Корректирующие коды

Контроль работы цифрового автомата. Систематические коды.

- Алгоритмы выполнения арифметических операций обеспечат правильный результат только в случае, если машина работает без нарушений. При возникновении какого-либо нарушения нормального функционирования результат будет неверным, однако пользователь об этом не узнает, если не будут предусмотрены меты для создания системы обнаружения возможной ошибки, а с другой стороны, должны быть проработаны меры, позволяющие исправить ошибки. Эти функции следует возложить на систему контроля работы цифрового автомата.
- **Система контроля —** совокупность методов и средств, обеспечивающих определение правильности работы автомата в целом или его отдельных узлов, а также автоматическое исправление ошибки.
- Ошибки в работе цифрового автомата могут быть вызваны либо выходом из строя какой-то детали, либо отклонением от нормы параметров (например, изменение напряжения питания) или воздействием внешних помех. Вызванные этими нарушениями ошибки могут принять постоянный или случайный характер. Постоянные ошибки легче обнаружить и выявить. Случайные ошибки, обусловленные кратковременными изменениями параметров, наиболее опасны и их труднее обнаружить.
- Поэтому система контроля должна строится с таким расчетом, чтобы она позволяла обнаружить и по возможности исправить любые нарушения. При этом надо различать следующие виды ошибок результата:
- возникающие из-за погрешностей в исходных данных;
- обусловленные методическими погрешностями;
- появляющиеся из-за возникновения неисправностей в работе машины.

- Проверка правильности функционирования отдельных устройств машины и выявление неисправностей может осуществляться по двум направлениям:
- профилактический контроль, задача которого – предупреждение появления ошибок в работе;
- оперативный контроль, задача которого – проверка правильности выполнения машиной всех операций.
- Решение всех задач контроля становится возможным только при наличии определенной **избыточности.** Избыточность может быть либо аппаратными (схемными) средствами, либо логическими или информационными средствами. К методам логического контроля можно отнести следующие приемы. В ЭВМ первого и второго поколений отсутствие системы оперативного контроля приводило к необходимости осуществления «двойного счета», когда каждая задача решалась дважды, и в случае совпадения ответов принималось решение о правильности функционирования ЭВМ.
- Первые два вида ошибок не являются объектом для работы системы контроля. Конечно, погрешности перевода или представления числовой информации в разрядной сетки автомата приведут к возникновению погрешности в результате решения задачи. Эту погрешность можно заранее рассчитать и, зная её максимальную величину, правильно выбрать длину разрядной сетки машины. Методические погрешности также учитываются предварительно.
- Если в процессе решения какой-то задачи вычисляются тригонометрические функции, то для контроля можно использовать известные соотношения между этими функциями, например,

$$\sin^2 x + \cos^2 x = 1$$

- Если это соотношение выполняется заданной точностью на каждом шаге вычислений, то можно с уверенностью читать, что ЭВМ работает правильно.
- Вычисление определенного интеграла с заданным шагом интегрирования можно контролировать сравнением полученных при этом результатов с теми результатами, которые соответствуют более крупному шагу. Такой «сокращенный» алгоритм даст, видимо, более грубые оценки и по существу требует дополнительных затрат машинного времени.
- Все рассмотренные примеры свидетельствуют о том, что такие методы контроля позволяют лишь зафиксировать факт появления ошибки, но не определяют место, где произошла эта ошибка. Для оперативного контроля работы ЭВМ определение места, где произошла ошибка, т.е. решение задачи поиска неисправности, является весьма существенным вопросом.

4. Систематические коды

- Как уже указывалось, функции контроля можно осуществить при информационной избыточности. Такая возможность появляется при использовании специальных методов кодирования информации. В самом деле, некоторые методы кодирования информации допускают наличие разрешенных и запрещенных комбинаций. В качестве примера можно привести двоично-десятичные системы представления числовой информации (Д-коды). Появление запрещенных комбинаций для подобного представления свидетельствует об ошибке в результатах решения задачи. Такой метод можно использовать для контроля десятичных операций. Однако он является частным примером и не решает общей задачи.

- Задача кодирования информации представляется как некоторое преобразование числовых данных в заданной программе счисления. В частном случае эта операция может быть сведена к группированию символов (представление в виде триад и тетрад) или представлению в виде символов позиционной системы счисления. Так как любая позиционная система счисления не несет в себе избыточности информации, и все кодовые комбинации являются разрешенными, то использовать такие системы для контроля не представляется возможным.
- **Систематический код –** код, содержащий в себе кроме информационных контрольные разряды.
- В контрольные разряды записывается некоторая информация об исходном числе. Поэтому можно говорить, что систематический код обладает избыточностью. При этом абсолютная избыточность будет выражаться количеством контрольных разрядов k, а относительная избыточность – отношением k/n, где n=m+k – общее количество разрядов в кодовом слове (m – количество информационных разрядов).
- Понятие корректирующей способности кода обычно связывают с возможностью с возможностью обнаружения и исправления ошибки. Количественно корректирующая способность кода определяется вероятностью обнаружения или исправления ошибки. Если имеем n- разрядный код и вероятность искажения одного символа будет P, то вероятность того, что искажены k символов, а остальные n-k символов не искажены, по теореме умножения вероятностей будет
    - $W=P^k(1-P)^{n-k}$.
- Число кодовых комбинаций, каждая из которых содержит k искаженных элементов, равна числу сочетаний из n по k:
- Тогда вероятность искажения
- Так как на практике $P=10^{-3}\div10^{-4}$, наибольший вес в сумме вероятностей имеет вероятность искажения одного символа. Следовательно, основное внимание нужно обратить на обнаружение и исправление одиночной ошибки.
- Корректирующая способность кода связана также с понятием кодового расстояния.

- **Кодовое расстояние** d(A,B) **кодовых комбинаций** A и B определяется как вес такой третьей кодовой комбинации, которая получается сложением исходных комбинаций по модулю 2.
- **Вес кодовой комбинации** V(A) – количество единиц, содержащихся в кодовой комбинации.
- Коды можно рассматривать и как некоторые геометрические (пространственные) фигуры. Например, триаду можно представить в виде единичного куба, имеющего координаты вершин, которые отвечают двоичным символам в этом случае кодовое расстояние воспринимается как сумма длин ребер между соответствующими вершинами куба (принято, что длина одного ребра равна 1). Оказывается, что любая позиционная система отличается тем свойством, что минимальное кодовое расстояние равно 1.
- В теории кодирования показано, что систематический код обладает способностью обнаружить ошибки только тогда, когда минимальное кодовое расстояние для него больше или равно 2t, т.е.

$$d_{\min} \geq 2t,$$

- , где t – кратность обнаруживаемых ошибок t=1 (в случае обнаружения одиночных ошибок t=1). Это означает, что между соседними кодовыми комбинациями должна существовать по крайней мере одна кодовая комбинация
- В случае если необходимо не только обнаруживать, но и исправлять ошибку (указать место ошибки), минимальное кодовое расстояние должно быть

$$d_{\min} \geq 2t + 1.$$

Обнаружение и исправление одиночных ошибок путем использования дополнительных разрядов

- Рассмотрим возможность использования дополнительных (контрольных) разрядов для обнаружения и исправления ошибок. Эта возможность заключается в том, что к *n* информационных разрядов добавляется один контрольный разряд. В него записывается 0 или 1 таким образом, чтобы для каждого из передаваемых чисел сумма его разрядов по модулю 2 была бы равна нулю (кодирование по методу четности) или единице (нечетности). Появление ошибки в числе обнаружится по нарушению четности или нечетности. При этом виде кодирования допускается возможность выявления только одиночной ошибки. Чтобы одна комбинация разрядов числа превратилась в другую без выявления ошибки, необходимо изменение четного (2, 4, 6 и т. д.) числа разрядов одновременно. Пример реализации метода контроля по методу четности-нечетности приведен в табл. 5.

Таблица 5

Число	Контрольный разряд	Проверка (нечетности)
11011011	1	0
01101101	1	1 – ошибка
11010101	0	0
10101001	1	0
01010111	0	0

- Рассмотренный способ контроля по методу четности-нечетности может быть видоизменен для выявления места ошибки в числе. Длинное число разбивается на группы разрядов, каждая из которых содержит *к* разрядов.
- Контрольные разряды выделяются всем группам по строкам и по столбцам согласно следующей схеме:

a_1	a_2	a_3	a_4	a_5	k_1
a_6	a_7	a_8	a_9	a_{10}	k_2
a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	k_3
a_{16}	a_{17}	a_{18}	a_{19}	a_{20}	k_4
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	k_5
k_6	k_7	k_8	k_9	k_{10}	

- Если ошибка произошла в разряде *as* (единица изменилась на нуль или наоборот), то при проверке на четность (нечетность) сумма по i-й строке и j-му столбцу, на пересечении которых находится элемент a_s, изменится. Следовательно, можно зафиксировать нарушение четности (нечетности) по этой строке и столбцу. Это не только позволит обнаружить ошибку, но и локализовать ее место. Изменив значение разряда a_s на противоположное, можно исправить возникшую ошибку.
- Контроль по методу четности-нечетности используется для контроля записи и считывания информации, а также для выполнения арифметических операций.

