ТЕМА 2 АРИФМЕТИЧЕСКИЕ ОСНОВЫ ЦИФРОВЫХ СИСТЕМ

2.1 СИСТЕМЫ СЧИСЛЕНИЯ

В повседневной жизни для человека наиболее привычной является десятичная система счисления (10 цифр – 10 пальцев у человека; кстати, слово «разряд» (digit) означает «палец» по латыни).

ДЕСЯТИЧНАЯ СИСТЕМА, также называемая «системой по основанию 10», состоит из 10 цифр-символов: 0, 1, 2, 3, 4, 5, 6, 7, 8. 9, которые называются «**АЛФАВИТОМ ДЕСЯТИЧНОЙ СИСТЕМЫ**»; используя их, можно выразить любую величину (целую или дробную).

Десятичная система – это **ПОЗИЦИОННАЯ СИСТЕМА**, в которой значение (или вес) разряда зависит от его положения (позиции). Например, в десятичном числе 435 – цифра 4 представляет собой четыре сотни, 3 – три десятка, а 5 – пять единиц.

Следующий пример – число 5238,47. Это число состоит из пяти тысяч, двух сотен, трех десятков, восьми единиц, четырех десятых и семи сотых:

$$5238{,}47 = (5 \cdot 1000) + (2 \cdot 100) + (3 \cdot 10) + (8 \cdot 1) + (4 \cdot 0{,}1) + (7 \cdot 0{,}01).$$

Десятичная точка (или десятичная запятая) используется для отделения целой части числа от дробной. Различные позиции относительно десятичной точки указывают вес, выраженный степенью 10. Десятичная запятая отделяет **положительные степени** 10 от **отрицательных**:

$$5238{,}47 = (5 \cdot 10^3) + (2 \cdot 10^2) + (3 \cdot 10^1) + (8 \cdot 10^0) + (4 \cdot 10^{-1}) + (7 \cdot 10^{-2}). \quad (2.1)$$

10^3	10^2	10^1	10^0		10^{-1}	10^{-2}
5	2	3	8	,	4	7

Рис. 2.1 – *Разряды десятичных чисел*

К сожалению, десятичная система счисления не применяется при реализации цифровых систем. Очень сложно спроектировать электронное оборудование, которое работает с десятью различными уровнями напряжения (каждый уровень представляет один десятичный символ от 0 до 9).

С другой стороны, очень легко спроектировать простую и точную электронную схему, которая работает только с двумя уровнями напряжения – низкий уровень, высокий уровень.

В **ДВОИЧНОЙ СИСТЕМЕ** существует всего два символа, т.е. **алфавит двоичной системы** – «0» и «1», но даже при этом система «с основанием 2» может представлять любые целые и дробные числа.

Перечисленные ранее правила десятичной системы применимы и к двоичной системе. Пример двоичного числа:

$$1011{,}101_2 = (1 \cdot 2^3) + (0 \cdot 2^2) + (1 \cdot 2^1) + (1 \cdot 2^0) + (1 \cdot 2^{-1}) + (0 \cdot 2^{-2}) + (1 \cdot 2^{-3}) =$$
$$= 8 + 0 + 2 + 1 + 0{,}5 + 0 + 0{,}125 = 11{,}625_{10}. \qquad (2.2)$$

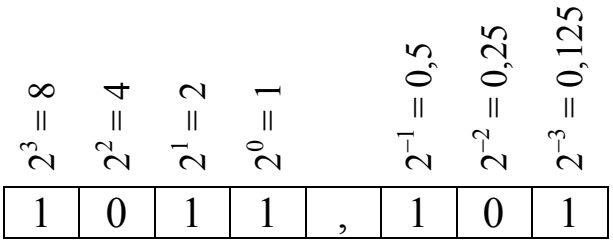


Рис. 2.2 – *Разряды двоичных чисел*

В двоичной системе термин «**двоичный разряд**» обозначается словом «**бит**». Таким образом, в показанном примере четыре бита слева от двоичной точки представляют его целую часть, а три бита справа от точки – его дробную часть.

Кроме десятичной и двоичной системы счисления в цифровых компьютерах используют также восьмеричную и шестнадцатеричную системы счисления.

Алфавит **восьмеричной системы** счисления состоит из восьми символов: 0, 1, 2, 3, 4, 5, 6, 7. Перечисленные ранее правила десятичной и двоичной системы применимы и к восьмеричной системе счисления:

$$372{,}6_8 = (3 \cdot 8^2) + (7 \cdot 8^1) + (2 \cdot 8^0) + (6 \cdot 8^{-1}) =$$
$$= 3 \cdot 64 + 7 \cdot 8 + 2 \cdot 1 + 6 \cdot 0{,}125 = 250{,}75_{10} \qquad (2.3)$$

Алфавит **ШЕСТНАДЦАТЕРИЧНОЙ СИСТЕМЫ** счисления состоит из шестнадцати символов – это числа: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 и шесть первых букв латинского алфавита: A, B, C, D, E, F. Символ A обозначает шестнадцатеричное число 10, символ B – число 11, символ C – число 12, символ D – число 13, символ E – число 14, символ F – число 15.

Вес каждого разряда определяется аналогично ранее приведенным системам счисления – степенями числа 16:

$$3C2{,}6_{16} = (3 \cdot 16^2) + (12 \cdot 16^1) + (2 \cdot 16^0) + (6 \cdot 16^{-1}) =$$
$$= 3 \cdot 256 + 12 \cdot 16 + 2 \cdot 1 + 6 \cdot 0{,}0625 = 962{,}375_{10} \qquad (2.4)$$

Обратите внимание – во всех системах счисления основание системы обозначается одинаковыми символами. В десятичной системе – основание: 10_{10}. В двоичной системе – основание: $10_2 = 2_{10}$. В восьмеричной системе – основание: $10_8 = 8_{10}$. В шестнадцатеричной системе – основание: $10_{16} = 16_{10}$.

Система счисления может указываться в виде подстрочной цифры после числа, как в примерах (2.2), (2.3), (2.4) или латинскими буквами:

B / b – binary – двоичные числа;
Q / q – octal – восьмеричные числа (чтобы не путать цифру «0» и букву «О» используют похожий символ – «Q»);
D / d– decimal – десятичные числа;
H / h– hexadecimal – шестнадцатеричные числа.

2.1.1 Преобразование чисел с разными системами счисления

В компьютеры исходные числа (операнды) вводятся в привычной для всех десятичной системе счисления. Внутри компьютера эти числа преобразовываются в двоичную систему. Над двоичными числами выполняются арифметические действия, и полученный двоичный результат перед выводом на экран переводится в десятичный формат. Поэтому студенты должны понимать алгоритмы и методы преобразования чисел с различными системами счисления.

Преобразование двоичных чисел в десятичные осуществляется в соответствии с примером (2.2) и рис. 2.2. Количество двоичных разрядов целой части и дробной части может быть произвольным.

Алгоритм преобразования **целых** двоичных чисел (у которых все степени двойки – положительные) в целые десятичные числа из примера (2.2) можно записать в общем виде:

$$A_{10} = a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + ... + a_1 \cdot 2^1 + a_0 \cdot 2^0 = \sum_{i=0}^{n-1} a_i \cdot 2^i = \qquad (2.5)$$

или заменить равенством:

$$= (...(a_{n-1} \cdot 2 + a_{n-2}) \cdot 2 + ... + a_1) \cdot 2 + a_0, \qquad (2.6)$$

где: A_{10} – целое число в десятичной системе;
n – количество двоичных разрядов в целой части числа;
a_{n-1} – старший двоичный разряд;
a_0 – младший двоичный разряд.

Преимущества этого алгоритма (2.6) – все умножения **проводятся только на два:**

- умножить **старший** бинарный разряд $\boldsymbol{a_{n-1}}$ на 2;
- добавить к полученному произведению следующий бинарный разряд;
- умножить результат сложения на 2;
- добавить к полученному произведению следующий бинарный разряд;
- повторять две последние операции до тех пор, пока не будет добавлен последний младший разряд $\boldsymbol{a_0}$.

Пример использования алгоритма (2.6):

$$101011_2 = ((((1 \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 1 = 43_{10}.$$

Подробная запись вычислений:

$1 \cdot 2 = 2$; $2 + 0 = 2$; $2 \cdot 2 = 4$; $4 + 1 = 5$; $5 \cdot 2 = 10$;

$10 + 0 = 10;$ $10 \cdot 2 = 20;$ $20 + 1 = 21;$ $21 \cdot 2 = 42;$ $42 + 1 = 43.$

Алгоритм преобразования **дробных** двоичных чисел (у которых все степени двойки – отрицательные) – в дробные десятичные числа из примера (2.2) можно записать в общем виде:

$$A_{10} = a_1 \cdot 2^{-1} + a_2 \cdot 2^{-2} + ... + a_{m-1} \cdot 2^{-(m-1)} + a_m \cdot 2^{-m} = \sum_{i=1}^{m} a_i \cdot 2^{-i} = \qquad (2.7)$$

или заменить равенством:

$$= ((...(((a_m / 2) + a_{m-1}) / 2 + ... + a_2) / 2) + a_1) / 2, \qquad (2.8)$$

где: A_{10} – дробное число в десятичной системе;
m – количество двоичных разрядов в дробной части числа;
a_1 – старший дробный двоичный разряд;
a_m – младший дробный двоичный разряд.

Преимущества этого алгоритма (2.8) – все деления **проводятся только на два:**

- разделить **младший** бинарный разряд $\boldsymbol{a_m}$ на 2;
- добавить к полученному результату следующий (слева) двоичный разряд;
- разделить результат сложения на 2;
- повторять две последние операции до тех пор, пока не будет добавлен последний старший дробный разряд $\boldsymbol{a_1}$ и результат сложения поделен на 2.

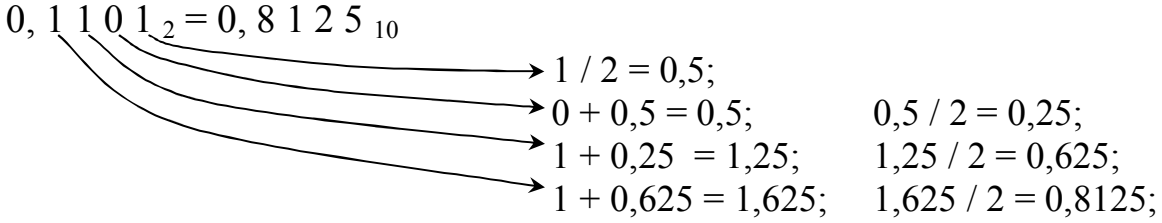


Рис. 2.3 – *Пример использования алгоритма* (2.8)

Обратите внимание – в алгоритмах (2.6) и (2.8) преобразование начинается с самого дальнего от двоичной запятой (точки) разряда, следующие разряды выбираются в направлении к бинарной запятой, и заканчивается преобразование разрядом, самым близким к двоичной запятой.

Преобразование восьмеричных чисел – в десятичные осуществляется в соответствии с примером (2.3). Алгоритм (2,6) для целой части и алгоритм (2.8) для дробной части можно применить и к восьмеричной системе, только все деления и умножения необходимо производить **на 8** – на основание системы.

