2.1 Машинные элементы информации

Обычно входные и выходные данные представляются в форме, удобной для человека. Числа люди привыкли изображать в десятичной системе счисления. Для компьютера удобнее двоичная система. Это объясняется тем, что технически гораздо проще реализовать устройства (например, запоминающий элемент) с двумя, а не с десятью устойчивыми состояниями . Обычно сигнал от 0 до 1 В представляет одно значение (например, 0), а сигнал от 2 до 5 В — другое значение (например, 1). Напряжение за пределами указанных величин недопустимо. Крошечные электронные устройства, которые называются вентилями, позволяют получать различные функции от этих двузначных сигналов. Вентили лежат в основе аппаратного обеспечения, на котором строятся все цифровые компьютеры.

Описание принципов работы вентилей выходит за рамки данного курса лекций, поскольку относится к уровню физических устройств, который находится ниже уровня 0. Тем не менее мы очень кратко коснемся основного принципа, который не так уж и сложен. Вся современная цифровая логика основывается на том, что транзистор может работать как очень быстрый двоичный переключатель. На рис. 2.1.1, (а) изображен биполярный транзистор, встроенный в простую схему. Транзистор имеет три соединения с внешним миром: коллектор, базу и эмиттер. Если входное напряжение V_{in} ниже определенного критического значения, транзистор выключается и действует как очень большое сопротивление. Это приводит к выходному сигналу V_{out} , близкому к V_{CC} (напряжению, подаваемому извне), — для данного типа транзистора это обычно +5 В. Если V_{in} превышает критическое значение, транзистор включается и действует как проводник, вызывая заземление сигнала V_{out} (по соглашению — это 0 В).

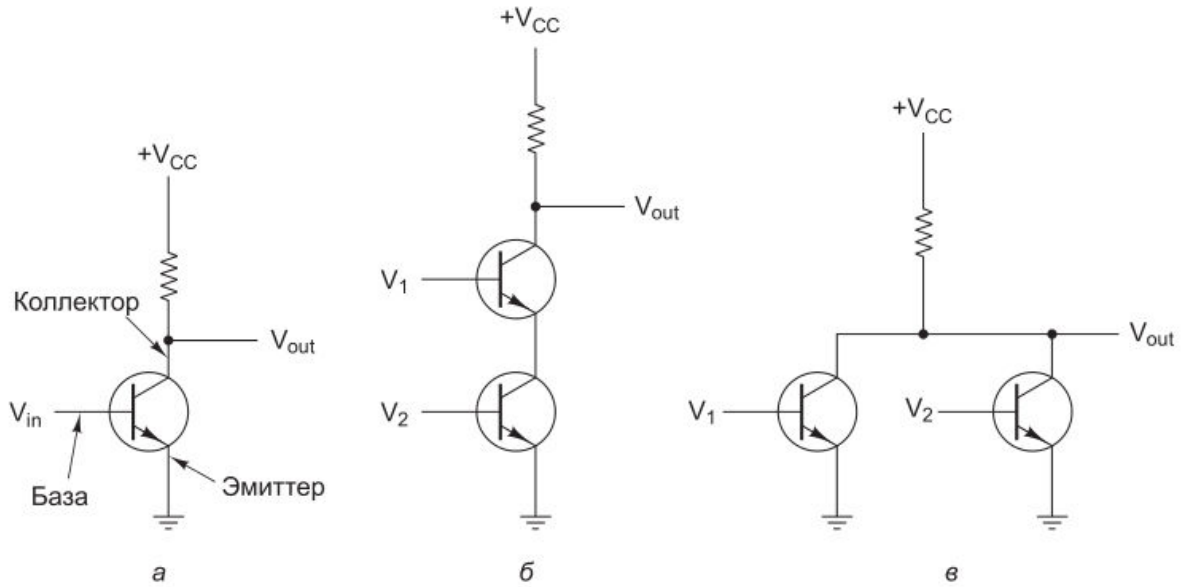


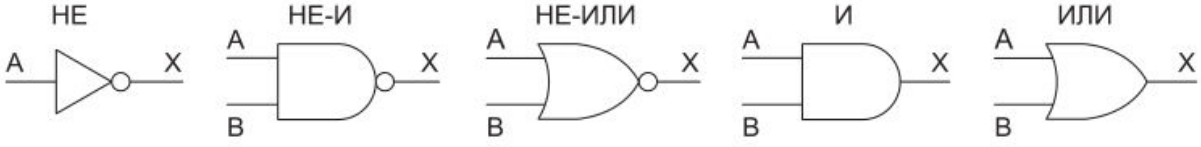
Рис. 2.1.1 Транзисторный инвертор (а); вентиль НЕ И (б); вентиль НЕ ИЛИ (в)

Важно отметить, что если напряжение V_{in} низкое, то V_{out} высокое, и наоборот. Эта схема, таким образом, является инвертором, превращающим логический 0 в логическую 1 и логическую 1 в логический 0. Резистор нужен для ограничения тока, проходящего через транзистор, чтобы транзистор не сгорел. На переключение с одного состояния на другое обычно требуется не более наносекунды.