Код Хэ́мминга

- **Код Хэ́мминга** — вероятно, наиболее известный из первых самоконтролирующихся и самокорректирующихся кодов. Построен применительно к двоичной системе счисления. Позволяет исправлять одиночную ошибку (ошибка в одном бите) и находить двойную
- История
- В середине 1940-х годов в лаборатории фирмы Белл (Bell Labs) была создана счётная машина Bell Model V. Это была электромеханическая машина, использующая релейные блоки, скорость которых была очень низка: один оборот за несколько секунд. Данные вводились в машину с помощью перфокарт, поэтому в процессе чтения часто происходили ошибки. В рабочие дни использовались специальные коды, чтобы обнаруживать и исправлять найденные ошибки, при этом оператор узнавал об ошибке по свечению лампочек, исправлял и снова запускал машину. В выходные дни, когда не было операторов, при возникновении ошибки машина автоматически выходила из программы и запускала другую.
- Хэмминг часто работал в выходные дни, и все больше и больше раздражался, потому что часто должен был перезагружать свою программу из-за ненадежности перфокарт. На протяжении нескольких лет он проводил много времени над построением эффективных алгоритмов исправления ошибок. В 1950 году он опубликовал способ, который известен как код Хэмминга.

Самоконтролирующиеся коды

- Коды Хэмминга являются самоконтролирующимися кодами, то есть кодами, позволяющими автоматически обнаруживать ошибки при передаче данных. Для их построения достаточно приписать к каждому слову один добавочный (контрольный) двоичный разряд и выбрать цифру этого разряда так, чтобы общее количество единиц в изображении любого числа было, например, нечетным. Одиночная ошибка в каком-либо разряде передаваемого слова (в том числе, может быть, и в контрольном разряде) изменит четность общего количества единиц. Счетчики по модулю 2, подсчитывающие количество единиц, которые содержатся среди двоичных цифр числа, могут давать сигнал о наличии ошибок.
- При этом невозможно узнать, в каком именно разряде произошла ошибка, и, следовательно, нет возможности исправить её. Остаются незамеченными также ошибки, возникающие одновременно в двух, четырёх, и т. д. — в четном количестве разрядов. Впрочем, двойные, а тем более четырёхкратные ошибки полагаются маловероятными.
- Коды, в которых возможно автоматическое исправление ошибок, называются самокорректирующимися. Для построения самокорректирующегося кода, рассчитанного на исправление одиночных ошибок, одного контрольного разряда недостаточно. Как видно из дальнейшего, количество контрольных разрядов k должно быть выбрано так, чтобы удовлетворялось неравенство
- $$2^k \geq k + m + 1$$
- или
$$k \geq \log_2(k + m + 1)$$
- где m — количество основных двоичных разрядов кодового слова.
- Минимальные значения k при заданных значениях m, найденные
- в соответствии с этим неравенством, приведены в таблице.

Диапазон m	k_{min}
1	2
2-4	3
5-11	4
12-26	5
27-57	6

- В настоящее время наибольший интерес представляют двоичные блочные корректирующие коды. При использовании таких кодов информация передаётся в виде блоков одинаковой длины и каждый блок кодируется и декодируется независимо друг от друга. Почти во всех блочных кодах символы можно разделить на информационные и проверочные. Таким образом, все комбинации кодов разделяются на разрешенные (для которых соотношение информационных и проверочных символов возможно) и запрещенные.
- Основными характеристиками самокорректирующихся кодов являются:
- 1. Число разрешенных и запрещенных комбинаций. Если n — число символов в блоке, r — число проверочных символов в блоке, k — число информационных символов, то 2^n — число возможных кодовых комбинаций, 2^k — число разрешенных кодовых комбинаций, $2^n - 2^k$ — число запрещенных комбинаций.
- 2. Избыточность кода. Величину k/n называют избыточностью корректирующего кода.
- 3. Минимальное кодовое расстояние. Минимальным кодовым расстоянием d называется минимальное число искаженных символов, необходимое для перехода одной разрешенной комбинации в другую.
- 4. Число обнаруживаемых и исправляемых ошибок. Если g — количество ошибок, которое код способен исправить, то необходимо и достаточно, чтобы
- $$d \geq 2g + 1$$
- 5. Корректирующие возможности кодов.
- Граница Плоткина даёт верхнюю границу кодового расстояния
- $d \leqslant \frac{n \cdot 2^{k-1}}{2^k - 1}$ или $r \geq 2 \cdot (d-1) - \log_2 d$ при $n \geq 2 \cdot d - 1$
- Есть еще Граница Хемминга устанавливает максимально возможное число разрешенных кодовых комбинаций и Граница Варшамова — Гилберта для больших n определяет нижнюю границу числа проверочных символов.
- Все вышеперечисленные оценки дают представление о **верхней границе** d при фиксированных n и k или **оценку снизу** числа проверочных символов

Код Хэмминга

- Построение кодов Хэмминга основано на принципе проверки на четность числа единичных символов: к последовательности добавляется такой элемент, чтобы число единичных символов в получившейся последовательности было четным.

$$r_1 = i_1 \oplus i_2 \oplus \ldots \oplus i_k. \qquad S = i_1 \oplus i_2 \oplus \ldots \oplus i_n \oplus r_1 .$$

- S=0 — ошибки нет, S=1 - однократная ошибка.
- Такой код называется (k+1, k) или (n, n-1) . Первое число — количество элементов последовательности, второе — количество информационных символов.
- Для каждого числа проверочных символов r = 3, 4, 5, … существует классический код Хэмминга с маркировкой (n,k) = (2^r – 1, 2^r – 1 - r) то есть — (7,4), (15,11), (31,26) . При иных значениях k получается так называемый усеченный код, например международный телеграфный код МТК-2, у которого k=5 . Для него необходим код Хэмминга (9,5) , который является усеченным от классического (15,11), .

- Для примера рассмотрим классический код Хемминга (7,4). Сгруппируем проверочные символы следующим образом:

$$r_1 = i_1 \oplus i_2 \oplus i_3$$
$$r_2 = i_2 \oplus i_3 \oplus i_4$$
$$r_3 = i_1 \oplus i_2 \oplus i_4$$

- Получение кодового слова выглядит следующим образом:

$$\begin{pmatrix} i_1 & i_2 & i_3 & i_4 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} i_1 & i_2 & i_3 & i_4 & r_1 & r_2 & r_3 \end{pmatrix}$$

- На вход декодера поступает кодовое слово $V = (i'_1, i'_2, i'_3, i'_4, r'_1, r'_2, r'_3)$
- где штрихом помечены символы, которые могут исказиться в результате помехи. В декодере в режиме исправления ошибок строится последовательность синдромов:

$$S_1 = r_1 \oplus i_1 \oplus i_2 \oplus i_3$$
$$S_2 = r_2 \oplus i_2 \oplus i_3 \oplus i_4$$
$$S_3 = r_3 \oplus i_1 \oplus i_2 \oplus i_4$$

- $S=(S_1, S_2, S_3)$называется синдромом последовательности.

- Получение синдрома выглядит следующим образом:

$$(i_1 \quad i_2 \quad i_3 \quad i_4 \quad r_1 \quad r_2 \quad r_3) \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = (S_1 \quad S_2 \quad S_3)$$