Для преобразования шестнадцатеричных чисел – в десятичные применяют правило (2.4). Алгоритм (2,6) для целой части и алгоритм (2.8) для дробной части можно применить и к шестнадцатеричной системе, только все деления и умножения необходимо производить **на 16** – на основание системы.

При преобразовании десятичного числа – в двоичное – необходимо использовать различные правила для **целой части** числа и для **дробной части**.

Пример преобразования десятичного числа **157,465** показан на рис. 2.4 и рис. 2.5.

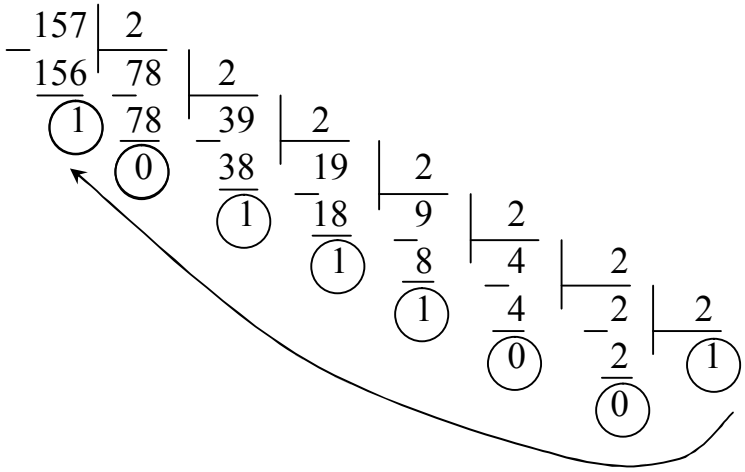


Рис. 2.4 – *Преобразование целой части десятичного числа в двоичное*

Для целой части двоичного числа используют правило **ДЕЛЕНИЯ на основание системы**, а остатки от деления (на рис. 2.4 обведены кружками) – это и будут двоичные биты преобразованного числа. Старший бит преобразованного числа – это последний остаток, а младший бит преобразованного числа – это самый первый остаток. Бинарное число записывается в направлении стрелки (рис. 2.4).

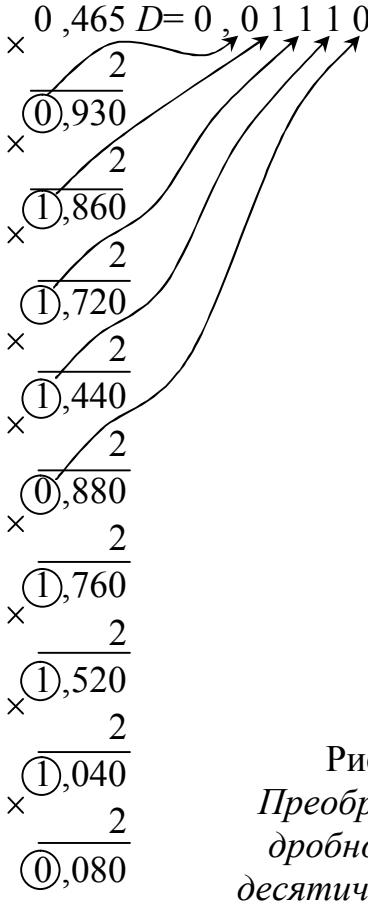


Рис. 2.5 *Преобразование дробной части десятичного числа*

По этому правилу целое десятичное число 157 имеет двоичный код:

$$157\ D = 10011101\ B$$

Проверка правильности преобразования целой части десятичного числа производится методом обратного преобразования по формуле (2.2) (или (2.6)):

$$10011101\ B = (1 \cdot 2^7) + (1 \cdot 2^4) +$$
$$+ (1 \cdot 2^3) + (1 \cdot 2^2) + (1 \cdot 2^0) =$$
$$= 128 + 16 + 8 + 4 + 1 = 157\ D$$

Для преобразования дробной части десятичного числа необходимо **УМНОЖАТЬ дробную часть на основание системы** (рис. 2.5).

Нули и единицы, которые выходят в целую часть произведений (на рис. 2.5 они обведены кружками), переносятся как дробные разряды преобразованного бинарного числа (на рис. 2.5 показаны стрелками).

Обратите внимание – во всех умножениях участвуют **только дробные разряды** десятичного числа.

Преобразование заканчивается, когда в дробной части останутся одни нули, или при достижении заданной точности преобразования (т.е. заданного количества бинарных разрядов в дробной части преобразованного числа).

Проверка правильности преобразования дробной части десятичного числа производится методом обратного преобразования по формуле (2.2) (или (2.8)):

$$0{,}011101110\ B = (1 \cdot 2^{-2}) + (1 \cdot 2^{-3}) + (1 \cdot 2^{-4}) + (1 \cdot 2^{-6}) + + (1 \cdot 2^{-7}) + (1 \cdot 2^{-8}) =$$

$$0{,}25 + 0{,}125 + 0{,}0625 + 0{,}015625 + 0{,}0078125 + 0{,}00390625 = 0{,}46484375\ D$$

Полученный результат немного меньше, чем дробная часть исходного десятичного числа – **0,465**. Это объясняется неполным преобразованием дробной части на рис. 2.5. Если продолжать умножения и получить результат с большим количеством дробных бинарных разрядов, то преобразованное бинарное число будет точнее приближаться к дробной части исходного десятичного числа.

На рис. 2.6 показан еще один алгоритм перевода целых десятичных чисел – в двоичные. Для успешного применения этого метода необходимо выучить степени числа два. В таблице 2.1 приведены степени числа два до 10-й степени.

Таблица 2.1 – *Степени числа 2*

Степень	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Число	1	2	4	8	16	32	64	128	256	512	1024

Преобразование начинается с подбора максимально возможной степени числа 2, которая помещается в десятичное число – **157**. В нашем примере максимальная степень – «128» вычитается из преобразуемого числа, а в результирующее бинарное число записывается старший бит – «1».

В полученный остаток «29» не помещается следующая степень числа 2 – «64», поэтому в следующий разряд бинарного числа записывается нуль.

Поочередное вычитание **всех степеней**, которые помещаются в остаток, – приводит к записи в соответствующий разряд результирующего бинарного числа – единицы. Если очередная степень не помещается в остаток, – в очередной бинарный разряд записывается нуль (см. рис. 2.6).

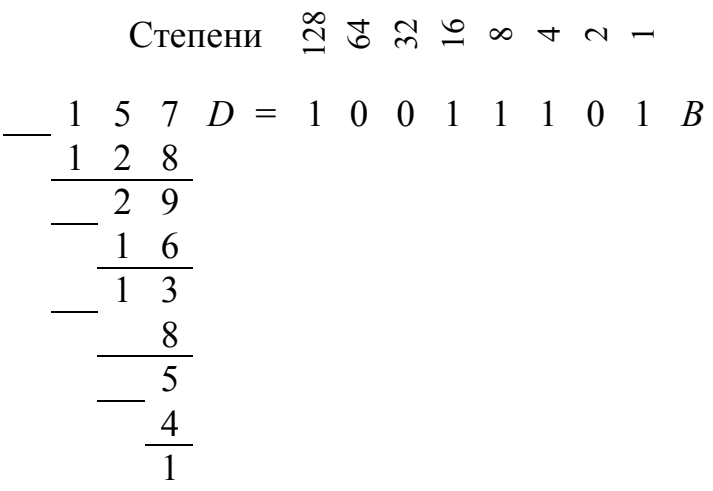


Рис. 2.6 – *Преобразование целой части десятичного числа в двоичное*

Проверка результата осуществляется в обратном порядке – сложением степеней числа 2, которым соответствуют единицы в двоичном числе. Этот метод взаимно обратный алгоритму перевода двоичных чисел в десятичные (см. пример (2.2)).

Аналогичные правила можно применить для перевода десятичного числа **157,465** в **восьмеричную систему** счисления. Целую часть получаем в виде остатков от **деления** этого числа на 8 – на основание системы (рис. 2.7):

157 *D* = 235 *Q*.

Дробная часть преобразовывается **умножением** дробной части десятичного числа на 8 – на основание системы (рис. 2.8):

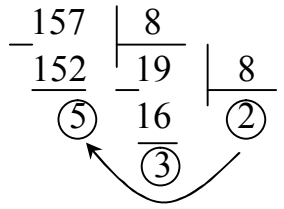


Рис. 2.7 – *Преобразование целой части в восьмеричную систему*

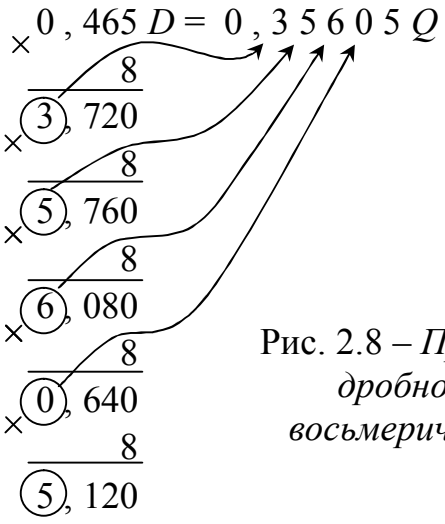


Рис. 2.8 – *Преобразование дробной части в восьмеричную систему*

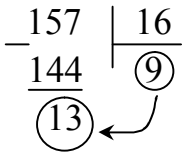


Рис. 2.9 – *Преобразование целой части в шестнадцатеричную систему*

Аналогично десятичное число **157,465** переведем в **шестнадцатеричную систему** (см. рис. 2.9, и рис. 2.10):

157 *D* = 9D *H*

Очень легко переводить числа из восьмеричной системы или из шестнадцатеричной системы счисления – в двоичную.

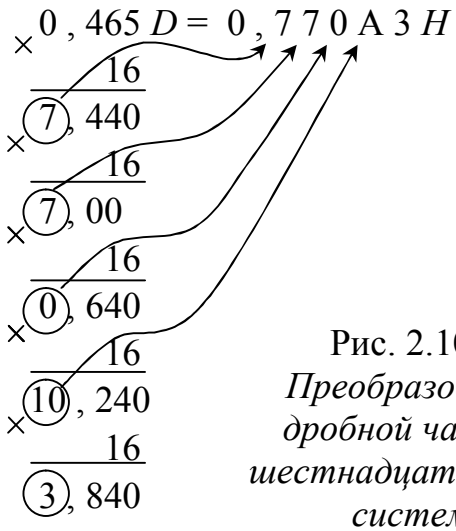


Рис. 2.10 – *Преобразование дробной части в шестнадцатеричную систему*

Для перевода числа из восьмеричной системы счисления в двоичную – необходимо каждый восьмеричный символ заменить тремя битами (триадами) с весами 4-2-1 (см. таблицу 2.2).