На рис. 2.1.1, (б) два транзистора соединены последовательно. Если и напряжение V_1 , и напряжение V_2 высокое, то оба транзистора становятся проводниками и снижают V_{out} . Если одно из входных напряжений низкое, то соответствующий транзистор выключается, и напряжение на выходе становится высоким. Другими словами, напряжение V_{out} является низким тогда и только тогда, когда и напряжение V_1 , и напряжение V_2 высокое.

На рис. 2.1.1, (в) два транзистора соединены параллельно. Если один из входных сигналов высокий, включается соответствующий транзистор и снижает выходной сигнал. Если оба напряжения на входе низкие, то выходное напряжение становится высоким.

Эти три схемы образуют три простейших вентиля. Они называются вентилями НЕ, НЕ-И и НЕ-ИЛИ соответственно. Вентили НЕ часто называют инверторами. Если мы примем соглашение, что высокое напряжение (V_{CC}) — это логическая 1, а низкое напряжение («земля») — логический 0, то мы сможем выражать значение на выходе как функцию от входных значений. Значки, которые используются для изображения этих трех типов вентилей, показаны на рис. 2.1.2, а–в. Там же показаны режимы работы функции для каждой схемы. На этих рисунках A и B — входные сигналы, X — выходной сигнал. Каждая строка таблицы определяет выходной сигнал для различных комбинаций входных сигналов.



A	X
0	1
1	0

а

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

б

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

в

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

г

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

д

Рис. 2.1.2. Значки для изображения пяти основных вентилей. Режимы работы функции для каждого вентиля

Если выходной сигнал на рис. 2.1.1.2, (б) подать в инвертор, мы получим другую схему, противоположную вентилю НЕ-И, то есть такую схему, у которой выходной сигнал равен 1 тогда и только тогда, когда оба входных сигнала равны 1.

Такая схема называется вентилем И; ее схематическое изображение и описание соответствующей функции даны на рис. 2.1.2, (г). Точно так же

вентиль НЕ-ИЛИ может быть связан с инвертором. Тогда получится схема, у которой выходной сигнал равен 1 в том случае, если хотя бы один из входных сигналов единичный, и равен 0, если оба входных сигнала нулевые. Изображение этой схемы, которая называется вентилем ИЛИ, а также описание соответствующей функции даны на рис. 2.1.2, (д). Маленькие кружочки в схемах инвертора, вентиля НЕ-И и вентиля НЕ-ИЛИ называются инвертирующими выходами. Они также могут использоваться в другом контексте для указания на инвертированный сигнал.

Пять вентилей, изображенные на рис. 2.1.2, составляют основу цифрового логического уровня. Из предшествующего обсуждения должно быть ясно, что вентили НЕ-И и НЕ-ИЛИ требуют два транзистора каждый, а вентили И и ИЛИ — три транзистора каждый. По этой причине во многих компьютерах используются вентили НЕ-И и НЕ-ИЛИ, а не И и ИЛИ. Следует упомянуть, что вентили могут иметь более двух входов. В принципе вентиль НЕ-И, например, может иметь произвольное количество входов, но на практике больше восьми обычно не бывает.

Чтобы описать схемы, получаемые сочетанием различных вентилей, нужен особый тип алгебры, в которой все переменные и функции могут принимать только два значения: 0 и 1. Такая алгебра называется булевой. Она названа в честь английского математика Джорджа Буля (1815–1864).

2.2 Представление чисел с конечной точностью

Когда люди выполняют какие-либо арифметические действия, их не волнует вопрос, сколько десятичных разрядов занимает то или иное число. Физики, к примеру, могут вычислить, что во вселенной существуют 10^{78} электронов, и их не волнует тот факт, что полная запись этого числа потребует 79 десятичных разрядов. Проблема нехватки бумаги для записи числа никогда не возникает.

С компьютерами дело обстоит иначе. В большинстве компьютеров количество доступной памяти для хранения чисел фиксировано и зависит от того, когда был выпущен этот компьютер. Если приложить усилия, программист сможет представлять числа в два, три и более раз большие, чем позволяет объем памяти, но это не меняет природы данной проблемы. Память компьютера ограничена, поэтому мы можем иметь дело только с такими числами, которые можно представить в фиксированном количестве разрядов. Такие числа называются **числами конечной точности**.