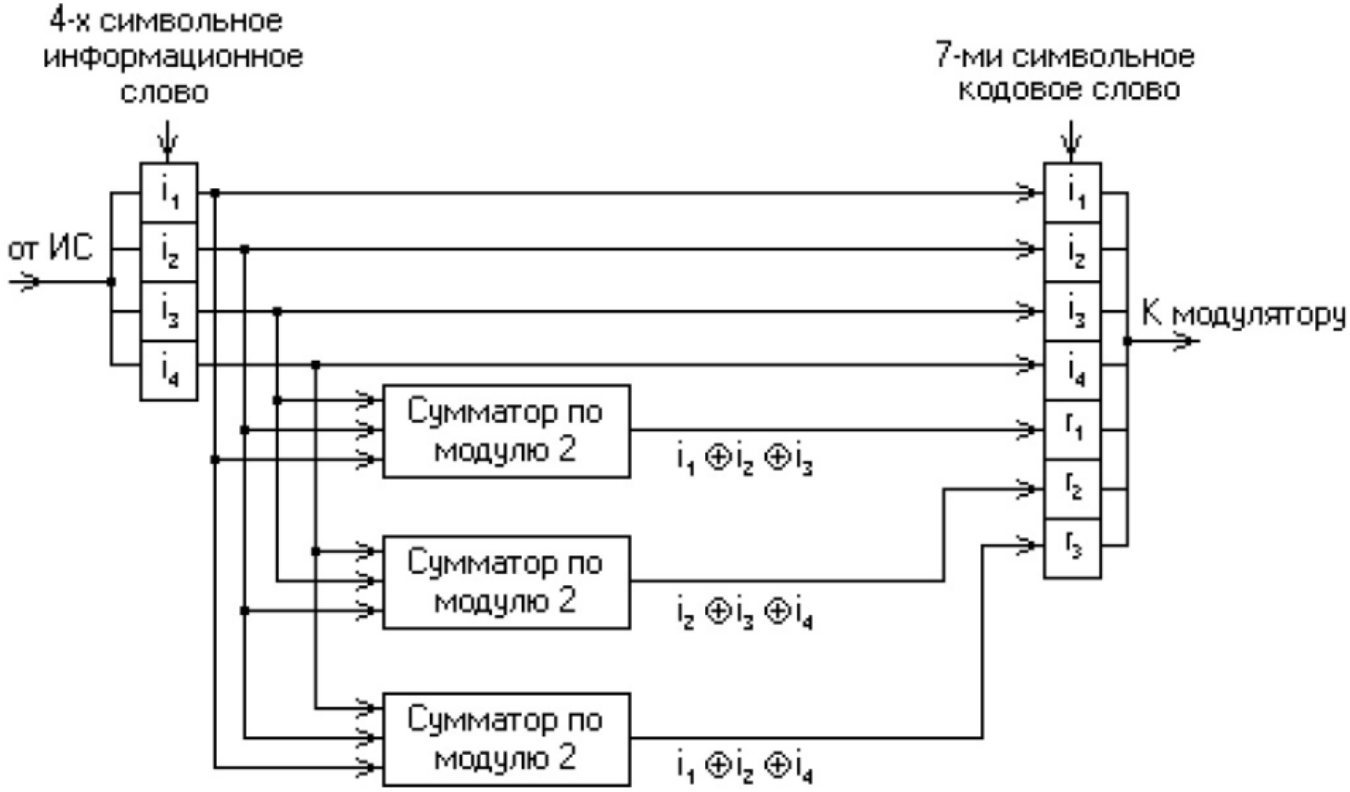
- Кодовые слова кода Хэмминга

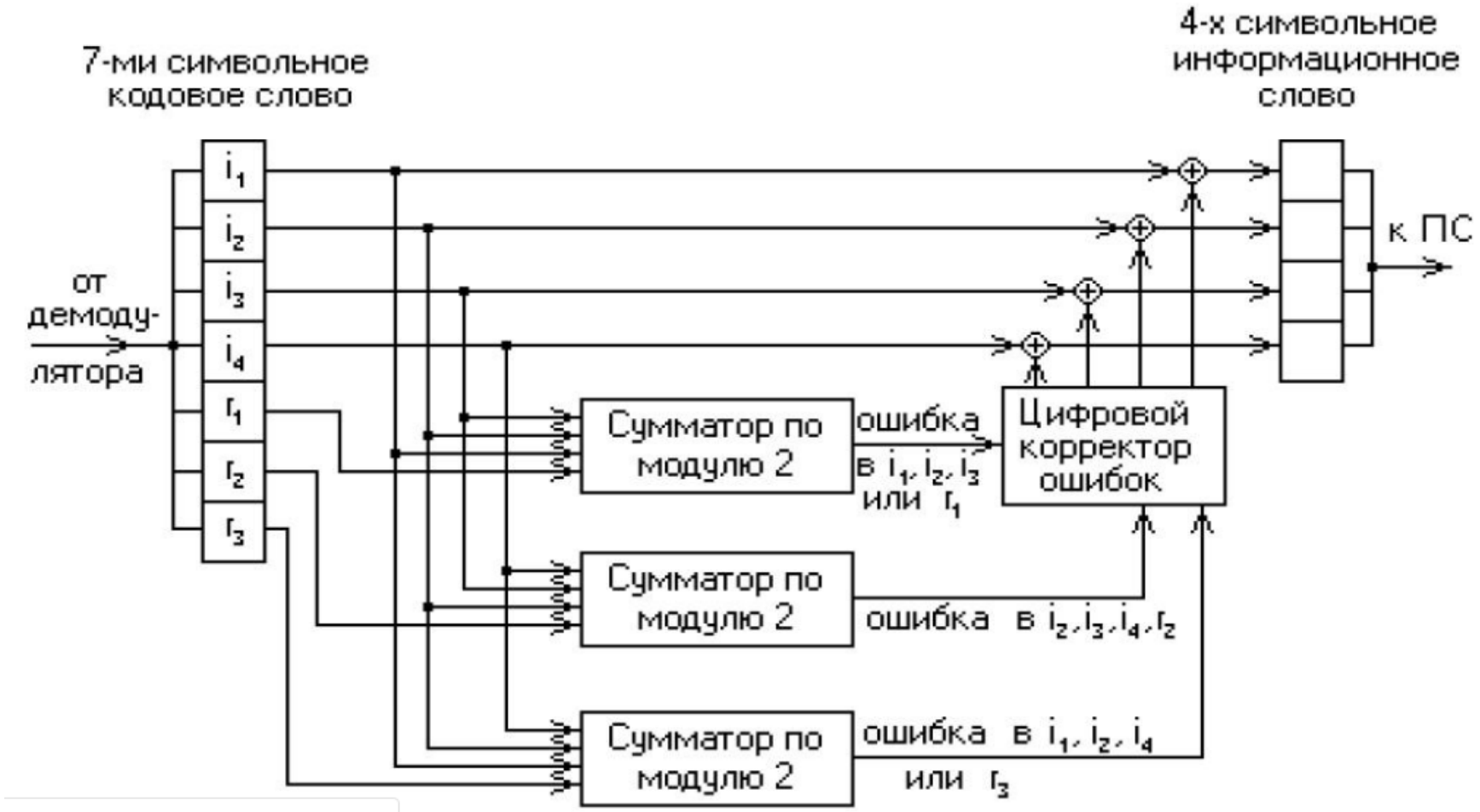
- Синдром (0,0,0) указывает на то, что в
- последовательности нет искажений. Каждому
- ненулевому синдрому соответствует
- определенная конфигурация ошибок, которая
- исправляется на этапе декодирования.

i_1	i_2	i_3	i_4	r_1	r_2	r_3
0	0	0	0	0	0	0
0	0	0	1	0	1	1
0	0	1	0	1	1	0
0	0	1	1	1	0	1
0	1	0	0	1	1	1
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	0	1	1	1	0
1	0	1	0	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	1	0
1	1	0	1	0	0	1
1	1	1	0	1	0	0
1	1	1	1	1	1	1

- Для кода (7,4) в таблице указаны ненулевые синдромы и соответствующие им конфигурации ошибок (для вида: $i_1, i_2, i_3, i_4, r_1, r_2, r_3$,).

Синдром	001	010	011	100	101	110	111
Конфигурация ошибок	0000001	0000010	0001000	0000100	1000000	0010000	0100000
Ошибка в символе	r_3	r_2	i_4	r_1	i_1	i_3	i_2




7-ми символьное
кодовое слово
4-х символьное
информационное
слово
от
демоду-
лятора
i_1
i_2
i_3
i_4
r_1
r_2
r_3
к ПС
Сумматор по
модулю 2
ошибка
в i_1, i_2, i_3
или r_1
Цифровой
корректор
ошибок
Сумматор по
модулю 2
ошибка в i_2, i_3, i_4, r_2
Сумматор по
модулю 2
ошибка в i_1, i_2, i_4
или r_3

Алгоритм кодирования

- Предположим, что нужно сгенерировать код Хэмминга для некоторого информационного кодового слова. В качестве примера возьмём 15-битовое кодовое слово **x**1…**x**15, хотя алгоритм пригоден для кодовых слов любой длины. В приведённой ниже таблице в первой строке даны

ие

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
1	0	0	1	0	0	1	0	1	1	1	0	0	0	1

- Вставим в информационное слово контрольные биты **r**0…**r**4 таким образом, чтобы номера их позиций представляли собой целые степени двойки: 1, 2, 4, 8, 16… Получим 20-разрядное слово с 15 информационными и 5 контрольными битами.

- Первоначально контрольные биты устанавливаем равными нулю. На рисунке контрольные биты выделены розовым цветом.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
r_0	r_1	x_1	r_2	x_2	x_3	x_4	r_3	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	r_4	x_{12}	x_{13}	x_{14}	x_{15}
0	0	1	0	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	1

- В общем случае количество контрольных бит в кодовом слове равно двоичному логарифму числа, на единицу большего, чем количество бит кодового слова (включая контрольные биты); логарифм округляется в большую сторону. Например, информационное слово длиной 1 бит требует двух контрольных разрядов, 2-, 3- или 4-битовое информационное слово — трёх, 5…11-битовое — четырёх, 12…26-битовое — пяти и т. д.
- Добавим к таблице 5 строк (по количеству контрольных битов), в которые поместим матрицу преобразования. Каждая строка будет соответствовать одному контрольному биту (нулевой контрольный бит — верхняя строка, четвёртый — нижняя), каждый столбец — одному биту кодируемого слова. В каждом столбце матрицы преобразования поместим двоичный номер этого столбца, причём порядок следования битов будет обратный — младший бит расположим в верхней строке, старший — в нижней. Например, в третьем столбце матрицы будут стоять числа 11000, что соответствует двоичной записи числа три: 00011.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
r_0	r_1	x_1	r_2	x_2	x_3	x_4	r_3	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	r_4	x_{12}	x_{13}	x_{14}	x_{15}	
0	0	1	0	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	1	
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	r_0
0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	r_1
0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	r_2
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	r_3
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	r_4

- В правой части таблицы мы оставили пустым один столбец, в который поместим результаты вычислений контрольных битов. Вычисление контрольных битов производим следующим образом. Берём одну из строк матрицы преобразования (например, r0) и находим её скалярное произведение с кодовым словом, то есть перемножаем соответствующие биты обеих строк и находим сумму произведений. Если сумма получилась больше единицы, находим остаток от его деления на 2. Иными словами, мы подсчитываем сколько раз в кодовом слове и соответствующей строке матрицы в одинаковых позициях стоят единицы и берём это число по модулю 2.
- Если описывать этот процесс в терминах матричной алгебры, то операция представляет собой перемножение матрицы преобразования на матрицу-столбец кодового слова, в результате чего получается матрица-столбец контрольных разрядов, которые нужно взять по модулю 2.
- Например, для строки r0:
- $r_0 = (1 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 1 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 1)$
- $\bmod 2 = 5 \bmod 2 = 1$.