Ранее преобразованное десятичное число **157,465** *D* в восьмеричную систему счисления (см. рис. 2.7 и рис. 2.8) можно представить в двоичной системе:

$$2\ \ 3\ \ 5\ ,\ 3\ \ 5\ \ 6\ \ 0\ \ 5\ \ Q =$$

$$= 010\ 011\ 101\ ,\ 011\ 101\ 110\ 000\ 101\ B \qquad (2.9)$$

Таблица 2.2 – *Перевод восьмеричных чисел в двоичный код с весами 4-2-1*

Восьм. число	Двоичн. число		
	4	2	1
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Аналогично для перевода чисел из шестнадцатеричной системы счисления в двоичную – необходимо каждый шестнадцатеричный символ (см. пример на рис. 2.9 и рис. 2.10) заменить на четыре бита (тетраду) с весами 8-4-2-1 в соответствии с таблицей 2.3:

$$9\ \ D\ ,\ 7\ \ 7\ \ 0\ \ A\ \ 3\ \ H =$$

$$= 1001\ 1101\ ,\ 0111\ 0111\ 0000\ 1010\ 0011\ B \qquad (2.10)$$

Обратите внимание на то, что целая и дробная часть двоичного числа в примерах (2.9) и (2.10) совпадают с преобразованием в двоичную систему счисления десятичного числа **157,465** *D* на рис. 2.4 и рис. 2.5.

Поэтому можно рекомендовать для перевода десятичных чисел в двоичный формат – сначала преобразовать это число в восьмеричную или шестнадцатеричную систему счисления (см. рис. 2.7, рис. 2.8, или рис. 2.9, рис. 2.10), а потом заменить каждый символ на двоичный эквивалент в соответствии с табл. 2.2 или табл. 2.3.

Также очень легко преобразовывать бинарные числа в восьмеричную или в шестнадцатеричную систему. Для этого надо разделить двоичное число, **начиная от двоичной точки** (или двоичной запятой) на группы по 3 бита (триады) для восьмеричной системы (см. последнюю строку в при-

Таблица 2.3 – *Перевод шестнадцатеричных чисел в двоичный код с весами 8-4-2-1*

Дес. число	Шестн. число	Двоичн.число			
		8	4	2	1
0	0	0	0	0	0
1	1	0	0	0	1
2	2	0	0	1	0
3	3	0	0	1	1
4	4	0	1	0	0
5	5	0	1	0	1
6	6	0	1	1	0
7	7	0	1	1	1
8	8	1	0	0	0
9	9	1	0	0	1
10	A	1	0	1	0
11	B	1	0	1	1
12	C	1	1	0	0
13	D	1	1	0	1
14	E	1	1	1	0
15	F	1	1	1	1

мере (2.9)) или на группы по 4 бита (тетрады) для шестнадцатеричной системы (см. последнюю строку в примере (2.10)). После этого каждая двоичная **группа битов** заменяется на соответствующий **символ** из табл. 2.2 или из табл. 2.3 (см. первые строки в примерах (2.9) или (2.10)).

Внутри компьютера все числа хранятся и обрабатываются **только в виде нулей** и **единиц** – в двоичной системе счисления. Для просмотра содержимого ячеек памяти или регистров очень неудобно выводить на экран длинные цепочки нулей и единиц. Для сокращения записи **бинарных чисел** – они выводятся на экран в виде **шестнадцатеричного эквивалента**. Можно сказать, что шестнадцатеричный формат – это компактная форма представления двоичных чисел.

Поэтому студентам необходимо выучить таблицу 2.3. При этом не нужно запоминать код каждого числа. Нужно запомнить (как в школе) из каких степеней двойки состоят числа от 0 до 15.

Например: число $3 = 2 + 1 = 2^1 + 2^0$, поэтому код числа $3_{10} = 11_2$. В табл. 2.3 числу 3_{10} соответствует код 0011_2. Вспомните школьные навыки: если к любому положительному числу в целой части дописать слева произвольное количество нулей, – то значение числа от этого не изменится.

Число $5 = 4 + 1$, поэтому двоичный код числа 5_{10} равен 101_2 или 0101_2. Число $7 = 4 + 2 + 1$, поэтому двоичный код числа 7_{10} равен 111_2 или 0111_2. Число $13 = 8 + 4 + 1 = 2^3 + 2^2 + 2^0$, поэтому двоичный код числа $13_{10} = D_{16}$ равен 1101_2.

Для перевода целых чисел из одной системы счисления в любую другую можно использовать программу «Инженерный калькулятор» (calc.exe) из набора стандартных программ WINDOWS.

Сначала необходимо выбрать систему счисления исходного числа, ввести целое число и кликнуть мышкой кнопку нужной системы счисления. На рис. 2.11 показано перевод числа 157 из десятичной системы счисления в шестнадцатеричную систему, в восьмеричную систему и двоичную систему.

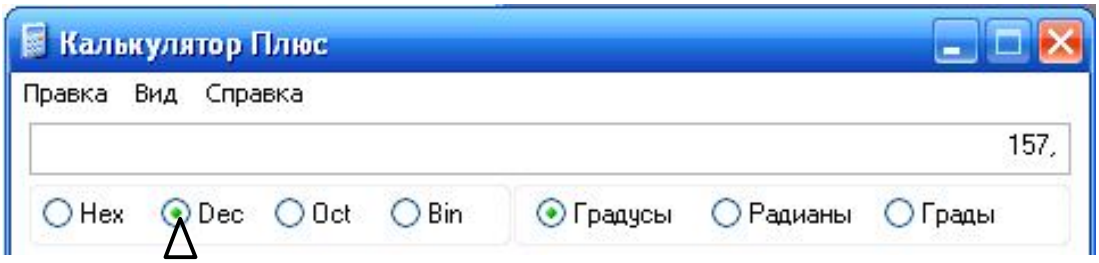


Рис. 2.11 – *Число 157 в десятичной системе счисления*

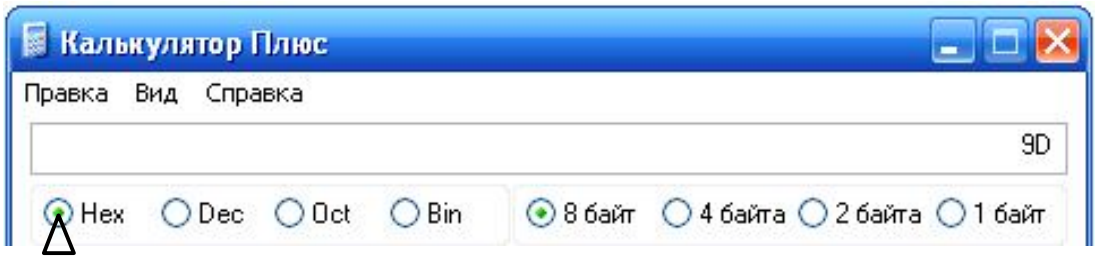


Рис. 2.11-1 – *Число* 157_{10} *в шестнадцатеричной системе счисления*

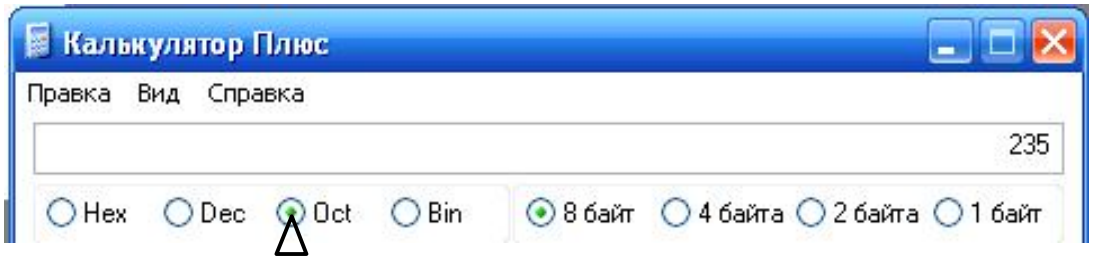


Рис. 2.11-2 – *Число* 157_{10} *в восьмеричной системе счисления*

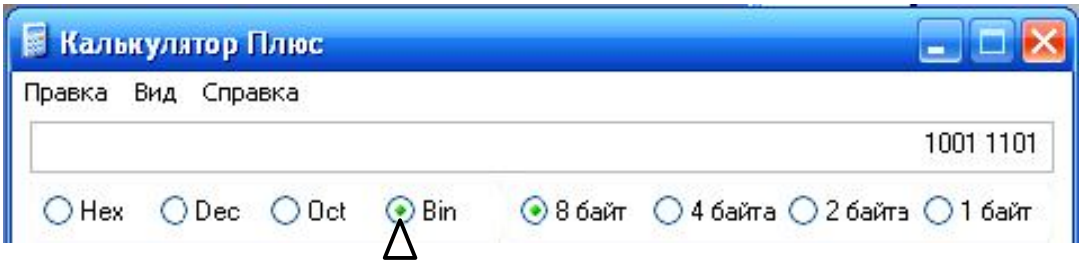


Рис. 2.11-3 – *Число* 157_{10} *в двоичной системе счисления*

2.1.2 Диапазон представимых чисел

В десятичной системе счисления одним десятичным разрядом можно записать 10 цифр – от 0 до 9. Двумя десятичными разрядами можно представить 100 цифр – от 0 до 99. Поэтому **диапазон представимых чисел** (N) определяется **основанием** системы и **количеством разрядов** числа (n):

$$N = 10^n.$$

Для двоичной системы счисления диапазон представимых чисел:

$$N = 2^n.$$

Для любой системы счисления диапазон представимых чисел:

$$N = A^n, \qquad (2.11)$$

где: A – основание системы счисления.

Двоичные числа размерностью – 1 Байт (8 бит) могут принимать значения от $0_{10} = 0000\ 0000_2$ до $255_{10} = (2^8 - 1)_{10} = 1111\ 1111_2 = FF_{16}$. Поэтому одним байтом можно закодировать $2^8 = 256$ различных чисел.

Двумя байтами (или одним «словом» – 16 бит) можно закодировать числа в диапазоне от $0_{10} = 0000\ 0000\ 0000\ 0000_2$ до $65\ 535_{10} = (2^{16} - 1)_{10} = 1111\ 1111\ 1111\ 1111_2 = FFFF_{16}$.

«Двойное слово» (четыре байта – 32 бита) позволяет записать числа от нуля до:

$$4\ 294\ 967\ 295_{10} = (2^{32} - 1)_{10} = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = FF\ FF\ FF\ FF_{16}.$$

Большие числа в компьютерной литературе обозначаются теми же приставками (Кило, Мега, Гига и т.д.), что и десятичные числа, но имеют немного другое числовое значение (равное числу 2 в степени, кратной 10). Обычное десятичное число 1 километр = 10^3 метров = 1000 метров. В компьютерной литературе – 1 КилоБайт = 2^{10} Байтов = 1024 Байт.

В таблице 2.4 приведены точные и приблизительные значения степеней числа 2 и их обозначение в компьютерной литературе.