Рассмотрим ряд положительных целых чисел, которые можно записать тремя десятичными разрядами без десятичной точки и без знака. В этот ряд входит ровно 1000 чисел: 000, 001, 002, 003, ..., 999. При таком ограничении невозможно выразить определенные типы чисел. Сюда входят:

- числа больше 999;
- отрицательные числа;
- дроби;
- иррациональные числа;

- комплексные числа.

Одно из свойств набора всех целых чисел — **замкнутость** по отношению к операциям сложения, вычитания и умножения. Другими словами, для каждой пары целых чисел i и j числа $i + j$, i $– j$ и $i \times j$ — тоже целые числа. Ряд целых чисел не замкнут относительно деления, поскольку существуют такие значения i и j, для которых i/j не выражается в виде целого числа (например, 7/2 или 1/0).

Числа конечной точности не замкнуты относительно всех четырех операций. Вот примеры операций над трехразрядными десятичными числами.

Слишком большое число:

$$600 + 600 = 1200.$$

Отрицательное число:

$$003 – 005 = –2.$$

Слишком большое число:

$$050 \times 050 = 2500.$$

Не целое число:

$$007 / 002 = 3{,}5.$$

Отклонения можно разделить на два класса: операции, результат которых больше самого большого числа ряда (ошибка переполнения) или меньше самого маленького числа ряда (ошибка потери значимости), и операции, результат которых не является слишком маленьким или слишком большим, а просто не является членом ряда. Из четырех приведенных примеров первые три относятся к первому классу, а четвертый — ко второму классу.

Поскольку объем памяти компьютера ограничен, и компьютер должен выполнять арифметические действия над числами конечной точности, с точки зрения классической математики результаты определенных вычислений оказываются неправильными. Ошибка в данном случае — это только следствие конечной природы представления чисел в вычислительном устройстве. Некоторые компьютеры имеют встроенную аппаратную поддержку для обнаружения ошибок переполнения.

Алгебра чисел конечной точности отличается от обычной алгебры. В качестве примера рассмотрим ассоциативный закон

$$a + (b – c) = (a + b) – c.$$

Вычислим обе части выражения для $a = 700$, $b = 400$ и $c = 300$. В левой части сначала вычислим значение $(b – c)$. Оно равно 100. Затем прибавим это число к a и получим 800. Чтобы вычислить правую часть, сначала вычислим $(a + b)$. Для 3-разрядных целых чисел получится переполнение. Результат будет зависеть от компьютера, но он окажется неравным 1100. Вычитание 300 из какого-то числа, отличного от 1100, не даст в результате 800. Ассоциативный закон не имеет силы. Важна очередность выполнения операций.

Другой пример — дистрибутивный закон:

$$a \times (b – c) = a \times b – a \times c.$$

Вычислим обе части выражения для $a = 5$, $b = 210$ и $c = 195$. В левой части $5 \times 15 = 75$. В правой части 75 не получается, поскольку результат выполнения операции $a \times b$ выходит за пределы ряда.

Исходя из этих примеров, кто-то может сделать вывод, что компьютеры совершенно непригодны для выполнения арифметических действий. Вывод, естественно, неверен, но эти примеры наглядно показывают, как важно понимать механизм работы компьютера и знать о его ограничениях.

2.3 Позиционные системы счисления

Обычное десятичное число состоит из цепочки десятичных разрядов и иногда десятичной точки (запятой). Общая форма записи показана на рис. 2.3.1. Десятка выбрана в качестве основы возведения в степень (и называется **основанием системы счисления**), поскольку мы используем 10 цифр. В компьютерах удобнее иметь дело с другими основаниями системы счисления. Самые важные из них — 2, 8 и 16. Соответствующие системы счисления называются **двоичной**,

Рис. 2.3.1. Общая форма десятичного числа

Система счисления с основанием k требует k различных символов для записи разрядов с 0 по $k - 1$. Десятичные числа строятся из 10 десятичных цифр:

0 1 2 3 4 5 6 7 8 9

Двоичные числа, напротив, строятся только из двух двоичных цифр:

0 1

Восьмеричные числа состоят из восьми цифр:

0 1 2 3 4 5 6 7

Для шестнадцатеричных чисел требуется 16 цифр. Это значит, что нам нужно 6 новых символов. Для обозначения цифр, следующих за 9, принято использовать прописные латинские буквы от A до F. Таким образом, шестнадцатеричные числа строятся из следующих цифр:

0 1 2 3 4 5 6 7 8 9 A B C D E F

Двоичный разряд (то есть 1 или 0) обычно называют **битом**. На рис. А.2 десятичное число 2001 представлено в двоичной, восьмеричной и шестнадцатеричной системах счисления. Число 7B9, очевидно, шестнадцатеричное, поскольку символ B встречается только в

шестнадцатеричных числах. А число 111 может быть записано в любой из четырех систем счисления. Чтобы избежать неоднозначности, следует указывать основание системы счисления (если оно не очевидно из контекста).