- Полученные контрольные биты вставляем в кодовое слово вместо стоявших там ранее нулей. По аналогии находим проверочные биты в остальных строках. Кодирование по Хэммингу завершено. Полученное кодовое слово — 11110010001011110001.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		
r_0	r_1	x_1	r_2	x_2	x_3	x_4	r_3	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	r_4	x_{12}	x_{13}	x_{14}	x_{15}		
0	0	1	0	0	0	1	0	0	0	1	0	1	1	1	0	0	0	0	1		
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	r_0	1
0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	r_1	1
0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	r_2	1
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	r_3	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	r_4	1

Алгоритм декодирования

- Алгоритм декодирования по Хэммингу абсолютно идентичен алгоритму кодирования. Матрица преобразования соответствующей размерности умножается на матрицу-столбец кодового слова и каждый элемент полученной матрицы-столбца берётся по модулю 2. Полученная матрица-столбец получила название «матрица синдромов». Легко проверить, что кодовое слово, сформированное в соответствии с алгоритмом, описанным в предыдущем разделе, всегда даёт нулевую матрицу синдромов.
- Матрица синдромов становится ненулевой, если в результате ошибки (например, при передаче слова по линии связи с шумами) один из битов исходного слова изменил своё значение. Предположим для примера, что в кодовом слове, полученном в предыдущем разделе, шестой бит изменил своё значение с нуля на единицу (на рисунке обозначено красным цветом). Тогда получим следующую матрицу синдромов.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20		
r_0	r_1	x_1	r_2	x_2	x_3	x_4	r_3	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	r_4	x_{12}	x_{13}	x_{14}	x_{15}		
1	1	1	1	0	1	1	0	0	0	1	0	1	1	1	1	0	0	0	1		
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	s_0	0
0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	s_1	1
0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	s_2	1
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	s_3	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	s_4	0

- Заметим, что при однократной ошибке матрица синдромов всегда представляет собой двоичную запись (младший разряд в верхней строке) номера позиции, в которой произошла ошибка. В приведённом примере матрица синдромов (01100) соответствует двоичному числу 00110 или десятичному 6, откуда следует, что ошибка произошла в шестом бите.

Применение

- Код Хэмминга используется в некоторых прикладных программах в области хранения данных, особенно в RAID 2; кроме того, метод Хэмминга давно применяется в памяти типа ECC и позволяет «на лету» исправлять однократные и обнаруживать двукратные ошибки.
- **RAID** (англ. *Redundant Array of Independent Disks — избыточный массив независимых дисков*) — технология виртуализации данных, которая объединяет несколько дисков в логический элемент для избыточности и повышения производительности.
- **ECC-память** (англ. *error-correcting code memory*, память с коррекцией ошибок) — тип компьютерной памяти, которая автоматически распознаёт и исправляет спонтанно возникшие изменения (ошибки) битов памяти. Память не поддерживающая коррекцию ошибок, обозначается **non-ECC**.

Языки описания аппаратуры

- До сих пор мы рассматривали разработку комбинационных и последовательностных цифровых схем на уровне схемотехники. Процесс поиска наилучшего набора логических элементов для выполнения данной логической функции трудоемок и чреват ошибками, так как требует упрощения логических таблиц или выражений и перевода конечных автоматов в вентили вручную. В 1990-е годы разработчики обнаружили, что их производительность труда резко возрастала, если они работали на более высоком уровне абстракции, определяя только логическую функцию и предоставляя создание оптимизированных логических элементов системе автоматического проектирования (САПР). Два основных языка описания аппаратуры (Hardware Description Language, HDL) – SystemVerilog и VHDL.
- SystemVerilog и VHDL построены на похожих принципах, но их синтаксис весьма различается. Их обсуждение в этой главе разделено на две колонки для сравнения, где SystemVerilog будет слева, а VHDL – справа. При первом чтении сосредоточьтесь на одном из языков. Как только вы разберетесь с одним, при необходимости вы сможете быстро усвоить другой. В последующих главах показана аппаратура и в схематическом виде и в форме HDL-модели. Если вы предпочтете пропустить эту главу и не изучать языки описания цифровой аппаратуры, вы тем не менее сможете постичь принципы архитектуры микропроцессоров на уровне схем. Однако, подавляющее большинство коммерческих систем сейчас строится с использованием языков описания цифровой аппаратуры, а не на уровне схемотехники. Если вы когда-либо в вашей карьере собираетесь заниматься разработкой цифровых схем, мы настоятельно рекомендуем вам выучить один из языков описания аппаратуры.

Модули

- Блок цифровой аппаратуры, имеющий входы и выходы, называется модулем. Логический элемент “И”, мультиплексор и схема приоритетов являются примерами модулей цифровой аппаратуры. Есть два общепринятых типа описания функциональности модуля – поведенческий и структурный. Поведенческая модель описывает, что модуль делает. Структурная модель описывает то, как построен модуль из простых элементов, с применением принципа иерархии. Код на SystemVerilog и VHDL из примера 4.1 показывает поведенческое описание модуля, который рассчитывает булеву функцию. На обоих языках модуль назван sillyfunction и имеет 3 входа, a, b и c и один выход y, и, как и следовало ожидать, следует принципу модульности. Он имеет полностью определенный интерфейс, состоящий из его входов и выходов, и выполняет определенную функцию. Конкретный способ, которым модуль был описан, неважен для тех, кто будет использовать модуль в будущем, поскольку модуль выполняет свою функцию.

$$\overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + A\overline{B}C$$

SystemVerilog

```
module sillyfunction(input logic a, b, c, output logic y);
  assign y = ~a & ~b & ~c |
             a & ~b & ~c |
             a & ~b & c;
endmodule
```

- Модуль на SystemVerilog начинается с имени модуля и списка входов и выходов. Оператор assign описывает комбинационную логику. Тильда (~) означает НЕ, амперсанд (&) – И, а вертикальная черта (|) – ИЛИ. Сигналы типа logic, как входы и выходы в примере – логические переменные, принимающие значения 0 или 1. Они также могут принимать плавающее и неопределенное значения. Тип logic появился в SystemVerilog. Он введен для замены типа reg. Тип logic стоит использовать везде, кроме описания сигналов с несколькими источниками. Такие сигналы называются цепями (net)

- Код на VHDL состоит из трех частей: объявления используемых библиотек и внешних объектов (library, use), объявления интерфейса объекта (entity) и его внутренней структуры (architecture). В объявлении интерфейса указывается имя модуля и перечисляются его входы и выходы. Блок architecture определяет, что модуль делает. У сигналов в VHDL, в том числе входов и выходов, должен быть указан тип. Цифровые сигналы стоит объявлять как STD_LOGIC. Сигналы этого типа принимают значения ‘0’ или ‘1’, а также плавающее и неопределенное значения. Тип STD_LOGIC определен в библиотеке IEEE.STD_LOGIC_1164, поэтому библиотеку объявлять обязательно. VHDL не определяет соотношение приоритетов операций AND и OR, поэтому при записи логических выражений нужно всегда использовать скобки.