Таблица 2.4 – *Значения степеней числа 2*

Приставка	Степень 2	Точное значение	Приблиз. значение
К – Кило	2^{10}	1 024	$\approx 10^3$
М – Мега	2^{20}	1 048 576	$\approx 10^6$
G – Гига	2^{30}	1 073 741 824	$\approx 10^9$
Т – Терра	2^{40}	1 099 511 627 776	$\approx 10^{12}$
Р – Пета	2^{50}	1 125 899 906 842 624	$\approx 10^{15}$
Е – Экса	2^{60}	1 152 921 504 606 846 976	$\approx 10^{18}$

Такая привязка к степеням числа 2 позволяет очень просто вычислять двоичные логарифмы или возводить число 2 в целую степень. Для этого студентам необходимо выучить таблицу 2.1 и первые два столбца таблицы 2.4.

$$2^{16} = 2^{10+6} = 2^{10} \cdot 2^6 = 64К;$$

$$2^{24} = 2^{20+4} = 2^{20} \cdot 2^4 = 16М;$$

$2^{32} = 2^{30+2} = 2^{30} \cdot 2^2 = 4G.$

Log (16K) = log (16) + log (2^{10}) = 4 + 10 = 14;

Log (32M) = log (32) + log (2^{20}) = 5 + 20 = 25;

Log (64G) = log (64) + log (2^{30}) = 6 + 30 = 36.

В таблице 2.5 приведены различные названия для двоичных и десятичных чисел в соответствии с международными стандартами.

Таблица 2.5 – *Названия для двоичных и десятичных чисел*

Десятичная приставка			Двоичная приставка			
Название	Символ	Степень	Название	Символ		Степень
килобайт	kB	10^3	кибибайт	KiB	Кбайт	2^{10}
мегабайт	MB	10^6	мебибайт	MiB	Мбайт	2^{20}
гигабайт	GB	10^9	гигибайт	GiB	Гбайт	2^{30}
терабайт	TB	10^{12}	тебибайт	TiB	Тбайт	2^{40}
петабайт	PB	10^{15}	пебибайт	PiB	Пбайт	2^{50}
эксабайт	EB	10^{18}	эксбибайт	EiB	Эбайт	2^{60}
зеттабайт	ZB	10^{21}	зебибайт	ZiB	Збайт	2^{70}
йоттабайт	YB	10^{24}	йобибайт	YiB	Йбайт	2^{80}

2.1.3 Специальное двоичное кодирование

В современных компьютерах используют также смешанную систему счисления – **ДВОИЧНО-ДЕСЯТИЧНЫЙ КОД** (Binary-Coded-Decimal – BCD), которая объединяет черты как десятичной, так и двоичной системы.

Если каждый разряд десятичного числа представить двоичным эквивалентом (по четыре бита – смотрите первые десять строк таблицы 2.3), то в результате получаем двоично-десятичный код.

Десятичное число 925_{10} имеет двоично-десятичный код:

9 2 5_{10} *Десятичный код*

↓ ↓ ↓

1 0 0 1 0 0 1 0 0 1 0 $1_{2\text{-}10}$ *Двоично-десятичный код*

Важно помнить, что двоично-десятичный код не является еще одной системой счисления. Это фактически та же десятичная система, в которой каждый десятичный символ представлен двоичным эквивалентом.

Главное преимущество двоично-десятичного кода – это простота преобразования в десятичную систему и обратно. **Главный недостаток** – это значительно меньший диапазон представимых чисел, чем в двоичной системе (при одинаковом количестве двоичных разрядов).

В одном байте можно закодировать 256 различных двоичных чисел – от нуля до 255. Диапазон представимых двоично-десятичных чисел в одном байте (8 бит или 2 тетрады) – от нуля до 99 (т.е. в два с половиной раза меньше).

Двоичные числа размером «слово» (2 байта или 16 бит) позволяют записывать числа от нуля до 65 535. Диапазон представимых двоично-десятичных чисел в одном «слове» от нуля до 9 999 (т.е. в шесть с половиной раза меньше).

Арифметические операции над двоично-десятичными числами требуют дополнительных команд и, в общем случае, **выполняются медленнее**, чем над двоичными числами.

Большой интерес представляет также «**код ГРЕЯ**». Главная особенность этого кода – соседние двоичные комбинации **отличаются только в одном бите**. Для преобразования двоичного позиционного кода в код Грея необходимо выполнить операцию «сложение по модулю 2» (XOR – «ИСКЛЮЧАЮЩЕЕ ИЛИ») над всеми соседними битами в двоичном числе (на рис. 2.12 представлены четырехбитовые числа, у которых соседние биты объединяются операцией XOR – показано стрелками).

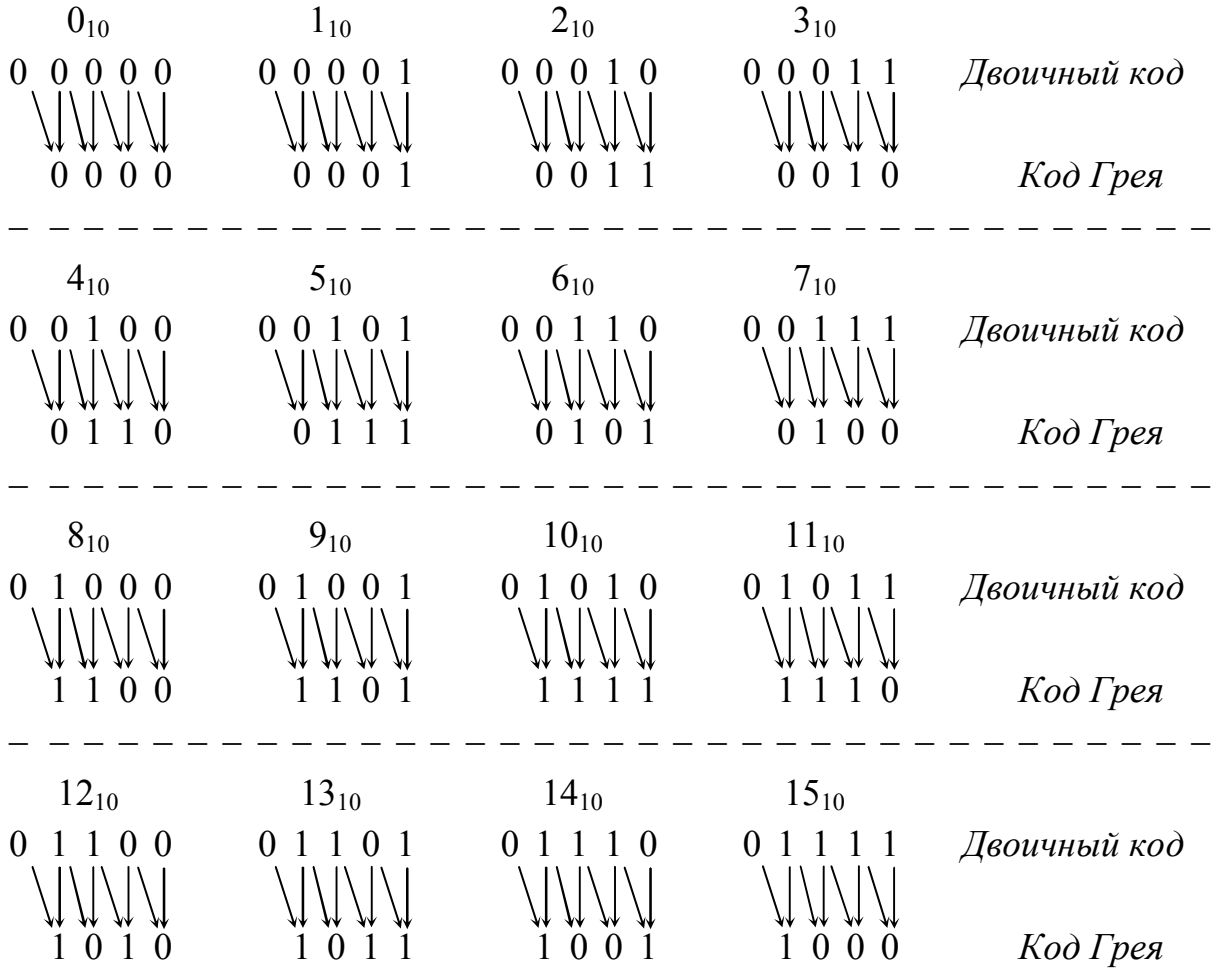


Рис. 2.12 – *Перевод двоичных чисел в код Грея*

Результат выполнения операции «ИСКЛЮЧАЮЩЕЕ ИЛИ» равен нулю, если два бита одинаковые, и равен единице, – если два бита противоположные. Как уже упоминалось ранее: слева от положительного двоичного числа можно дописать любое количество нулей (см. рис. 2.12).

Код Грея может содержать любое количество битов от двух и более. **Обратите внимание** – крайние комбинации кода Грея (нулевая и последняя) также отличаются только в одном бите.

2.1.4 Коды положительных и отрицательных двоичных чисел

В цифровых компьютерах хранятся и обрабатываются бинарные положительные и отрицательные числа. Поэтому кроме значащих двоичных символов необходимо кодировать и знак числа (который записывается в крайнюю левую позицию). Во всех современных ЭВМ знак «плюс» кодируется нулем, а знак «минус» кодируется единицей.

Число +157 *D* записывается в двоичном коде 0 1001 1101 *B*.
Число –157 *D* записывается в двоичном коде 1 1001 1101 *B*.
Такая запись называется – **ПРЯМОЙ КОД**.

Однако наиболее часто для хранения и обработки положительных и отрицательных чисел в компьютерах используют **ДОПОЛНИТЕЛЬНЫЙ КОД**. Дополнительный код положительного числа совпадает с его прямым кодом. Дополнительный код отрицательного числа получают инверсией (заменой на противоположные) **всех значащих битов** и прибавлением единицы.

Для перевода числа «–157» в дополнительный код – его записывают в прямом коде. Инвертируют **значащие биты** и добавляют единицу:

Знаковый разряд у отрицательных чисел в дополнительном коде **всегда равен 1** (см. рис. 2.13).

1	1001 1101	*Прямой код*
	↓ ↓	
1	0110 0010	*Инверсия*
+	1	*Добавление* 1
1	0110 0011	*Дополнит. код*

Рис. 2.13 – *Преобразование в дополнительный код*

Если у положительных чисел можно записать слева любое количество **нулей** (т.е. знаков +), то у отрицательных чисел в дополнительном коде можно записать слева любое количество **единиц** (т.е. знаков «минус»). Это называется – **знаковым расширением чисел в дополнительном коде**.

Четырьмя двоичными битами можно записать $2^4 = 16$ положительных чисел от нуля до $15_{10} = 1111_2$ (см. внешний круг на рис.. 2.14).

