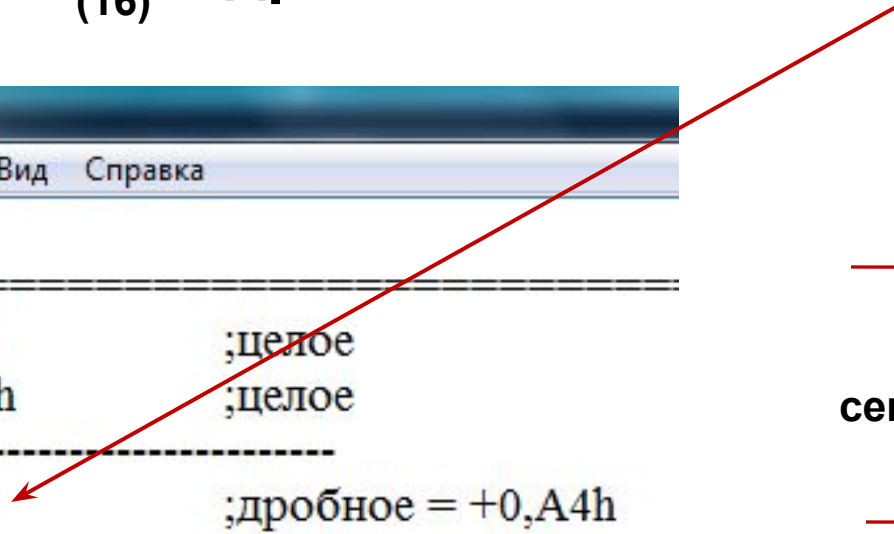
2

Целые числа $19_{(16)}$ и $229_{(16)}$ и дробное со знаком $+0,A4_{(16)}$

```
LEK1 - Блокнот
Файл  Правка  Формат  Вид  Справка
|data segment
;=====
      A dw 19h          ;целое
      B dw 299h        ;целое
;-----
      C db 52h         ;дробное = +0,A4h
data ends
;=====
code segment
assume cs:code, ds:data, ss:nothing
start:      mov ax, data ;load adress
            mov ds, ax ; data segment
            ;=====
            ; "тело" ВЫЧИСЛЕНИЯ
            ;=====
quit:      mov ax, 4c00h ; cod to finish 0
            int 21h ; exit to dos
code ends
end start
```

сегмент данных

сегмент кода



Целые числа $19_{(16)}$ и $229_{(16)}$ и дробное $0,A4_{(16)}$

```
DOS BOX DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip: 0, Program: TF
File View Run Breakpoints Data Options
[ ] CPU 80486
cs:0000 B85A46 mov ax,465A
cs:0003 8ED8 mov ds,ax
cs:0005 B8004C mov ax,4C00
cs:0008 CD21 int 21
cs:000A 0000 add [bx+si],al
cs:000C 0000 add [bx+si],al
cs:000E 0000 add [bx+si],al
cs:0010 0000 add [bx+si],al
cs:0012 0000 add [bx+si],al
cs:0014 0000 add [bx+si],al
cs:0016 0000 add [bx+si],al
cs:0018 0000 add [bx+si],al
cs:001A FF db FF
cs:001B FF00 inc word ptr [bx+si]
cs:001D 00FF add bh,bh

ds:0000 19 00 99 02 52 00 00 00 ↓ ÜÖR
ds:0008 00 00 00 00 00 00 00 00
ds:0010 BB 5A 46 8E D8 B8 00 4C ¶ZFA¶ L
ds:0018 CD 21 00 00 00 00 00 00 =!
ds:0020 00 00 00 00 00 00 00 00

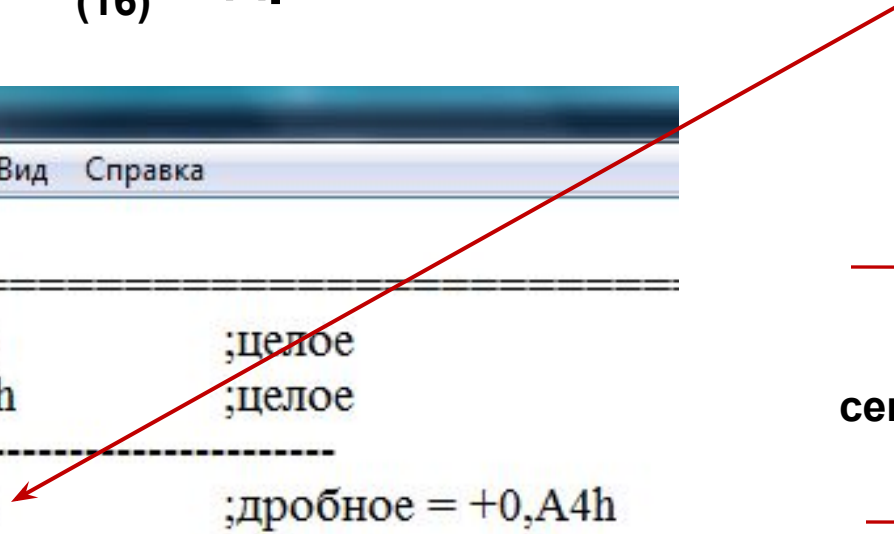
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-
```