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;
entity sillyfunction is
  port(a, b, c: in  STD_LOGIC;
       y:       out STD_LOGIC);
 end;
 architecture synth of sillyfunction is
 begin
  y <= (not a and not b and not c) or
       (a and not b and not c) or
       (a and not b and c);
 end;
```

$\overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + A\overline{B}C$

Происхождение языков SystemVerilog и VHDL

- Примерно в половине вузов, где преподают цифровую схемотехнику, читают VHDL, а в оставшейся половине – Verilog. В промышленности склоняются к SystemVerilog, но много компаний еще используют VHDL, поэтому многим разработчикам нужно владеть обоими языками. По сравнению с SystemVerilog, VHDL более многословный и громоздкий, чем можно было бы ожидать от языка, разработанного комитетом («Верблюд – это лошадь, разработанная комитетом» – американская шутка. – прим. перев.)
- SystemVerilog
- Верилог был разработан компанией Gateway Design Automation в 1984 году как фирменный язык для симуляции логических схем. В 1989 году Gateway приобрела компания Cadence, и Verilog стал открытым стандартом в 1990 году под управлением сообщества Open Verilog International. Язык стал стандартом IEEE (прим. переводчика: институт инженеров по электротехнике и электронике (IEEE) – профессиональное сообщество, ответственное за многие компьютерные стандарты, например, Wi-Fi (802.11), Ethernet (802.3), и чисел с плавающей точкой (754)).в 1995 году. В 2005 году язык был расширен для упорядочивания и лучшей поддержки моделирования и верификации систем. Эти расширения были объединены в единый стандарт, который сейчас называется SystemVerilog(стандарт IEEE 1800-2009). Файлы языка SystemVerilog обычно имеют расширение .sv.

VHDL

- Аббревиатура VHDL расшифровывается как VHSIC Hardware Description Language. VHSIC, в свою очередь, происходит от сокращения Very High Speed Integrated Circuits – названия программы министерства обороны США. Разработка VHDL был начата в 1981 году министерством обороны для описания структуры и функциональности электронных схем. За основу для разработки был взят язык программирования ADA. Изначальной целью языка была документация, но затем он был быстро адаптирован для симуляции и синтеза. IEEE стандартизировал его в 1987 году, и после этого язык обновлялся несколько раз. Эта глава основана на редакции VHDL 2008 года (стандарт IEEE 1076-2008), которая упорядочивает язык во многих аспектах. На момент написания не все функции стандарта VHDL 2008 года поддерживаются в САПР; эта глава только используют те функции, которые поддерживаются в Synplicity, Altera Quartus и Modelsim. Файл языка VHDL имеет расширение .vhd . На обоих языках можно полностью описать любую электронную систему, но у каждого языка есть свои особенности. Лучше использовать язык, который уже распространен в вашей организации или тот, которого требуют ваши клиенты. Большинство САПР сейчас позволяют смешивать языки, поэтому разные модули могут быть написаны на разных языках.

Симуляция и Синтез

- Две основные цели HDL – логическая симуляция и синтез. Во время симуляции на входы модуля подаются некоторые воздействия и проверяются выходы, чтобы убедиться, что модуль функционирует корректно. Во время синтеза текстовое описание модуля преобразуется в логические элементы.
    - Симуляция
- Люди регулярно совершают ошибки. Ошибки в цифровой аппаратуре называют багами. Ясно, что устранение багов в цифровой системе
- очень важно, особенно когда от правильной работы аппаратуры зависят чьи-то жизни. Тестирование системы в лаборатории весьма трудоёмко. Исследовать причины ошибок в лаборатории может быть очень сложно, так как наблюдать можно только сигналы, подключенные к контактам чипа, а то, что происходит внутри чипа, напрямую наблюдать невозможно. Исправление ошибок уже после того, как система была выпущена, может быть очень дорого. Например, исправление одной ошибки в новейших интегральных микросхемах стоит больше миллиона долларов и занимает несколько месяцев. Печально известный баг в команде деления с плавающей точкой (FDIV) в процессоре Pentium вынудил корпорацию Intel отозвать чипы после того, как они были поставлены заказчикам, что стоило им 475 миллионов долларов. Логическая симуляция необходима для тестирования системы до того, как она будет выпущена.

- Рис. 4.1 показывает графики сигналов из симуляции предыдущего модуля sillyfunction, демонстрирующие, что модуль работает корректно. (Прим. переводчика: симуляция была проведена в программе ModelSim PE Student Edition версии 10.0c. Modelsim был выбран, так как он используется коммерчески и имеет студенческую версию с возможностью бесплатной симуляции до 10 тыс. строк кода). Y есть лог.1, когда a,b,c есть 000, 100, или 101, как и указано в логическом выражении

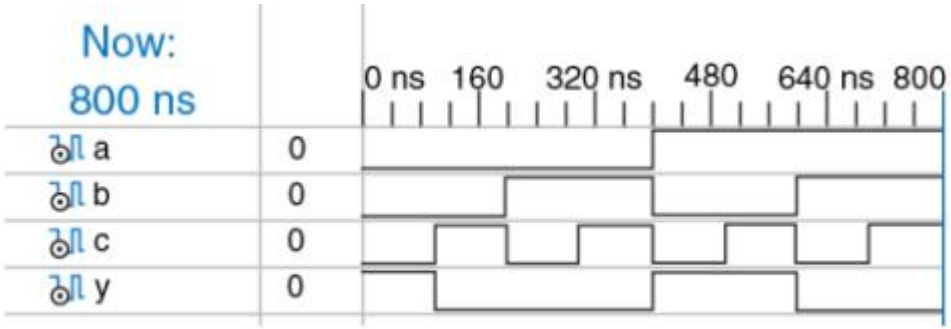


Рис. 4.1 Графики сигналов

Синтез

- Логический синтез преобразует код на HDL в нетлист, описывающий цифровую аппаратуру (т.е. логические элементы и соединяющие их проводники). Логический синтезатор может выполнять оптимизацию для сокращения количества необходимых элементов. Нетлист может быть текстовым файлом или нарисован в виде схемы, чтобы было легче визуализировать систему. Рис. 4.2 показывает результаты синтеза модуля sillyfunction (прим. переводчика: синтез был сделан с помощью программы Synplify Premier от Synplicity. Этот САПР был выбран, так как он является лидирующим коммерческим продуктом для синтеза HDL в программируемые логические интегральные схемы (смотри раздел 5.6.2) и так как он доступен по цене и дешев для использования в университетах). Обратите внимание, что , как три трех входовых элемента И упрощены в 2 двухвходовых элемента И, как мы обнаружили в примере 2.6, используя булеву алгебру.

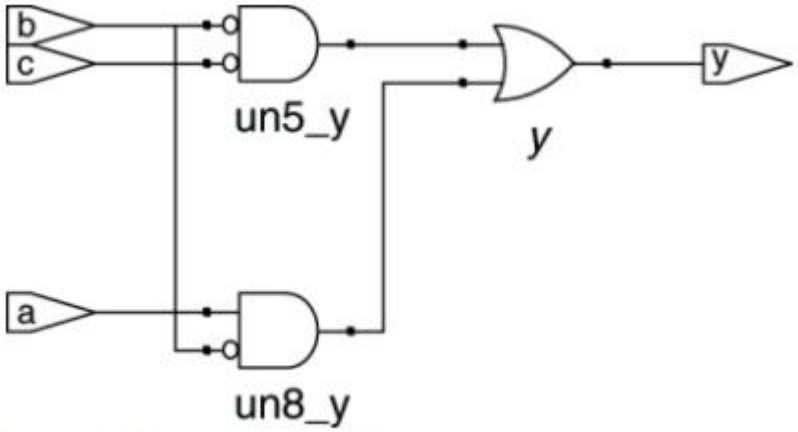
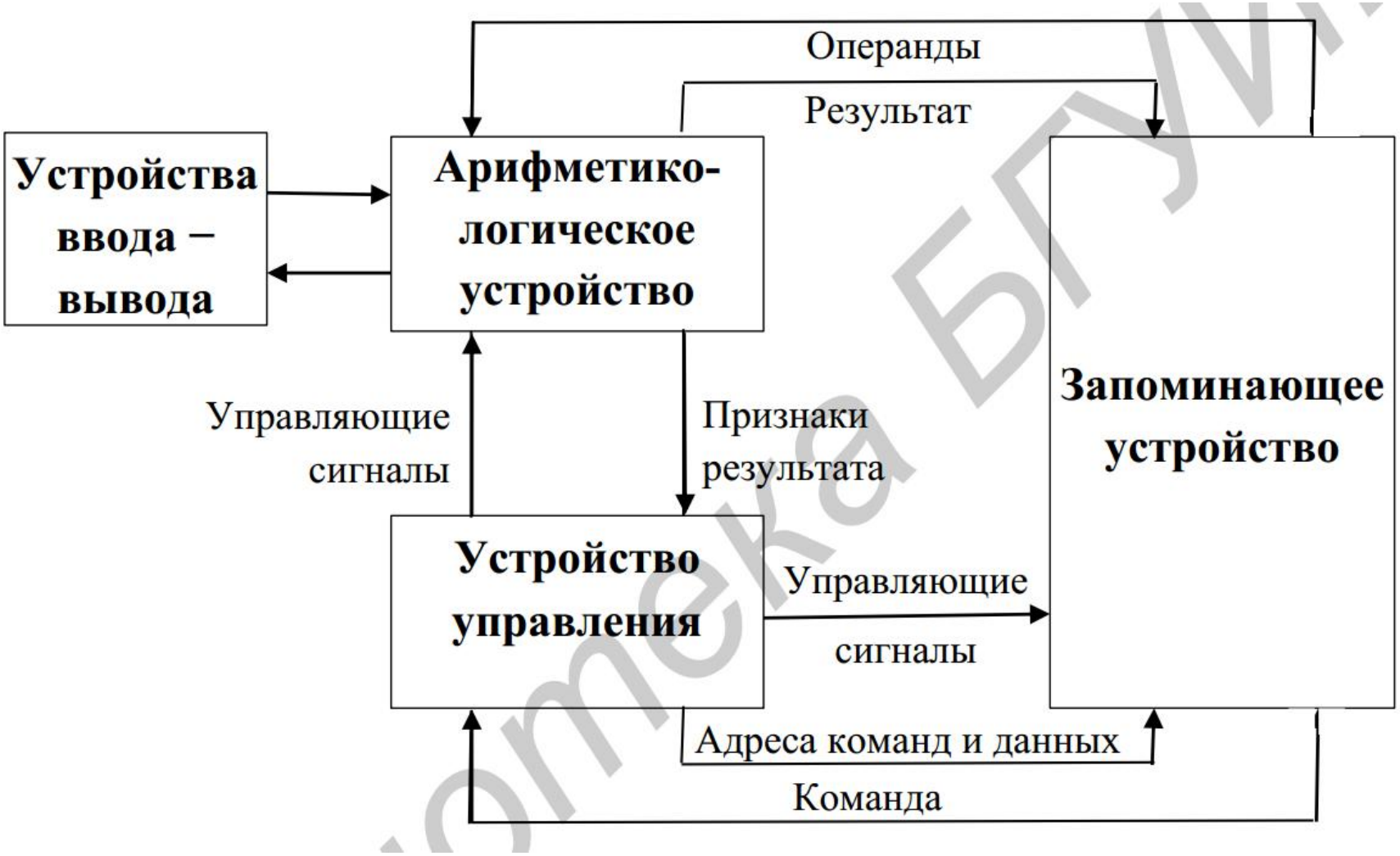


Рис. 4.2 Схема sillyfunction

- Описание схем на HDL напоминает программный код. Однако вы должны помнить, что ваш код предназначен для описания аппаратуры. SystemVerilog и VHDL – сложные языки со множеством операторов. Не все из них могут быть синтезированы в аппаратуре: например, оператор вывода результатов на экран во время симуляции не превращается в цифровую схему. Так как наша основная задача – создание цифровой схемы, мы акцентируем свое внимание на синтезируемом подмножестве языков. Точнее, мы будем делить код на HDL на синтезируемые модули и среду тестирования. Синтезируемые модули описывают цифровую схему. Среда тестирования содержит код, который подает воздействия на входы модуля и проверяет правильность значений его выходов, а также выводит несоответствия между ожидаемыми и действительными значениями. Код среды тестирования предназначается только для симуляции и не может быть синтезирован. Одна из главных ошибок начинающих заключается в том, что они думают о коде на HDL как о компьютерной программе, а не как о подспорье для описания цифровой аппаратуры. Если вы не представляете, хотя бы примерно, во что должен синтезироваться ваш код на HDL, то, скорее всего, результат вам не понравится. Ваша цифровая схема может получиться гораздо больше, чем нужно, или может оказаться, что ваш код симулируется правильно, но не может быть реализован в аппаратуре. Вместо этого, вы должны думать над вашей разработкой в понятиях комбинационной логики, регистров и конечных автоматов. Нарисуйте эти блоки на бумаге и покажите, как они будут подключены до того, как вы начнете писать код.