При записи чисел в дополнительном коде крайний левый бит кодирует знак. Поэтому в дополнительном коде четырьмя двоичными битами (один из них знаковый) можно записать 8 отрицательных чисел от «–8» до

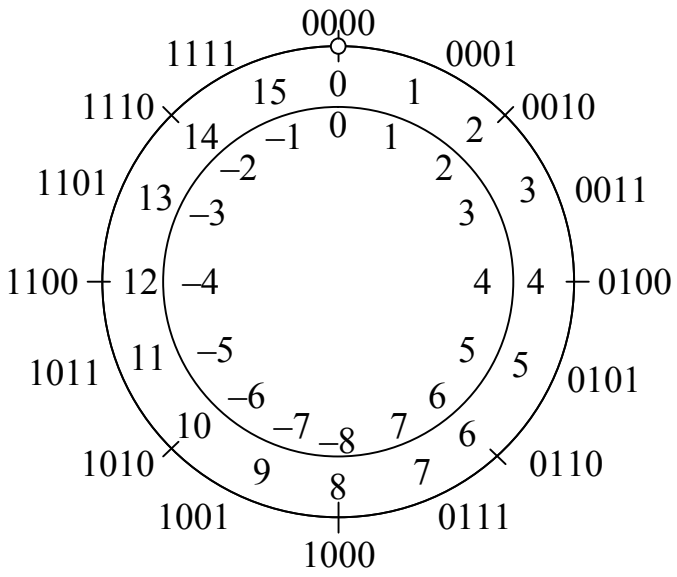


Рис. 2.14 – *Прямой и дополнительный коды четырехбитовых чисел*

«–1» и 8 положительных чисел от 0 до 7 (см. внутренний круг на рис. 2.14). Общее количество представимых чисел в дополнительном коде также равно: $2^4 = 16 = 8 + 8$.

Диапазон представимых чисел для одного байта (8 бит) равен $2^8 = 256$. Для положительных чисел в прямом коде это числа от нуля до $255_{10} = 1111\ 1111_2$.

На рис. 2.15 показана числовая ось и диапазоны представимых чисел в одном байте для беззнаковых положительных чисел (от 0 до 255) и для чисел со знаком в дополнительном коде (от –128 до +127). В обоих случаях диапазон представимых чисел: $N = 2^8 = 256$.

Обратите внимание – диапазон представимых чисел в дополнительном коде сдвинут влево – в область отрицательных чисел симметрично относительно нуля по сравнению с беззнаковыми числами.

Рис. 2.15 – *Диапазоны представимых байтовых положительных беззнаковых чисел и чисел в дополнительном коде*

Для двоичных чисел размером «слово» (2 байта или 16 бит) диапазон представления чисел в дополнительном коде от –32 768 до +32 767 сдвинут симметрично относительно нуля по сравнению с диапазоном положительных беззнаковых чисел – от нуля до 65 535.

Аналогично для бинарных чисел размером «двойное слово» (4 байта или 32 бита) диапазон представимых чисел в дополнительном коде от – 2 147 483 647 до +2 147 483 648 сдвинут симметрично относительно нуля по сравнению с диапазоном положительных беззнаковых чисел – от нуля до 4 294 967 296.

Числа +157 и –157 нельзя записать **в дополнительном коде** размером в один байт – 8 бит, потому что они выходят за диапазон представимых чисел (см. рис. 2.15). Эти числа можно записать **в дополнительном коде** со знаковым расширением в формате «слово» – 16 бит:

+157 *D* = 0000 0000 1001 1101 *B*;
–157 *D* = 1111 1111 0110 0011 *B*;

или в формате «двойное слово» – 32 бита:

+157 *D* = 0000 0000 0000 0000 0000 0000 1001 1101 *B*;
–157 *D* = 1111 1111 1111 1111 1111 1111 0110 0011 *B*.

2.2 АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ В ДВОИЧНОЙ СИСТЕМЕ СЧИСЛЕНИЯ

2.2.1 Сложение двоичных чисел

Сложение двоичных чисел осуществляется даже проще, чем десятичных чисел, потому что таблица сложения для одноразрядных двоичных чисел (см. табл. 2.6) значительно меньше, чем аналогичная таблица для десятичных чисел.

Таблица 2.6 – *Таблица сложения двоичных чисел*

A	B	P	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Обозначения в таблице 2.6:
A – первое битовое слагаемое;
B – второе битовое слагаемое;
S – сумма;
P – бит переполнения.

Уже в этом примере сложения простейших двоичных чисел получен результат, размер которого на один бит больше, чем исходные однобитовые операнды (см. последнюю строку табл. 2.6):

$$1_2 + 1_2 = 10_2.$$

Как и при сложении десятичных чисел, этот дополнительный двоичный разряд называется **переполнением** (P) и складывается с последующими (более старшими) разрядами двоичных чисел.

Поэтому полная таблица сложения однобитовых двоичных чисел должна включать и **бит переноса** (C) от предыдущего младшего разряда (см. табл. 2.7).

Таблица 2.7 – *Таблица сложения двоичных чисел*

C	A	B	P	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

С учетом табл. 2.7 сложим два четырехбитовых числа:

$A = 7_{10} = 0111_2$ и $B = 14_{10} = 1110_2$.

Десятичные		*Двоичные*		
1		1 1 1		*Перенос*
+ 7		+ 0 1 1 1	*A*	
1 4		1 1 1 0	*B*	
2 1		1 0 1 0 1	*S*	

Результат сложения (S) получился на один бит длиннее, чем исходные четырехбитовые операнды A и B.

Аналогично по правилам таблицы 2.7 можно складывать двоичные числа любой разрядности, например байтовые операнды: $A = 157_{10} = 1001\ 1101_2$ и $B = 85_{10} = 0101\ 0101_2$.

Десятичные		*Двоичные*		
1 1		1 1 1 1		*Перенос*
+ 1 5 7		+ 1 0 0 1 1 1 0 1	*A*	
8 5		0 1 0 1 0 1 0 1	*B*	
2 4 2		1 1 1 1 0 0 1 0	*S*	(2.12)

2.2.2 Вычитание двоичных чисел

Операция вычитания в современных компьютерах выполняется как алгебраическое сложение положительных и отрицательных чисел в **МОДИФИЦИРОВАННОМ ДОПОЛНИТЕЛЬНОМ КОДЕ**.

Количество разрядов двоичного сумматора в цифровых компьютерах на один бит превышает разрядность обрабатываемых операндов. Числа, которые хранятся в памяти в **дополнительном коде**, записываются в регистры сумматора в **модифицированном дополнительном коде** – с двумя одинаковыми знаковыми разрядами. Этот код позволяет обнаруживать переполнение в результирующем операнде.

Рассмотрим примеры алгебраического сложения двоичных положительных чисел (для положительных чисел прямой и дополнительный коды совпадают) и отрицательных чисел в модифицированном дополнительном коде (в этом коде два одинаковых старших бита – это знак числа):

$+105_{10} = 0110\ 1001_2$;	$+83_{10} = 0101\ 0011_2$;	*Прямой код*
$1001\ 0110_2$;	$1010\ 1100_2$;	*Инверсия*
+ 1	+ 1	*Добавление* 1
$-105_{10} = 1001\ 0111_2$;	$-83_{10} = 1010\ 1101_2$;	*Дополнит. Код*
$-105_{10} = 1\ 1001\ 0111_2$;	$-83_{10} = 1\ 1010\ 1101_2$;	*Модиф. дополн. Код*

Все числа (т.е. исходные операнды) попадают в диапазон представимых чисел для байтовых операндов в дополнительном коде (см. рис. 2.15).

Десятичные	*Двоичные*	
	1 1 1 1 1 1	*Перенос*
+ 105	+ 0 0 1 1 0 1 0 0 1	*A*
– 83	1 1 0 1 0 1 1 0 1	*B*
+ 22	⓪ ⓪ 0 0 1 0 1 1 0	*S*
	CF SF	

(2.13)

В примере (2.13) последний левый перенос после второго знакового бита – теряется (для него нет разрядов в сумматоре). Полученный результат – это положительное число: $0\ 0001\ 0110_2 = 16 + 4 + 2 = 22_{10}$, так как два старших знаковых бита – равны нулю:

Значение старшего знакового бита записывается в специальную ячейку – **CARRY** Flag – **CF** – флаг переполнения.

Младший знаковый бит результата копируется в специальную ячейку – **SIGN** Flag – **SF** – знаковый флаг.

Еще в одной ячейке формируется – **OVERFLOW** Flag – **OF**– флаг арифметического переполнения. Значение этого флага определяется операцией XOR – «ИСКЛЮЧАЮЩЕЕ ИЛИ» над флагами CF и SF. Если два старших знаковых бита результата (т.е. флаги CF и SF) одинаковые, – то формируется OF = 0. Если эти флаги различные, – формируется OF = 1. Единичное значение этого флага показывает – результат арифметической операции вышел за диапазон представимых чисел в дополнительном коде.

Десятичные	*Двоичные*	
	1 1 1	*Перенос*
+ 83	+ 0 0 1 0 1 0 0 1 1	*A*
– 105	1 1 0 0 1 0 1 1 1	*B*
– 22	① ① 1 1 0 1 0 1 0	*S*
	CF SF	

(2.14)

Полученный результат – это отрицательное число в дополнительном коде, так как два старших знаковых бита – равны единице:

Дополн. Код *Инверсия* *Добавлен.* 1

1 110 1010 > 1 001 0101 > $1\ 001\ 0110_2 = -22_{10}$.

Во флаг – CARRY Flag – CF записана 1. Это означает, что из меньшего числа «83» вычли большее число «105». По правилам вычитания – **необходим заем.**

Если при сложении CF = 1 – это переполнение (т.е. результат больше представимого числа), то при вычитании CF = 1 – это заем (т.е. результат меньше представимых положительных чисел – меньше нуля).

Десятичные	*Двоичные*	
	1 1 1	*Перенос*
+ + 83	+ 0 0 1 0 1 0 0 1 1	*A*
+105	0 0 1 1 0 1 0 0 1	*B*
+188	0 1 0 1 1 1 1 0 0	*S*
	CF SF	

(2.15)

В примере (2.15) знаковый флаг SF =1, хотя результат в десятичном формате: +188 – положительный. **Обратите внимание**: флаги CF и SF – противоположные, поэтому флаг OF = 1, а это означает, что результат вышел за диапазон представимых байтовых чисел **в дополнительном коде** (см. рис. 2.15): результат +188 > +127. Если рассматривать результат, как сумму положительных беззнаковых чисел, то число +188 попадает в диапазон представимых байтовых чисел: от 0 до 255. Поэтому флаг CF = 0, т.е. нет переполнения для беззнаковых чисел.

Десятичные	*Двоичные*	
	1 1 1 1 1 1 1 1 1	*Перенос*
+ +105	+ 0 0 1 1 0 1 0 0 1	*A*
–105	1 1 0 0 1 0 1 1 1	*B*
0	⓪⓪ 0 0 0 0 0 0 0 0	*S*
	CF SF	

(2.16)

Результат вычитания одинаковых чисел равен нулю. Поэтому в специальной ячейке – **ZERO** Flag – **ZF** – флаг нулевого результата – устанавливается единица. Если результат операции отличен от нуля, то ZF = 0.

ВЫВОДЫ:

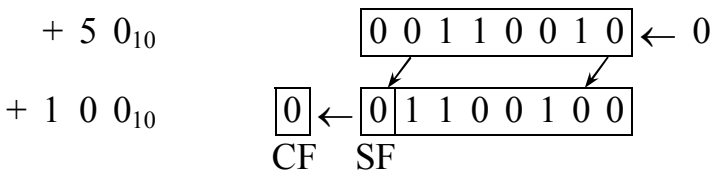
- Если результат операции алгебраического сложения вышел за диапазон представимых положительных беззнаковых чисел (для байтовых операндов – этот диапазон: от 0 до 255), то устанавливается в единицу флаг переполнения: CF = 1.
- Если результат операции алгебраического сложения вышел за диапазон представимых чисел в дополнительном коде (для байтовых операндов – этот диапазон: от –128 до +127), то устанавливается в единицу флаг арифметического переполнения: OF = 1.