Целые числа $19_{(16)}$ и $229_{(16)}$ и дробное со знаком $+0,A4_{(16)}$

```
LEK1 - Блокнот
Файл  Правка  Формат  Вид  Справка
data segment
;=====
    A dw 19h          ;целое
    B dw 299h        ;целое
;-----
    C db 52h         ;дробное = +0,A4h
data ends
;=====
code segment
assume cs:code, ds:data, ss:nothing
start:
    mov ax, data ;load adress
    mov ds, ax ; data segment
;=====
; "тело" ВЫЧИСЛЕНИЯ
;=====
quit:
    mov ax, 4c00h ; cod to finish 0
    int 21h ; exit to dos
code ends
end start
```

сегмент данных

сегмент кода



Целые числа $19_{(16)}$ и $229_{(16)}$ и дробное $0,A4_{(16)}$

```
DOS BOX  DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip: 0, Program: TF
File View Run Breakpoints Data Options
[ ] CPU 80486
cs:0000 B85A46      mov     ax,465A
cs:0003 8ED8        mov     ds,ax
cs:0005 B8004C        mov     ax,4C00
cs:0008 CD21        int     21
cs:000A 0000        add     [bx+si],al
cs:000C 0000        add     [bx+si],al
cs:000E 0000        add     [bx+si],al
cs:0010 0000        add     [bx+si],al
cs:0012 0000        add     [bx+si],al
cs:0014 0000        add     [bx+si],al
cs:0016 0000        add     [bx+si],al
cs:0018 0000        add     [bx+si],al
cs:001A FF          db     FF
cs:001B FF00       inc     word ptr [bx+si]
cs:001D 00FF       add     bh,bh

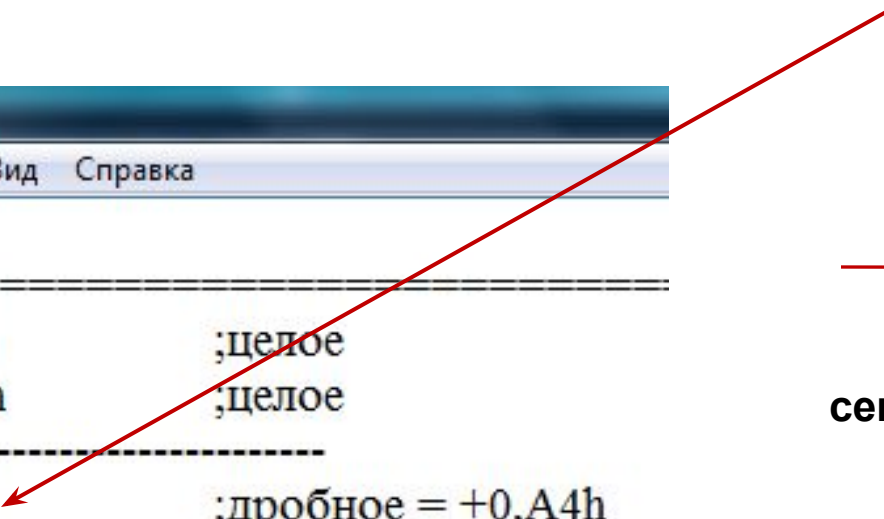
ds:0000 19 00 99 02 52 00 00 00 ↓ ÜØR
ds:0008 00 00 00 00 00 00 00 00
ds:0010 BB 5A 46 8E D8 B8 00 4C ¶ZFA¶ L
ds:0018 CD 21 00 00 00 00 00 00 =!
ds:0020 00 00 00 00 00 00 00 00

F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-
```

```
LEK1 - Блокнот
Файл  Правка  Формат  Вид  Справка
|data segment
;=====
      A dw 19h          ;целое
      B dw 299h        ;целое
;-----
      C db 52h          ;дробное = +0,A4h
data ends
D dw 2934h
;=====
code segment
assume cs:code, ds:data, ss:nothing
start:
      mov ax, data ;load adress
      mov ds, ax ; data segment
;=====
; "тело" ВЫЧИСЛЕНИЯ
;=====
quit:
      mov ax, 4c00h ; cod to finish 0
      int 21h ; exit to dos
code ends
end start
```