Структура классической ЭВМ



IEEE 754-2019

Стандарт IEEE для арифметики с плавающей запятой (IEEE 754) — это технический стандарт для арифметики с плавающей запятой, установленный в 1985 году Институтом инженеров по электротехнике и электронике (IEEE). Стандарт решил многие проблемы, обнаруженные в различных реализациях с плавающей запятой, которые затрудняли их надежное и переносимое использование.
Используется в программных (компиляторы разных языков программирования) и аппаратных (CPU и FPU) реализациях арифметических действий (математических операций).

Стандарт определяет:

арифметические форматы: наборы двоичных и десятичных данных с плавающей запятой, которые состоят из конечных чисел (включая нули со знаком и субнормальные числа), бесконечности и специальных значений «не числа» (NaN)
форматы обмена: кодировки (битовые строки), которые могут использоваться для обмена данными с плавающей запятой в эффективной и компактной форме.
правила округления: свойства, которые должны выполняться при округлении чисел во время арифметических операций и преобразований.
операции: арифметические и другие операции (например, тригонометрические функции) над арифметическими форматами.
обработка исключений: индикация исключительных условий (таких как деление на ноль, переполнение и т. д.)

Разработка стандарта

IEEE 754-2008, опубликованный в августе 2008 г., включает почти весь исходный стандарт IEEE 754-1985, а также стандарт IEEE 854-1987 для независимой от системы счисления арифметики с плавающей запятой. Текущая версия IEEE 754-2019 была опубликована в июле 2019 г. Это дороботаная версия предыдущей версии, включающая в основном уточнения, исправления дефектов и новые рекомендуемые операции.
Текущая версия IEEE 754—2008 была опубликована в 2008 году.
Международный стандарт ISO/IEC/IEEE 60559:2011 (с идентичным IEEE 754—2008) был одобрен и опубликован для JTC1/SC 25 под соглашением ISO/IEEE PSDO.
Бинарные форматы в первоначальном стандарте включены в новый стандарт наряду с тремя новыми основными форматами (одним бинарным и двумя десятичными). Для того, чтобы соответствовать текущему стандарту, реализация должна реализовать по крайней мере один из основных форматов.

Стандартная разработка

Первый стандарт для арифметики с плавающей запятой, IEEE 754-1985, был опубликован в 1985 году. Он охватывал только двоичную арифметику с плавающей запятой.

Новая версия IEEE 754-2008 была опубликована в августе 2008 г. после семилетнего процесса пересмотра под председательством Дэна Зураса и под редакцией Майка Каулишоу. Он заменил как IEEE 754-1985 (двоичная арифметика с плавающей запятой), так и стандарт IEEE 854-1987 для независимой от системы счисления

арифметики с плавающей запятой. Двоичные форматы исходного стандарта включены в этот новый стандарт вместе с тремя новыми базовыми форматами, одним двоичным и двумя десятичными. Чтобы соответствовать текущему стандарту, реализация должна реализовать по крайней мере один из основных форматов как арифметический формат, так и формат обмена.

Международный стандарт ISO/IEC 60559:2020 (с содержанием, идентичным IEEE 754-2019) был одобрен для принятия через ISO/IEC JTC 1/SC 25 и опубликован. Следующая запланированная редакция стандарта запланирована на 2028 год.

Формат

Формат IEEE 754 представляет собой «совокупность представлений числовых значений и символов». Формат может также включать в себя способ кодирования.
Формат с плавающей запятой определяется
основание b (b может быть 2 или 10);
точность p;
диапазон показателей степени от emin до emax, где emin = 1 – emax для всех форматов IEEE 754.

Формат включает:
Числа, которые могут рассматриваться в двоичной или десятичной системе счисления. Вещественное число представляется тремя целыми числами s, c и q, где s — знак (0 для положительного и 1 для отрицательного), c — мантисса (коэффициент), имеющая не более p цифр при записи по основанию b (т. е. целое число в диапазоне от 0 до до $b^p - 1$), q — экспонента. Для заданных целых чисел s, c и q значением соответствующего вещественного числа является: $(-1)^s \times c \times b^q$.
Например, число с основанием 10, битом знака 1 (число отрицательное), мантиссой 12345 и экспонентой –3 определяют число
$(-1)^1 \times 12345\ 10^{-3} = -12.345$.
Положительный нуль +0 и отрицательный нуль –0.
Две бесконечности: $+\infty$ и $-\infty$.
Например, если b = 10, p = 7 и emax = 96, то emin = −95, мантиссы удовлетворяют условию $0 \le c \le 9999999$, а показатель степени удовлетворяет условию $-101 \le q \le 90$.

Два вида NaN (не число): тихий NaN (qNaN) и сигнальный NaN (sNaN).
Возможные конечные значения, которые могут быть представлены в формате, определяются основанием b, числом знаков в мантиссе (с точностью p) и максимальным значением E_{max}:
c должен быть целым числом в диапазоне от нуля до $b^p - 1$} (если b = 10 p = 7 тогда c может быть от 9999999)
q должно быть целым числом, чтобы $1 - E_{max} £ q + p - 1 £ E_{max}$ (если p = 7 и E_{max} =96, то q может быть от –101 до 90).
Следовательно, наименьший ненулевой положительное число, которое может быть представлено, равно 1×10^{-101}, а наибольшее — 9999999×10^{90} (9,999999 × 1096), поэтому полный диапазон чисел составляет от $-9{,}999999 \times 10^{96}$ до $9{,}999999 \times 10^{96}$.
Числа $-b^{-Emax}$ и b^{-Emax} (здесь -1×10^{-95} и 1×10^{-95}) являются наименьшими (по величине) нормальными числами; ненулевые числа между этими наименьшими числами называются субнормальными числами.

Представление и кодирование в памяти

Некоторые числа могут иметь несколько представлений в формате, в котором они были только что описаны. Например, если b = 10 и p = 7, то −12,345 может быть представлено как −12345 ×10, −123450 ×10 и −1234500 ×10. Однако для большинства операций, таких как арифметические операции, результат (значение) не зависит от представления входных данных.

Для десятичных форматов допустимо любое представление, и набор этих представлений называется когортой. Если результат может иметь несколько представлений, стандарт определяет, какой член когорты выбран.

Для двоичных форматов представление становится уникальным путем выбора наименьшей представимой экспоненты, позволяющей точно представить значение. Кроме того, показатель степени не представлен напрямую, но добавляется смещение, так что наименьший представимый показатель представлен как 1, а 0 используется для субнормальных чисел. Для чисел с показателем степени в нормальном диапазоне (поле показателя не может содержать ни все единицы, ни все нули), ведущий бит мантиссы всегда будет равен 1. Следовательно, ведущая единица может подразумеваться, а не присутствовать явно в кодировании памяти, и по стандарту явно представленная часть мантиссы будет находиться между 0 и 1. Это правило называется соглашением о начальных битах, неявным соглашением о битах или соглашением о скрытых битах. Это правило позволяет двоичному формату иметь дополнительную точность. Соглашение о начальных битах не может использоваться для субнормальных чисел, поскольку они имеют показатель степени за пределами диапазона нормального показателя и масштабируются по наименьшей представленной экспоненте, используемой для наименьших нормальных чисел.