- Операция сравнения операндов (comported) выполняет вычитание – из первого числа вычитается второй операнд (но без записи результата вычитания) и выставляет флаги – CF, OF и ZF:
  - если первый положительный беззнаковый операнд больше второго, то: CF = 0 (при вычитании не было заема);
  - если второй положительный беззнаковый операнд больше первого, то: CF = 1 (был заем при вычитании);
  - если первый операнд в дополнительном коде больше второго, то OF = 0;
  - если второй операнд в дополнительном коде больше первого, то OF = 1;
  - если оба операнда одинаковые, то: ZF = 1 (результат вычитания – нулевой).

2.2.3 Команды сдвигов

С операциями умножения и деления тесно связаны **КОМАНДЫ СДВИГОВ**. Сдвиг двоичных операндов можно производить вправо или влево, на один двоичный разряд или на несколько разрядов (но не более длины операнда).

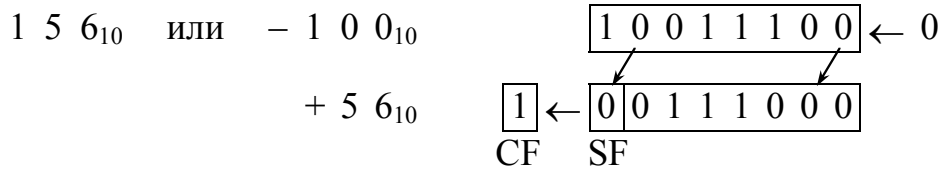
Для **удвоения** беззнакового положительного операнда или операнда в дополнительном коде нужно сдвинуть все биты на один разряд **влево**. В освобождаемые младшие разряды «вдвигаются» нули:



(2.17)

Сдвиг влево на несколько двоичных разрядов позволяет умножить число на $\mathbf{2^n}$, где n – количество разрядов сдвига. Последний выдвигаемый влево двоичный разряд сохраняется во флаге CF.

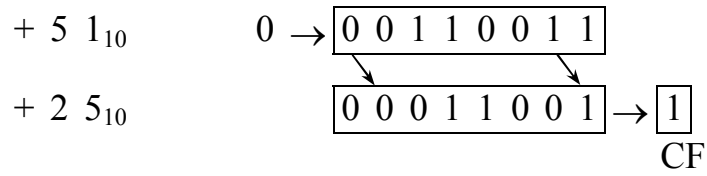
При сдвиге влево на один бит беззнаковых положительных операндов – флаг CF = 1 индицирует выход результата за диапазон представимых значений для беззнаковых положительных чисел. В примере (2.18) **исходный операнд** может быть положительным беззнаковым числом: 156 или отрицательным числом в дополнительном коде: –100.



(2.18)

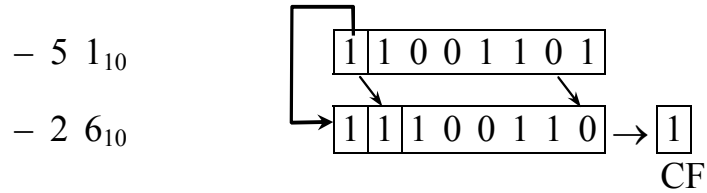
При сдвиге влево на один бит операндов в дополнительном коде – флаг OF, равный операции XOR над флагами CF и SF, индицирует выход результата за диапазон представимых значений в дополнительном коде (в примере (2.18) флаг OF = 1).

Сдвиг **вправо** беззнакового положительного операнда **делит** его на $\mathbf{2^n}$, где n – количество разрядов сдвига. В освобождаемые старшие разряды «вдвигаются» нули. Последний выдвигаемый вправо двоичный разряд сохраняется во флаге CF.



(2.19)

Сдвиг **вправо** положительных или отрицательных операндов в дополнительном коде также **делит** его на $\mathbf{2^n}$**.** Для этого используют специальную команду – **арифметический сдвиг вправо**. Главная отличительная особенность этой команды – в освобождаемые старшие разряды копируется старший (знаковый) бит исходного операнда.



(2.20)

В командах сдвигов вправо **не может быть переполнения**, т.е. результат никогда не выйдет за диапазон представимых чисел.

При сдвиге вправо во флаг CF записывается половина младшего разряда результирующего операнда. Для округления результата рекомендуется добавить значение флага CF к сдвинутому операнду.

2.3.4 Умножение двоичных чисел

Умножение реализуется по обычным правилам арифметики согласно таблице двоичного умножения (см. табл. 2.8).

Таблица 2.8 – *Таблица умножения двоичных чисел*

A	B	P
0	0	0
0	1	0
1	0	0
1	1	1

Обозначения в таблице 2.8:
A – первое битовое множимое;
B – второе битовое множимое;
P – произведение;

В компьютерах может быть реализовано как умножение, которое начинается с младшего бита множителя (наиболее при-

вычный способ), так и умножение, начинающееся со старшего бита множителя (см. рис. 2.16).

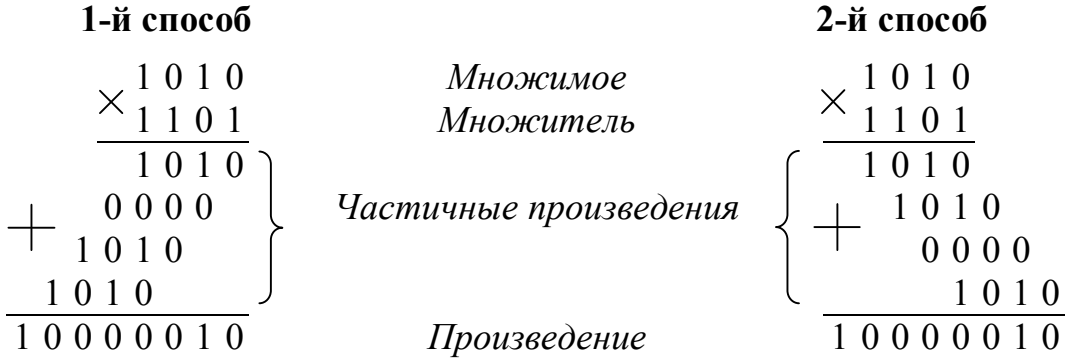


Рис. 2.16 – *Умножение двоичных чисел*

Каждое частичное произведение представляет собой либо нуль, либо множимое, сдвинутое на соответствующее количество разрядов влево или вправо в зависимости от способа умножения. Таким образом, умножение в двоичной системе сводится к двум операциям: сложению и сдвигу чисел.

Обычный сумматор рассчитан на одновременное сложение только **двух операндов**. Если нужно получить сумму нескольких частичных произведений, то происходит накопление суммы: сначала в сумматор записывают первое частичное произведение, к нему прибавляется второе, затем к полученной сумме добавляется третье и так продолжается до получения полной суммы (произведения).

Описанный принцип накопления позволяет реализовать 4 алгоритма умножения двоичных чисел:

- сдвигом множимого на требуемое количество разрядов влево (при **первом алгоритме**) или вправо (при **втором алгоритме**) и добавление полученного таким образом очередного частичного произведения к ранее накопленной неподвижной сумме (см. рис. 2.16);
- сдвигом суммы ранее полученных частичных произведений на каждом шаге на один разряд вправо (при **третьем алгоритме**) или влево (при **четвертом алгоритме**) и последующем добавлении к сдвинутой двоичной комбинации несдвинутого множимого, если анализируемая цифра множителя равна единице.

Обратите внимание – количество значащих разрядов в произведении (как и при умножении в других системах счисления) равно **сумме** количеств значащих разрядов в обоих сомножителях.

Умножение отрицательных и положительных чисел, которые хранятся в памяти или регистрах в дополнительном коде, реализуется в три этапа:

1. определяется знак произведения с помощью сложения знаковых (старших) битов сомножителей «по модулю 2» (операция XOR – «ИСКЛЮЧАЮЩЕЕ ИЛИ»);
2. перевод значащих битов отрицательных операндов из дополнительного кода – в прямой код и умножение положительных чисел одним из четырех ранее описанных алгоритмов;
3. перевод полученного произведения в дополнительный код с учетом знака, определенного на первом этапе.

2.3.5 Деление двоичных чисел

Деление двоичных чисел осуществляется по таким же правилам, как и деление десятичных чисел в столбик. Из делимого (начиная со старшей цифры) вычитают **произведение** старшей цифры частного на весь делитель. Но, поскольку в двоичной системе цифрами частного могут быть только «нуль» и «единица», – то и все произведения будут равны или нулю или всему делителю (поэтому вычитать из делимого будем только делитель).

Рассмотрим деление числа $20_{10} = 00010100_2$ на число $6_{10} = 0110_2$.

Рис. 2.17 – *Деление двоичных чисел с восстановлением остатков*

Команда вычитания делителя реализуется сложением делителя в дополнительном коде (знаковый разряд, добавленный слева, – равен единице): $-6_{10} = 1\ 1010_2$.

В компьютерах количество разрядов делимого в два раза больше количества разрядов остальных операндов – делителя, частного и остатка.

После первого вычитания делителя (шаг 1) получился отрицательный остаток (старший бит остатка равен единице). Это означает, что делитель больше левых (старших) значащих разрядов делимого. Поэтому в старший разряд частного записывается «нуль» (противоположный знаковому разряду бит).

Перед следующим шагом вычитания необходимо восстановить делимое. Для этого к нему в **прямом коде** (нулевой знаковый разряд дописан слева) добавляется делитель (шаг 2).

Восстановленный остаток сдвигается на один бит влево (шаг 3) и из него вычитают делитель (шаг 4). Полученный старший бит отрицательного остатка инвертируется и записывается следующим битом частного (см. пунктирные линии на рис. 2.17).

Следующий шаг 5 – восстановление остатка. Шаг 6 – сдвиг восстановленного остатка на один бит влево. После следующего вычитания делителя (шаг 7) новый остаток имеет положительный знак (нулевой знаковый бит). Это означает, что старшие биты предыдущего остатка больше делителя. Поэтому в следующий разряд частного записывается «единица» (противоположный знаковому разряду бит). **Положительный остаток восстанавливать не нужно**.

На шаге 8 остаток сдвигается влево. После следующего вычитания делителя (шаг 9) новый остаток имеет положительный знак. В последний младший бит частного записывается «единица» (противоположная знаковому биту остатка). На шаге 10 остаток сдвигается влево. На этом деление заканчивается.

Количество вычитаний делителя **и сдвигов** влево равно **разрядности делителя** или частного. Количество операций восстановления остатка может быть любым (но не более количества вычитаний).

В результате получено частное: $0011_2 = 3_{10}$ и остаток: $0010_2 = 2_{10}$, что соответствует целочисленному делению десятичных операндов:

$20_{10}\ \text{div}\ 6_{10} = 3_{10}; \quad 20_{10}\ \text{mod}\ 6_{10} = 2_{10}.$

В рассмотренном алгоритме (назовем это **первый алгоритм**) производится сдвиг влево последующих остатков при неподвижном делителе. Аналогичный результат можно получить, используя **второй алгоритм** – сдвиг вправо делителя при неподвижных делимом и остатках (этот алгоритм более привычный при делении десятичных чисел в столбик).