сегмент данных

сегмент кода



Целые числа $19_{(16)}$ и $229_{(16)}$; дробное $0,А4_{(16)}$; смешанное $29,34_{(16)}$

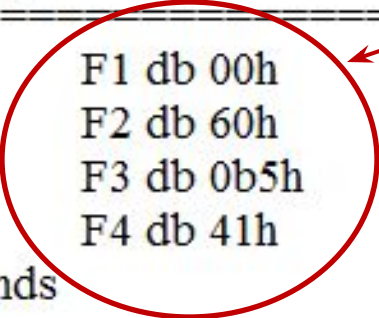
```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: TD
File View Run Breakpoints Data Options Wind
[ ]=CPU 80486
cs:0000 B85A46 mov ax,465A
cs:0003 8ED8 mov ds,ax
cs:0005 B8004C mov ax,4C00
cs:0008 CD21 int 21
cs:000A 0000 add [bx+si],al
cs:000C 0000 add [bx+si],al
cs:000E 0000 add [bx+si],al
cs:0010 0000 add [bx+si],al
cs:0012 0000 add [bx+si],al
cs:0014 0000 add [bx+si],al
cs:0016 0000 add [bx+si],al
cs:0018 0000 add [bx+si],al
cs:001A FF db FF
cs:001B FF00 inc word ptr [bx+si]
cs:001D 00FF add bh,bh

ds:0000 19 00 99 02 52 34 29 00 L ÜBR4)
ds:0008 00 00 00 00 00 00 00 00
ds:0010 BB 5A 46 8E DB BB 00 4C 7ZFÄ† L
ds:0018 CD 21 00 00 00 00 00 00 =!
ds:0020 00 00 00 00 00 00 00 00
```

смешанное $29,34_{(16)}$

Вещественное число 16,АС₍₁₆₎ в формате КВ

```
LEK11 - Блокнот
Файл  Правка  Формат  Вид  Справка
data segment
;=====
F1 db 00h           ;младший байт
F2 db 60h
F3 db 0b5h
F4 db 41h           ;старший байт
data ends
;=====
code segment
assume cs:code, ds:data, ss:nothing
start:
    mov ax, data ;load adress
    mov ds, ax ; data segment
    ;=====
    ; "тело" ВЫЧИСЛЕНИЯ
    ;=====
quit:
    mov ax, 4c00h ; cod to finish 0
    int 21h ; exit to dos
code ends
end start
```



сегмент данных

сегмент кода

Вещественное число 16,АС₍₁₆₎ в формате КВ

```
DOS BOX  DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: TC
File View Run Breakpoints Data Options
[ ] CPU 80486
cs:0000 B85A46      mov     ax,465A
cs:0003 8ED8        mov     ds,ax
cs:0005 B8004C        mov     ax,4C00
cs:0008 CD21        int     21
cs:000A 0000        add     [bx+si],al
cs:000C 0000        add     [bx+si],al
cs:000E 0000        add     [bx+si],al
cs:0010 0000        add     [bx+si],al
cs:0012 0000        add     [bx+si],al
cs:0014 0000        add     [bx+si],al
cs:0016 0000        add     [bx+si],al
cs:0018 0000        add     [bx+si],al
cs:001A FF          db     FF
cs:001B FF00       inc     word ptr [bx+si]
cs:001D 00FF       add     bh,bh

ds:0000 00 60 B5 41 00 00 00 00  'A
ds:0008 00 00 00 00 00 00 00 00
ds:0010 B8 5A 46 8E D8 B8 00 4C  ZFA L
ds:0018 CD 21 00 00 00 00 00 00  =!
ds:0020 00 00 00 00 00 00 00 00
```

СПАСИБО ЗА ВНИМАНИЕ!