Из-за возможности множественного кодирования (по крайней мере, в форматах, называемых форматами обмена), NaN может нести другую информацию: знаковый бит (который не имеет значения, но может использоваться некоторыми операциями) и полезные данные , который предназначен для диагностической информации, указывающей на источник NaN (но полезная нагрузка может иметь другое использование, например NaN-бокс).
Для десятичных форматов любое представление справедливо, и совокупность этих представлений называется когортами. Когда результат может иметь несколько представлений, стандарт определяет, какой выбран членом когорты.

Для бинарных форматов представление делается уникальным путём выбора наименьшего представляемого показателя. Для чисел с показателем в нормальном диапазоне (не все из них или все нули), ведущий бит мантиссы всегда будет равен 1. Следовательно, ведущий 1 бит может подразумеваться, а не сохраняться явно в памяти. Это правило называется ведущей битной конвенцией или скрытой битной конвенцией. Правило позволяет сберечь 1 бит памяти, чтобы иметь ещё один бит точности. Ведущий бит конвенции не используется для субнормальных чисел; их показатель находится за пределами нормального диапазона значений.

Базовые форматы и форматы обмена

Стандарт определяет пять основных форматов, названных по их числовой базе и количеству битов, используемых при их обменном кодировании. Существует три основных двоичных формата с плавающей запятой (с 32-, 64- или 128-битной

кодировкой) и два основных десятичных формата с плавающей запятой (с 64- или 128-битной кодировкой). Форматы binary32 и binary64 представляют собой одинарный и двойной форматы IEEE 754-1985 соответственно. Соответствующая реализация должна полностью реализовывать хотя бы один из основных форматов.

Стандарт также определяет форматы обмена, которые обобщают эти базовые форматы. Для двоичных форматов требуется соглашение о начальных битах. В следующей таблице приведены самые маленькие форматы обмена (включая основные).

Имя	Общее имя	Базовое	Знаки и биты или цифры	Десятичные цифры	Биты экспоненты
двоичное16	половинная точность	2	11	3,31	5
двоичный32	одинарная точность	2	24	7,22	8
binary64	Двойная точность	2	53	15.95	11
binary128	Четверная точность	2	113	34,02	15
двоичный 256	восьмеричная точность	2	237	71,34	19
десятичный32		10	7	7	7,58
десятичный64		10	16	16	9,58
десятичное 128		10	34	34	13,58

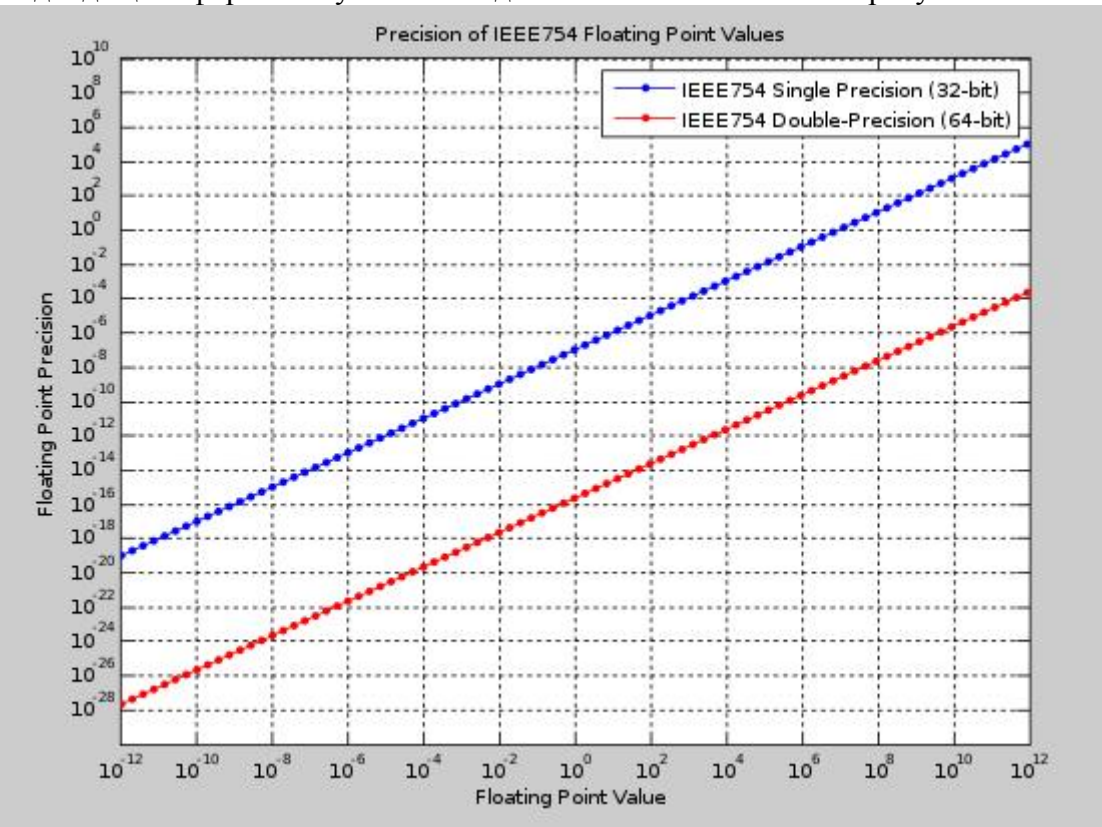
Имя	Десятичное E макс.	Экспоненциальное смещение	E min	E max	Примечания
двоичное16	4,51	2−1 = 15	−14	+15	не базовый
двоичный32	38.23	2−1 = 127	−126	+127	
binary64	307.95	2−1 = 1023	−1022	+1023	
binary128	4931,77	2−1 = 16383	−16382	+16383	
двоичный 256	78913,2	2−1 = 262143	−262142	+262143	не базовый
десятичный32	96	101	−95	+96	не базовый
десятичный64	384	398	- 383	+384	
десятичное 128	6144	6176	−6143	+6144	

Обратите внимание, что в приведенной выше таблице минимальные показатели степени указаны для нормальных чисел; специальное представление субнормального числа позволяет представлять даже меньшие числа (с некоторой потерей точности). Например, наименьшее положительное число, которое может быть представлено в двоичном формате, равно 2; Вклады в фигуру -1074 включают значение E min -1022 и все, кроме одного, из 53 значащих битов (2 = 2).

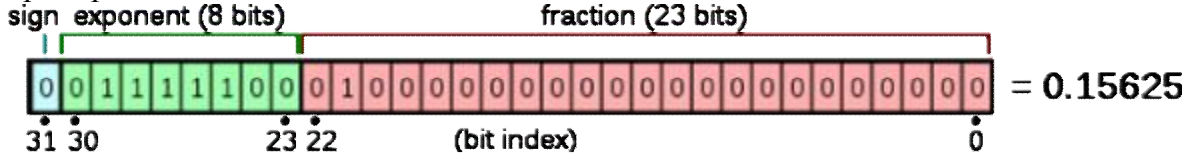
Десятичные цифры - это цифры × $\log_{10}$ основание. Это дает приблизительную точность в количестве десятичных цифр.

Десятичное E max равно Emax × $\log_{10}$ основание. Это дает приблизительное значение максимального десятичного показателя степени.

Форматы binary32 (одиночный) и binary64 (двойной) - два наиболее распространенных формата, используемых сегодня. На рисунке ниже показана абсолютная точность для обоих форматов в диапазоне значений. Этот рисунок может использоваться для выбора подходящего формата с учетом ожидаемого значения числа и требуемой точности.



Пример макета для 32-битной с плавающей запятой



Расширенные и расширяемые форматы точности

Стандарт определяет дополнительные расширенные и расширяемые форматы точности, которые обеспечивают большую точность, чем базовые форматы. Формат расширенной точности расширяет базовый формат за счет большей точности и большего диапазона экспонент. Расширяемый формат точности позволяет пользователю определять точность и диапазон экспоненты. Реализация может использовать любое внутреннее представление, выбранное для таких форматов; все, что необходимо определить, - это его параметры (b, p и emax). Эти параметры однозначно описывают набор конечных чисел (комбинации знака, значащей и экспоненты для данного основания системы счисления), которые он может представлять.

Стандарт рекомендует, чтобы языковые стандарты обеспечивали метод определения p и emax для каждой поддерживаемой базы b. Стандарт рекомендует, чтобы языковые стандарты и реализации поддерживали расширенный формат, который имеет большую точность, чем самый большой базовый формат, поддерживаемый для каждого основания b. Для расширенного формата с точностью между двумя основными форматами диапазон экспоненты должен быть таким же большим, как и для следующего более широкого базового формата. Так, например, 64-битное двоичное число с расширенной точностью должно иметь 'emax' не менее 16383. Этому требованию отвечает x8780-битный расширенный формат .

Форматы обмена
Форматы обмена предназначены для обмена данными с плавающей запятой с использованием битовой строки фиксированной длины для данного формата.

Двоичный
Для обмена двоичными числами с плавающей запятой определены форматы обмена длиной 16 бит, 32 бита, 64 бита и любое кратное 32 бит ≥ 128. 16-битный формат предназначен для обмена или хранения небольших чисел (например, для графики).

Схема кодирования для этих двоичных форматов обмена такая же, как и в IEEE 754-1985: знаковый бит, за которым следуют w битов экспоненты, которые описывают смещение экспоненты на смещение , и p - 1 бит, описывающий значение. Ширина поля экспоненты для k-битного формата вычисляется как $w = \text{round}\,(4 \log_2(k)) - 13$. Существующие 64- и 128-битные форматы следуют этому правилу, но 16- и 32-битные форматы имеют больше битов экспоненты (5 и 8 соответственно), чем дает эта формула (3 и 7 соответственно).

Как и в IEEE 754-1985, поле смещенной экспоненты заполняется всеми 1 битами, чтобы указать либо бесконечность (конечное поле значимости = 0), либо NaN (конечное значение значимости 0). Для NaN, тихие NaN и сигнальные NaN различаются исключительно использованием самого старшего бита конечного поля значимости, а полезная нагрузка переносится в оставшихся битах.

Десятичный
Для обмена десятичными числами с плавающей запятой определены форматы обмена, кратные 32 битам. Как и в случае двоичного обмена, схема кодирования для форматов десятичного обмена кодирует знак, показатель степени и значение. Определены две разные кодировки битового уровня, и обмен затруднен тем фактом, что может потребоваться некоторый внешний индикатор используемой кодировки.