2.4 Преобразование чисел из одной системы счисления в другую

Преобразовывать числа из восьмеричной в шестнадцатеричную или в двоичную систему счисления и обратно легко. Чтобы преобразовать двоичное число в восьмеричное, нужно разделить его на группы по три бита, причем три бита непосредственно слева от двоичной запятой формируют одну группу, следующие три бита слева от этой группы формируют вторую группу и т. д. Каждую группу по три бита можно преобразовать в один восьмеричный разряд со значением от 0 до 7 (см. первые строки табл. А.1). Чтобы дополнить группу до трех битов, нужно спереди приписать один или два нуля. Преобразование из восьмеричной системы в двоичную тоже тривиально. Каждый восьмеричный разряд просто заменяется эквивалентным 3-разрядным числом. Преобразование из шестнадцатеричной в двоичную систему по сути сходно с преобразованием из восьмеричной в двоичную систему, только каждый шестнадцатеричный разряд соответствует группе из четырех битов, а не из трех. На рис. 2.9.1 приведены примеры преобразований из одной системы в другую.

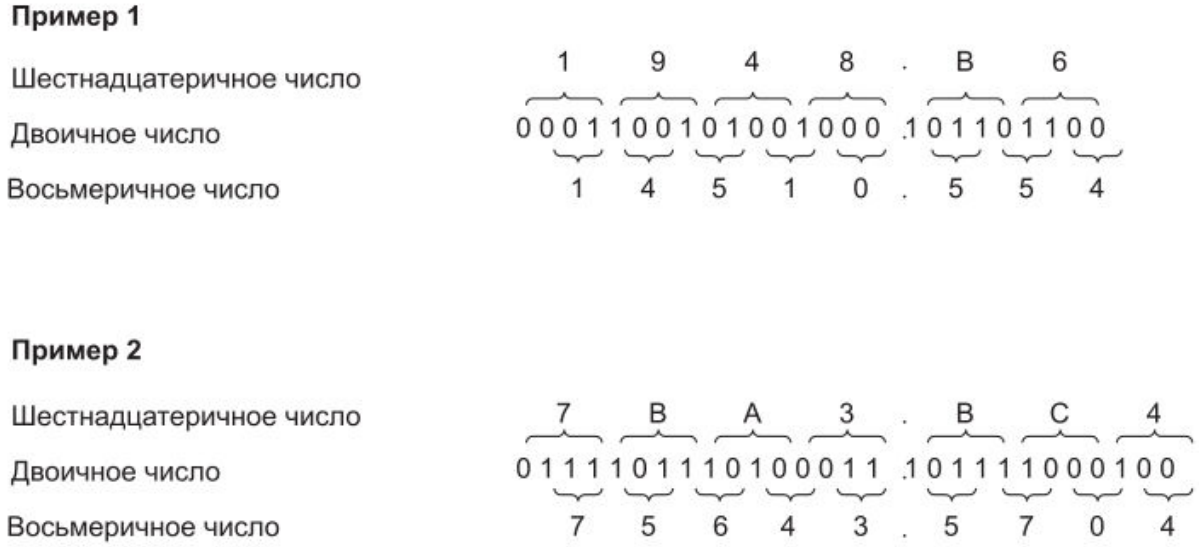


рис 2.9.1 Примеры преобразования из восьмеричной системы в двоичную и из шестнадцатеричной в двоичную

Преобразование десятичных чисел в двоичные можно совершать двумя разными способами. Первый способ непосредственно вытекает из определения двоичных чисел. Самая большая степень двойки, меньшая, чем число, вычитается из этого числа. Та же операция проделывается с полученной разностью. Когда число разложено по степеням двойки, двоичное число может быть получено следующим образом. Единички ставятся в тех позициях, которые соответствуют полученным степеням двойки, а нули — во всех остальных позициях.

Второй способ (подходящий только для целых чисел) — деление на 2. Частное записывается непосредственно под исходным числом, а остаток (0 или 1) записывается рядом с частным. То же проделывается с полученным частным. Процесс повторяется до тех пор, пока не повторяется до тех пор, пока не останется 0. В результате должно получиться две колонки чисел — частных и остатков. Двоичное число можно считать из колонки остатков снизу вверх. На рис. 2.4.2 показано, как происходит преобразование из десятичной в двоичную систему.