Существенный недостаток этих алгоритмов – это дополнительные затраты времени на восстановление остатков. Практический интерес представляет алгоритм деления двоичных чисел **без восстановления остатков**

(рис. 2.18). Этот алгоритм отличается простотой и регулярностью выполнения.

В примере на рис. 2.18 используются те же операнды, что и в примере на рис. 2.17 : делимое: $20_{10} = 00010100_2$ и делитель: $6_{10} = 0110_2$.

На первом шаге (как и в предыдущем примере) **делитель вычитается** из старших битов делимого. Полученный отрицательный остаток сдвигается влево (без восстановления) с занесением старшего знакового бита в регистр частного (шаг 2).

После отрицательного остатка на следующем шаге 3 **делитель суммируется** со сдвинутым остатком в прямом коде. Полученный следующий отрицательный остаток сдвигается влево (шаг 4) и его старший бит «вдигается» в регистр частного.

После отрицательного остатка на следующем шаге 5 делитель суммируется со сдвинутым остатком в прямом коде. Полученный положительный остаток сдвигается влево (шаг 6) и его старший знаковый бит «вдвигается» в регистр частного.

После положительного остатка на следующем шаге 7 **делитель вычитается** из сдвинутого остатка. Полученный остаток сдвигается влево (шаг 8) и его старший знаковый бит «вдвигается» в регистр частного.

Количество микроопераций вычитания (или суммирования) с последующим сдвигом влево равно разрядности операндов делителя и частного.

Частное	*Делимое*	*Делитель*		
	0 0 0 1 0 1 0 0	0 1 1 0		
	+ 1 1 0 1 0		1	*Вычитание делителя*
	(1) 1 1 0 0 1 0 0			*Отрицательный остаток*
1 ←	1 1 0 0 1 0 0		2	*Сдвиг остатка влево*
	+ 0 0 1 1 0		3	*Сложение с делителем*
	(1) 1 1 1 1 0 0			*Отрицательный остаток*
1 1 ←	1 1 1 1 0 0		4	*Сдвиг остатка влево*
	+ 0 0 1 1 0		5	*Сложение с делителем*
	(0) 0 1 0 0 0			*Положительный остаток*
1 1 0 ←	0 1 0 0 0		6	*Сдвиг остатка влево*
	+ 1 1 0 1 0		7	*Вычитание делителя*
	(0) 0 0 1 0			*Положительный остаток*
1 1 0 0 ←	0 0 1 0		8	*Сдвиг остатка влево*
0 0 1 1			9	*Инверсия частного*

Рис. 2.18 – *Деление двоичных чисел без восстановления остатков*

На последнем шаге 9 все биты частного инвертируются (заменяются на противоположные). Полученные операнды частного и остатка совпадают с результатами алгоритма на рис. 2.17. В реальных схемах отдельный

регистр частного не используется, а сдвиг осуществляется по кольцу, то есть разряды частного вдвигаются в младшие освобождающиеся биты делимого.

В алгоритме деления (рис. 2.18) без восстановления остатков (назовем это **третий алгоритм**) производится сдвиг влево остатков при неподвижном делителе. Аналогичный результат можно получить, используя **четвертый алгоритм** деления без восстановления остатков – сдвиг вправо делителя при неподвижных остатках.

Деление отрицательных и положительных чисел, которые хранятся в памяти или регистрах в дополнительном коде, реализуется в три этапа:

1. определяется знак частного с помощью сложения знаковых (старших) битов делимого и делителя «по модулю 2» (операция XOR – «ИСКЛЮЧАЮЩЕЕ ИЛИ»);
2. перевод значащих битов отрицательных операндов из дополнительного кода – в прямой код и деление положительных чисел одним из четырех ранее описанных алгоритмов;
3. перевод полученного частного в дополнительный код с учетом знака, определенного на первом этапе.

ВЫВОДЫ:

Для выполнения четырех основных арифметических действий над целочисленными операндами (сложение, вычитание, умножение и деление) не нужно в компьютере создавать отдельные: сумматор, вычитатель, умножитель и делитель, – потому что все эти действия сводятся к двум микрооперациям – сложение и сдвиг.

2.3 АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ НАД ЧИСЛАМИ С ПЛАВАЮЩЕЙ ТОЧКОЙ

Особенностью выполнения математических операций над целыми числами является округление результатов до целого значения. Так после деления числа «20» на «6» результат имеет два значения: частное – 3 и остаток – 2. После округления целочисленный процессор выдает результат – 3.

Во всех целочисленных математических операциях абсолютная точность вычислений (или погрешность вычислений) – ΔA – не превышает по величине младший бит результата. Но относительная погрешность ξ будет иметь различные значения и зависит от абсолютного значения «А» результата (см. формулу (1.32)): $\xi = \Delta A / A$.

После деления числа «20» на «6» относительная погрешность будет равна: $\xi = 1 / 3 = 0{,}33$.

А после деления числа «20 000» на «6» относительная погрешность равна: $\xi = 1 / 3333 = 0{,}0003$.

Но и это еще не все проблемы целочисленного процессора. Результаты вычисления синуса и других трансцендентных функций нельзя представить в виде целых чисел с достаточной для инженерных расчетов точностью. Поэтому во всех компьютерах кроме целочисленного процессора используется и процессор чисел с плавающей точкой (сопроцессор) – **Float Point Unit – FPU.**

Сопроцессор в соответствии со стандартом IEEE 754 и более новым, обобщенным стандартом IEEE 854, использует три формата чисел с плавающей точкой, хранимых в памяти (рис. 2.19). В регистрах сопроцессора все числа преобразуются в расширенный формат.

Каждый формат состоит из трех полей: знак (S), порядок и мантисса (рис. 2.19). Числа в этих форматах занимают в памяти: 4, 8 или 10 байт.

Рис. 2.19 – *Форматы чисел с плавающей точкой*

Байт с наименьшим адресом в памяти является младшим байтом мантиссы. Байт с наибольшим адресом содержит семь старших бит порядка и **БИТ ЗНАКА** (S). Знак кодируется: 0 – плюс, 1 – минус.

В ПОЛЕ МАНТИССЫ хранятся только **нормализованные числа**. Для этого необходимо скорректировать порядки (т.е. сдвинуть двоичную точку) так, чтобы в целой части числа (в двоичной системе счисления) до запятой была «1». Поэтому **все нормализованные** мантиссы попадают в диапазон: $1 \le m < 2$ и представляются в двоичной форме:

$$1.XXXXXXX...XXX \quad , \qquad \text{где:} \quad X = 0 \text{ или } 1$$

Но если старший бит всегда содержит 1, ее можно не хранить в каждом числе с плавающей точкой. Поэтому ради дополнительного бита точности сопроцессор хранит в памяти числа одинарной и двойной точности **без старшего бита мантиссы** (с неявным старшим битом). Исключением является спецкодирование нуля – нулевые поля мантиссы и порядка.

Числа с расширенной точностью хранятся и обрабатываются в регистрах сопроцессора **с явным старшим битом.**

ПОЛЕ ПОРЯДКА определяет степень числа 2, на которую нужно умножить нормализованную мантиссу для получения исходного значения числа с плавающей точкой. Число в поле порядка указывает – на сколько двоичных разрядов была сдвинута (влево или вправо) точка при нормализации мантиссы.

Чтобы хранить **отрицательные порядки**, в поле порядка записывается сумма истинного порядка и положительной константы, называемой **СМЕЩЕНИЕМ**. Для одинарной точности смещение равно – 127, для двойной точности – 1023, для расширенной точности – 16383 (т.е. половине максимального порядка). Двоичный код смещения для всех порядков равен : **0111…111** .

Числа в поле порядка: 00000..00 и 11111..111 – зарезервированы для спецкодирования или обработки ошибок.

Запись чисел с плавающей точкой в памяти:

Одинарная точность: $(-1)^S (1 . X1\ X2 \ldots X23) \cdot 2^{(E-127)}$,

Двойная точность: $(-1)^S (1 . X1\ X2 \ldots X52) \cdot 2^{(E-1023)}$,

Расширенная точность: $(-1)^S (X1 . X2 \ldots X64) \cdot 2^{(E-16383)}$,

где S – значение знакового бита;
X1 X2... – биты, хранимые в поле мантиссы;
E – число, хранимое в поле порядка.

Расширенный формат используется преимущественно внутри сопроцессора для представления промежуточных результатов, чтобы упростить получение округленных окончательных результатов в формате двойной точности.

Таблица 2.9 – *Диапазон представления чисел в десятичной системе*

Формат	Значащих десятичных цифр	Наименьшая степень числа 10	Наибольшая степень числа 10
Одинарный	7	–37	38
Двойной	15	–307	308
Расширенный	19	–4931	4932

Диапазон представления чисел с плавающей точкой в десятичной системе счисления приведен в табл. 2.9 и рис. 2.20.

Количество значащих десятичных цифр определяется разрядностью поля мантиссы для каждого формата чисел с плавающей точкой. Наи-

меньшая и наибольшая степени десятичных чисел определяются разрядностью поля порядка.

За счет нормализации результатов математических операций **относительная погрешность результатов для всех чисел в пределах одного формата будет одинаковой.** Для одинарной точности относительная погрешность равна (см. второй столбец таблицы 2.9): $\xi = 10^{-7}$, для двойной точности: $\xi = 10^{-15}$, для расширенной точности: $\xi = 10^{-19}$.

Отметим, что рис. 2.20 абсолютно симметричен для положительных и отрицательных чисел.

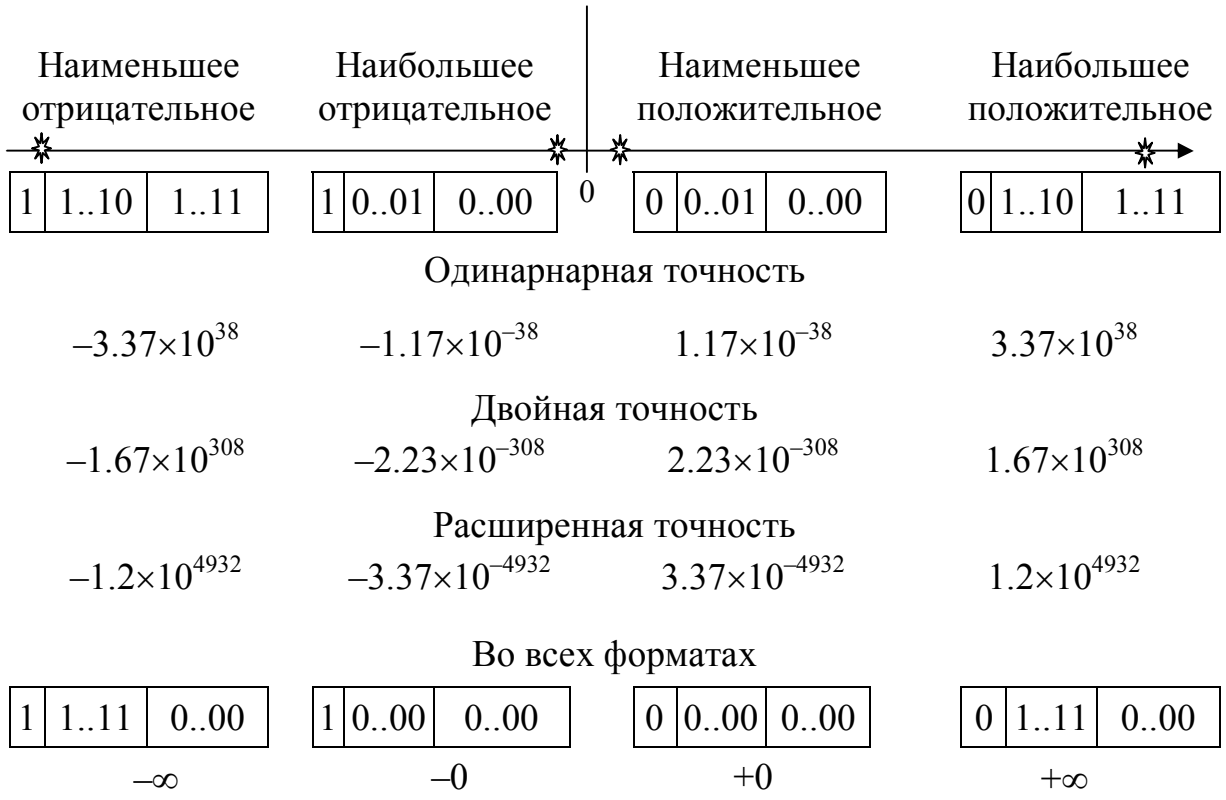


Рис. 2.20 – *Диапазон представления чисел*

Пример преобразования числа 157,465 в формат с плавающей точкой одинарной, двойной и расширенной точности.

На первом этапе необходимо преобразовать десятичное число в двоичный эквивалент (см. рис. 2.4 ÷ рис. 2.10) (для увеличения точности вычислений необходимо рассчитать большее количество битов в дробной части):

157,465 D = 1001 1101 , 0111 0111 0000 1010 0011… B.

После нормализации получаем:

$1 , 001\ 1101\ 0111\ 0111\ 0000\ 1010\ 0011 \cdot 10^{111}$ B.

Для **одинарной точности** к истинному порядку «7» необходимо добавить смещение «127» :

$$\begin{array}{r} 7 \\ +\,127 \\ \hline 134 \end{array} \qquad\qquad \begin{array}{r} 111 \\ +\,01111111 \\ \hline 10000110 \end{array}$$

В поле мантиссы одинарной и двойной точности записывается нормализованная мантисса без старшего значащего бита (с неявным старшим единичным битом).

Далее все двоичные разряды разделяются на тетрады по 4 бита и каждая тетрада заменяется шестнадцатеричным эквивалентом (см. таблицу 2.3):

Знак	Порядок		Мантиса							
0	1 0 0 0 0 1 1 0		0 0 1 1 1 0 1 0	1 1 1 0	1 1 1 0	0 0 0 1	0 1 0	
	4	3	1	D	7	7	0	A

Для **двойной точности** к истинному порядку «7» необходимо добавить смещение «1023» :

$$\begin{array}{r} 7 \\ +\,1023 \\ \hline 1030 \end{array} \qquad\qquad \begin{array}{r} 111 \\ +\,01111111111 \\ \hline 10000000110 \end{array}$$

Знак	Порядок			Мантиса								
0	10000000110			0011101011101110000101000110000000000…								
	4	0	6	3	A	E	E	1	4	6	0	0…

Для **расширенной точности** к истинному порядку «7» необходимо добавить смещение «16383» :

$$\begin{array}{r} 7 \\ +\,16383 \\ \hline 16390 \end{array} \qquad\qquad \begin{array}{r} 111 \\ +\,011111111111111 \\ \hline 100000000000110 \end{array}$$

Знак	Порядок				Мантиса										
0	100000000000110				1001110101110111000010100011000000000000…										
	4	0	0	6	9	D	7	7	0	A	3	0	0	0	…

В соответствии с описанным алгоритмом десятичное число 157,465 будет отображаться на экране монитора в формате с плавающей точкой (младшие биты мантиссы в формате двойной и расширенной точности заменили нулями – поскольку мы их не рассчитывали):

одинарной точности – 431D70A;
двойной точности – 4063AEE146000000;
расширенной точности – 40069D770A3000000000.

Преобразование чисел с плавающей точкой в десятичный формат осуществляется в обратном порядке:

1. по количеству шестнадцатеричных цифр определяется формат числа: 8 цифр – одинарная точность; 16 цифр – двойная точность; 20 цифр – расширенная точность;
2. каждая шестнадцатеричная цифра записывается в двоичном эквиваленте (см. таблицу 2.3);
3. выделяется поле знака, поле порядка и поле мантиссы (см. рис. 2.19);
4. из числа в поле порядка вычитается соответствующее смещение;
5. для одинарной и двойной точности в поле мантиссы записывается дополнительная старшая «единица»;
6. от левой (старшей) значащей единицы в поле мантиссы сдвигается точка (влево или вправо) на количество двоичных разрядов, рассчитанных на 4-м шаге (после вычитания смещения в поле порядка);
7. переводятся в десятичный формат отдельно целая часть числа и отдельно дробная часть числа (см. формулы (2.2), (2.6) и (2.8)).

2.3.1 Сложение-вычитание чисел с плавающей точкой

Аргументы команд сложения или вычитания записываются в регистры сопроцессора в расширенном формате:

$$X1 = (-1)^S \, m1 \cdot 10^{p1}, \qquad X2 = (-1)^S \, m2 \cdot 10^{p2},$$

где: S – содержимое поля знака;
m1, m2 – нормализованные мантиссы аргументов;
p1, p2 – порядки аргументов.

Операции сложения-вычитания чисел с плавающей точкой включает такие этапы:

1. выравнивание порядков аргументов и предварительное определение порядка результата, то есть вычисляется разность порядков: $\Delta = p1 - p2$; если $\Delta > 0$, то сдвигается вправо на «Δ» разрядов мантисса второго аргумента, а если $\Delta < 0$, то сдвигается вправо на «Δ» разрядов мантисса первого аргумента;
2. сложение-вычитание мантисс с выровненными порядками по правилам алгебраического сложения бинарных чисел;
3. нормализация результата и коррекция порядка результата; так как модули всех нормализованных аргументов попадают в диапазон: $1 \le |X| < 2$, то модуль результата сложения или вычитания будет находиться в диапазоне: $0 \le |Y| < 4$, что может вызвать необходимость нормализации

результата на один разряд влево (при $|Y| > 2$) или на произвольное количество разрядов вправо (при $|Y| < 1$);

4. округление результата.

2.3.2 Умножение чисел с плавающей точкой

Аргументы команды умножения записываются в регистры сопроцессора в расширенном формате:

$$X1 = (-1)^S m1 \cdot 10^{p1}, \qquad X2 = (-1)^S m2 \cdot 10^{p2},$$

где: S – содержимое поля знака;
m1, m2 – нормализованные мантиссы сомножителей;
p1, p2 – порядки сомножителей.

Произведение определяется по формуле:

$$X1 \cdot X2 = m1 \cdot m2 \cdot 10^{(p1 + p2)}. \qquad (2.21)$$

Поэтому умножение чисел в сопроцессоре производится в четыре этапа:

1. определение знака произведения путем «сложения по модулю 2» знаковых битов сомножителей;
2. перемножение мантисс сомножителей;
3. определение порядка произведения путем алгебраического сложения порядков сомножителей;
4. нормализация результата (в случае необходимости) и округление мантиссы произведения, потому что разрядность произведения в два раза больше разрядности сомножителей.

2.3.3 Деление чисел с плавающей точкой

Аргументы команды деления записываются в регистры сопроцессора в расширенном формате:

$$X1 = (-1)^S m1 \cdot 10^{p1}, \qquad X2 = (-1)^S m2 \cdot 10^{p2},$$

где: S – содержимое поля знака;
m1, m2 – нормализованные мантиссы делимого и делителя;
p1, p2 – порядки делимого и делителя.

Частное от деления определяется по формуле:

$$X1 / X2 = (m1 / m2) \cdot 10^{(p1 - p2)}. \qquad (2.22)$$

Поэтому деление чисел в сопроцессоре производится в четыре этапа:

1. определение знака частного путем «сложения по модулю 2» знаковых битов делимого и делителя;
2. деление мантиссы делимого на мантиссу делителя;

3. определение порядка частного путем алгебраического вычитания из порядка делимого порядка делителя;

4. нормализация результата (в случае необходимости) и округление мантиссы частного.

2.4 ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Назовите основные системы счисления, которые используются в компьютерах.
2. Какие системы счисления называются позиционными?
3. Назовите основные алгоритмы перевода чисел из двоичной, восьмеричной, шестнадцатеричной системы счисления – в десятичную.
4. Почему точные значения приставок Кило, Мега и др. не совпадают в десятичной системе счисления и в компьютерной литературе?
5. От чего зависит диапазон представимых чисел в различных системах счисления?
6. Назовите диапазон представимых двоичных чисел размером: «байт», «слово», «двойное слово».
7. Назовите основные преимущества и недостатки двоично-десятичного кода.
8. Назовите основное отличие кода Грея от обычного позиционного двоичного кода.
9. Назовите алгоритм перевода позиционного двоичного кода в код Грея.
10. Как в компьютерах кодируются положительные и отрицательные числа?
11. Назовите алгоритм перевода прямого кода в дополнительный код и наоборот.
12. Чем отличается диапазон представимых положительных беззнаковых чисел и чисел в дополнительном коде?
13. Как по разрядности слагаемых определить разрядность суммы?
14. Как в компьютерах выполняется операция вычитания?
15. Зачем в компьютерах применяют флаги: CARRY Flag – CF, SIGN Flag – SF, OVERFLOW Flag – OF, ZERO Flag – ZF ?
16. Как в компьютерах выполняется операция сравнения?
17. Как изменяется величина операндов при сдвигах числа вправо или влево?
18. Как определить переполнение при сдвигах двоичных операндов?
19. Чем отличаются команды сдвигов положительных и отрицательных чисел?
20. Как по разрядности сомножителей определить разрядность произведения?
21. Назовите основные алгоритмы умножения двоичных чисел.
22. Назовите основные алгоритмы деления двоичных чисел.

23. Чем отличаются алгоритмы умножения и деления положительных и отрицательных чисел?
24. От чего зависит абсолютная и относительная погрешность вычислений в целочисленных процессорах и процессорах чисел с плавающей точкой?
25. Какие форматы чисел с плавающей точкой использует сопроцессор?
26. Что такое нормализация дробных чисел и как она производится?
27. В каком диапазоне чисел может находиться нормализованная мантисса?
28. Что записывается в поле порядка и поле мантиссы чисел с плавающей точкой одинарной, двойной и расширенной точности?
29. Назовите алгоритмы преобразования десятичных чисел в форматы с плавающей точкой и обратное преобразование.
30. Особенности выполнения математических операций над числами с плавающей точкой.