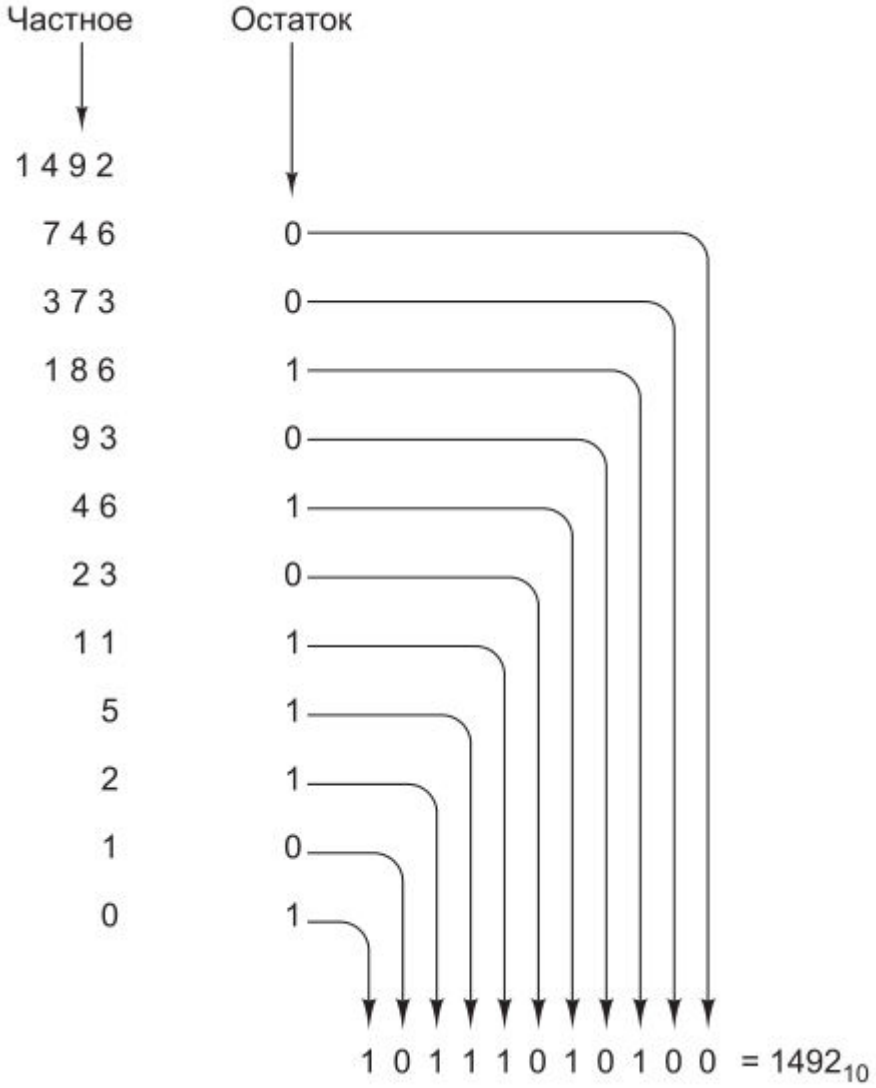


рис. 2.4.2 Пример преобразования десятичного числа в двоичное путем последовательного деления

Двоичные числа можно преобразовывать в десятичные двумя способами. Первый способ — суммирование степеней двойки, которые соответствуют битам 1 в двоичном числе. Например:

10110 = 24 + 22 + 21 = 16 + 4 + 2 = 22.

Во втором способе двоичное число записывается вертикально по одному биту в строке, крайний левый бит находится внизу. Самая нижняя строка — это строка 1, затем идет строка 2 и т. д. Десятичное число строится напротив этой колонки. Сначала обозначим строку 1. Элемент строки *n* состоит из удвоенного элемента строки *n* – 1 плюс бит строки *n* (0 или 1).

Элемент, полученный в самой верхней строке, и будет ответом. Метод иллюстрирует рис. 2.4.3.

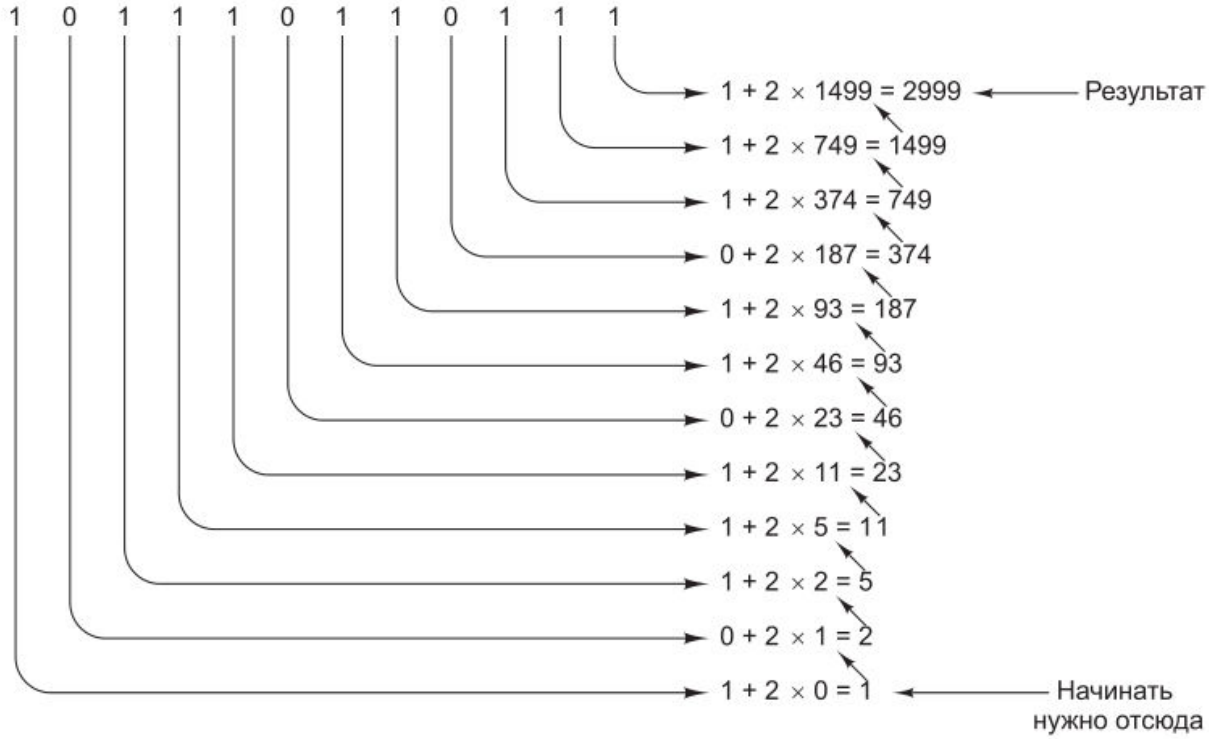


рис. 2.4.3 Пример преобразования двоичного числа в десятичное путем последовательного удвоения

Преобразование из десятичной в восьмеричную или шестнадцатеричную систему можно выполнить либо путем преобразования сначала в двоичную, а затем в нужную нам систему, либо путем вычитания степеней 8 или 16.

2.5 Отрицательные двоичные числа

На протяжении всей истории цифровых компьютеров для представления отрицательных чисел использовались 4 различные системы. Первая из них называется системой **со знаком**. В такой системе крайний левый бит — это знаковый бит (0 — плюс, 1 — минус), а оставшиеся биты показывают абсолютное значение числа.

Во второй системе, которая называется **дополнением до единицы**, тоже присутствует знаковый бит (0 — плюс, 1 — минус). Чтобы сделать число отрицательным, нужно заменить каждую единицу нулем и каждый ноль единицей. Это относится и к знаковому биту. Система дополнения до единицы уже устарела.

Третья система, **дополнение до двух**, содержит знаковый бит (0 — плюс, 1 — минус). Отрицание числа происходит в два этапа. Сначала каждая единица меняется на ноль, а каждый ноль — на единицу (как и в системе дополнения до единицы). Затем к полученному результату прибавляется единица.

В четвертой системе, которая для *m*-разрядных чисел называется **системой со смещением на 2m–1**, число представляется как сумма этого

числа и $2m$–1. Например, для 8-разрядного числа (m = 8) — это система со смещением на 128, в ней число сохраняется в виде суммы исходного числа и 128. Следовательно, –3 превращается в –3 + 128 = 125, и это число (–3) представляется 8-разрядным двоичным числом 125 (01111101). Числа от –128 до +127 выражаются числами от 0 до 255 (все их можно записать в виде 8-разрядного положительного числа). Отметим, что эта система соответствует системе с дополнением до двух с обращенным знаковым битом. В табл. 2.5.1 представлены примеры отрицательных чисел во всех четырех системах.

Таблица 2.5.1 Отрицательные 8-разрядные числа в четырех различных системах

N десятичное	N двоичное	–N в системе со знаком	–N дополнение до единицы	–N дополнение до двух	–N смещение на 128
1	00000001	10000001	11111110	11111111	011111111
2	00000010	10000010	11111101	11111110	01111110
3	00000011	10000011	11111100	11111101	01111101
4	00000100	10000100	11111011	11111100	01111100
5	00000101	10000101	11111010	11111011	01111011
6	00000110	10000110	11111001	11111010	01111010
7	00000111	10000111	11111000	11111001	01111001
8	00001000	10001000	11110111	11111000	01111000
9	00001001	10001001	11110110	11110111	01110111
10	00001010	10001010	11110101	11110110	01110110
20	00010100	10010100	11101011	11101100	01101100
30	00011110	10011110	11100001	11100010	01100010
40	10101000	10101000	11010111	11011000	01011000
50	00110010	10110010	11001101	11001110	01001110
60	00111100	10111100	11000011	11000100	01000100
70	01000110	11000110	10111001	10111010	00111010
80	01010000	11010000	10101111	10110000	00110000
90	01011010	11011010	10100101	10100110	00100110
100	01100100	11100100	10011011	10011100	00011100
127	01111111	11111111	10000000	10000001	00000001
128	Не существует	Не существует	Не существует	10000000	00000000

В системах со знаком и с дополнением до единицы есть два представления нуля: +0 и –0. Такая ситуация нежелательна. В системе с дополнением до двух такой проблемы нет, поскольку здесь плюс ноль — это всегда плюс ноль. Но зато в этой системе есть другая особенность. Набор битов, состоящий из единицы, за которой следуют все нули, является дополнением самого себя. В результате ряд положительных и отрицательных чисел несимметричен — существует одно отрицательное число без соответствующего ему положительного.

Мы считаем, что это проблема, поскольку хотим иметь систему кодировки, в которой:

существует только одно представление нуля;

количество положительных чисел равно количеству отрицательных.

Дело в том, что любой ряд чисел с равным количеством положительных и отрицательных чисел и только одним нулем содержит нечетное число членов, тогда как *m* бит предполагают четное число битовых комбинаций. В любом случае либо одна битовая комбинация окажется лишней, либо одной комбинации будет недоставать. Лишнюю битовую комбинацию можно использовать для обозначения числа –0, для большого отрицательного числа или для чего-нибудь еще, но она всегда будет создавать неудобства.

2.6 Принципы представления чисел с плавающей точкой

Числа можно выражать в следующей общепринятой экспоненциальной форме:

$n = f \times 10^e$,

где *f* называется **мантиссой**, а *e* (это положительное или отрицательное целое число) — **экспонентой**. Компьютерная версия такого представления называется представлением числа с **плавающей точкой**.

Область значений определяется по числу разрядов в экспоненте, а точность — по числу разрядов в мантиссе.

До 80-х годов каждый производитель поддерживал собственный формат чисел с плавающей точкой. Все они отличались друг от друга. Более того, в некоторых из них арифметические действия выполнялись неправильно, поскольку арифметика с плавающей точкой имеет некоторые тонкости, которые не очевидны.

Чтобы изменить эту ситуацию, в конце 70-х годов институт IEEE учредил комиссию по стандартизации арифметики с плавающей точкой. Целью было не только дать возможность переносить данные с одного компьютера на другой, но и обеспечить разработчиков аппаратного обеспечения заведомо правильной моделью. В результате в 1985 году вышел стандарт IEEE 754 [IEEE, 1985]. В настоящее время большинство процессоров (в том числе Intel, SPARC и JVM) содержат команды с плавающей точкой, которые соответствуют этому стандарту. В отличие от многих стандартов, ставших плодом неудачных компромиссов и мало кого устраивавших, этот стандарт неплох, причем в значительной степени благодаря тому, что его изначально разрабатывал один человек, профессор математики университета Беркли Вильям Каган (William Kahan). Рассмотрим этот стандарт.

Стандарт IEEE 754 определяет три формата: с одинарной точностью (32 бита), с удвоенной точностью (64 бита) и с повышенной точностью (80 бит). Формат с повышенной точностью предназначен для уменьшения ошибки округления. Он применяется главным образом в арифметических

устройствах с плавающей точкой, поэтому мы не будем о нем говорить. В форматах с одинарной и удвоенной точностью используются основание степени 2 для мантисс и смещенная экспонента. Форматы представлены на рис. 2.6.1.

а

б

рис. 2.6.1 Форматы стандарта IEEE с плавающей запятой: одинарная точность – а, двойная точность – б.

Оба формата начинаются со знакового бита для всего числа; 0 указывает на положительное число, 1 — на отрицательное. Затем следует смещенная экспонента. Для формата одинарной точности смещение равно 127, а для формата удвоенной точности — 1023. У них есть специальное предназначение, о котором мы поговорим позже. В конце идут мантиссы по 23 и 52 бита соответственно.

Числовые характеристики стандарта IEEE для чисел с плавающей точкой даны в табл. 2.6.1

табл. 2.6.1 Характеристики чисел с плавающей точкой стандарта IEEE

	Одинарная точность	**Удвоенная точность**
Количество битов в знаке	1	1
Количество битов в экспоненте	8	11
Количество битов в мантиссе	23	52
Общее число битов	32	64
Смещение экспоненты	Смещение 127	Смещение 1023
Область значений экспоненты	От −126 до +127	от −1022 до +1023
Самое маленькое нормализованное число	2^{-126}	2^{-1022}
Самое большое нормализованное число	Приблизительно 2^{128}	Приблизительно 2^{1024}
Диапазон десятичных дробей	Приблизительно от 10^{-38} до 10^{38}	Приблизительно от 10^{-308} до 10^{308}
Самое маленькое ненормализованное число	Приблизительно 10^{-45}	Приблизительно 10^{-324}

Традиционные проблемы, связанные с числами с плавающей точкой, — переполнение, потеря значимости (см. рис. 2.6.2) и неинициализированные числа. Подход, используемый в стандарте IEEE, отчасти заимствован у машины CDC 6600.

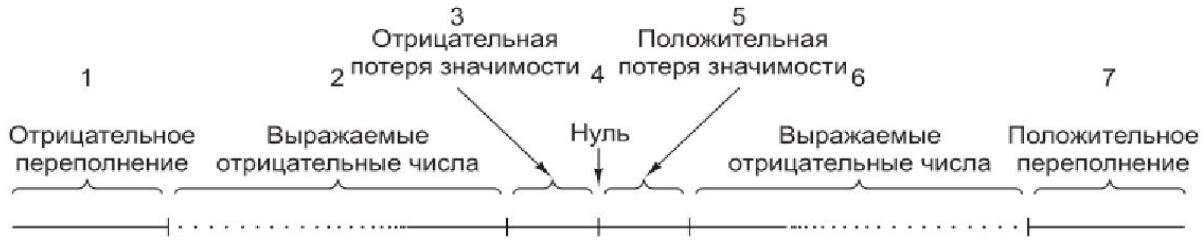


рис. 2.6.2 Область определения чисел с плавающей точкой

Проблема возникает в том случае, если абсолютное значение (модуль) результата меньше самого маленького нормализованного числа с плавающей точкой, которое можно представить в этой системе. Раньше аппаратное обеспечение действовало одним из двух способов: либо устанавливало результат на 0, либо вызывало ошибку потери значимости. Ни один из этих двух способов не является удовлетворительным, поэтому в стандарт IEEE введены **ненормализованные числа**. Эти числа имеют экспоненту 0 и мантиссу, представленную следующими 23 или 52 битами. Неявный бит 1 слева от двоичной точки превращается в 0. Ненормализованные числа можно легко отличить от нормализованных, поскольку у последних не может быть нулевой экспоненты.

Самое маленькое нормализованное число с одинарной точностью содержит 1 в экспоненте и 0 в мантиссе и представляет $1{,}0 \times 2^{-126}$. Самое большое ненормализованное число содержит 0 в экспоненте и все единицы в мантиссе и представляет примерно $0{,}9999999 \times 2^{-127}$, то есть почти то же

самое число. Следует отметить, что это число содержит только 23 бита значимости, а все нормализованные числа — 24 бита.

По мере уменьшения результата при дальнейших вычислениях экспонента по-прежнему остается равной 0, а первые несколько битов мантиссы превращаются в нули, что уменьшает и значение, и число значимых битов мантиссы. Самое маленькое ненулевое ненормализованное число содержит 1 в крайнем правом бите, а все остальные биты равны 0. Экспонента представляет 2^{-127}, а мантисса — 2^{-23}, поэтому значение равно 2^{-150}. Такая схема предусматривает постепенное исчезновение значимых разрядов, а не перескакивает на 0, когда результат не удается выразить в виде нормализованного числа.

В этой схеме присутствуют два нуля, положительный и отрицательный, определяемые по знаковому биту. Оба имеют экспоненту 0 и мантиссу 0. Здесь тоже бит слева от двоичной точки по умолчанию равен 0, а не 1.

Простого решения проблемы переполнения нет. Для этого существует специальное представление бесконечности: с экспонентой, содержащей все единицы, и мантиссой, равной 0. Это число можно использовать в качестве операнда. Оно подчиняется обычным математическим правилам для бесконечности. Например, бесконечность и любое число в сумме дают бесконечность. Конечное число, деленное на бесконечность, равно 0. Любое конечное число, разделенное на 0, стремится к бесконечности.

А что получится, если бесконечность разделить на бесконечность? Результат не определен. Для такого случая существует другой специальный формат — **не число** (Not a Number, **NaN**). Его тоже можно использовать в качестве операнда.