Эти две опции позволяют кодировать мантиссу как сжатую последовательность десятичных цифр с использованием плотно упакованного десятичного числа или, альтернативно, как двоичное целое число . Первый более удобен для прямой аппаратной реализации стандарта, а второй больше подходит для программной эмуляции на двоичном компьютере. В любом случае набор чисел (комбинации знака, значащей и экспоненты), который может быть закодирован, идентичен, и специальные значения (± ноль с минимальным показателем, ± бесконечность, тихие NaN и сигнализация NaN) имеют одинаковую кодировку.

https://ieeexplore.ieee.org/document/4610935 754-2008 - IEEE Standard for Floating-Point Arithmetic
60559-2020 - ISO/IEC/IEEE International Standard - Floating-point arithmetic

Правила округления

Стандарт определяет пять правил округления. Первые два правила округляются до ближайшего значения; остальные называются направленным округлением :

Округлением до ближайшего
Округлением до ближайшего, привязкой к четному - округлением до ближайшего значения; если число выпадает на полпути, оно округляется до ближайшего значения с четной младшей цифрой; это значение по умолчанию для двоичных чисел с плавающей запятой и рекомендуемое значение по умолчанию для десятичных.
Округление до ближайшего, отнесение к нулю - округление до ближайшего значения; если число падает на полпути, оно округляется до ближайшего значения выше (для положительных чисел) или ниже (для отрицательных чисел); это предназначено как опция для десятичных чисел с плавающей запятой.
Направленное округление
Округление в сторону 0 - направленное округление до нуля (также известное как усечение).
Округление в сторону + ∞ - направленное округление в сторону положительной бесконечности (также известное как округление вверх или потолок).
Округление в сторону −∞ - направленное округление в сторону отрицательной бесконечности (также известное как округление вниз или пол).
Пример округления до целых чисел с использованием IEEE 754 правила

Режим	Пример значения			
	+11,5	+12,5	−11,5	−12,5
до ближайшего, связи до четного	+12.0	+12.0	−12.0	−12.0
до ближайшего, увязывает от нуля	+12.0	+13.0	−12.0	−13.0
в сторону 0	+11.0	+12.0	−11.0	−12.0
в сторону + ∞	+12.0	+13.0	−11.0	- 12.0
в сторону −∞	+11.0	+12.0	−12.0	−13.0

Если не указано иное, результат операции с плавающей запятой определяется путем применения функции округления к бесконечно точному (математическому) результату. Такая операция называется правильно округленной. Это требование называется правильным округлением.

Необходимые операции

Необходимые операции для поддерживаемого арифметического формата (включая базовые форматы) включают:

Арифметические операции (сложение, вычитание, умножение, деление, квадратный корень, объединенное умножение – сложение , остаток)
Преобразования (между форматами, в и из строк и т. д.)
Масштабирование и (для десятичных) квантование
Копирование и изменение знака (абс., Отрицание и т. д.)
Сравнение и полное упорядочение
Классификация и тестирование для NaN и т. Д.
Тестирование и настройка flags
Разные операции.

Предикаты сравнения

Стандарт предоставляет предикаты сравнения для сравнения одного элемента данных с плавающей запятой с другим в поддерживаемом арифметическом формате. Любое сравнение с NaN считается неупорядоченным. −0 и +0 сравниваются как равные.

Предикат общего порядка

Стандарт предоставляет предикат totalOrder, который определяет общий порядок канонических элементов поддерживаемого арифметического формата. Предикат согласуется с предикатами сравнения, когда одно число с плавающей запятой меньше другого. Предикат totalOrder не устанавливает тотальный порядок для всех кодировок в формате. В частности, он не делает различий между разными кодировками одного и того же представления с плавающей запятой, как когда одна или обе кодировки являются неканоническими. IEEE 754-2019 включает пояснения к totalOrder.

Обработка исключений

Стандарт определяет пять исключений, каждое из которых возвращает значение по умолчанию и имеет соответствующий флаг состояния, который поднимается при возникновении исключения. Никакой другой обработки исключений не требуется, но рекомендуются дополнительные альтернативы не по умолчанию (см. § Альтернативная обработка исключений).

Пять возможных исключений:

Недопустимая операция: математически не определено, например, квадратный корень из отрицательного числа. По умолчанию возвращает qNaN.
Деление на ноль: операция с конечными операндами дает точный бесконечный результат, например, 1/0 или log (0). По умолчанию возвращает ± бесконечность.
Переполнение: результат слишком велик для правильного представления (т.е. его показатель степени с неограниченным диапазоном экспоненты будет больше, чем emax). По умолчанию возвращает ± бесконечность для режимов округления до ближайшего (и следует правилам округления для режимов направленного округления).
Незаполнение: результат очень мал (вне нормального диапазона) и неточен. По умолчанию возвращает субнормальное или ноль (в соответствии с правилами округления).
Неточный: точный (т. Е. Неокругленный) результат не представляется точно. По умолчанию возвращает правильно округленный результат.
Это те же пять исключений, которые были определены в IEEE 754-1985, но исключение деления на ноль распространено на операции, отличные от деления.

Для десятичных чисел с плавающей запятой существуют дополнительные исключения:

Ограничено: показатель степени результата слишком велик для формата назначения. По умолчанию к коэффициенту добавляются завершающие нули, чтобы уменьшить показатель степени до наибольшего полезного значения. Если это невозможно (потому что это приведет к тому, что количество цифр должно быть больше, чем формат назначения), тогда возникает исключение переполнения.
Округленное: коэффициент результата требует большего количества цифр, чем предоставляет формат назначения. О неточном исключении сигнализируется, если отбрасываются любые ненулевые цифры.
Кроме того, такие операции, как квантование, когда один из операндов бесконечен или когда результат не соответствует формату назначения, также будут сигнализировать об исключении недопустимой операции.

Рекомендации

Альтернативная обработка исключений

Стандарт рекомендует дополнительную обработку исключений в различных формах, включая предварительную подстановку пользовательских значений по умолчанию и ловушек (исключения, которые каким-то образом изменяют поток управления) и другие модели обработки исключений, которые прерывают поток, например try / catch. Ловушки и другие механизмы исключения остаются необязательными, как в IEEE 754-1985.

Рекомендуемые операции

В разделе 9 стандарта рекомендуются дополнительные математические операции, которые должны быть определены в языковых стандартах. Ничего не требуется для соответствия стандарту.
Рекомендуемые арифметические операции, которые должны правильно округляться:

- $e^x, 2^x, 10^x$
- $e^x - 1, 2^x - 1, 10^x - 1$
- $\ln x, \log_2 x, \log_{10} x$
- $\ln(1-x), \log_2(1-x), \log_{10}(1+x)$
- $\sqrt{x^2+y^2}$
- $\sqrt{x}$
- $(1-x)^n$
- $x^{\frac{1}{n}}$
- x^n, x^y
- $\sin x, \cos x, \tan x$
- $\arcsin x, \arccos x, \arctan x, \operatorname{atan2}(y, x)$
- $\operatorname{sinPi} x = \sin \pi x, \operatorname{cosPi} x = \cos \pi x, \operatorname{tanPi} x = \tan \pi x$ (see also: Multiples of π)
- $\operatorname{asinPi} x = \frac{\arcsin x}{\pi}, \operatorname{acosPi} x = \frac{\arccos x}{\pi}, \operatorname{atanPi} x = \frac{\arctan x}{\pi}, \operatorname{atan2Pi}(y, x) = \frac{\operatorname{atan2}(y, x)}{\pi}$
- $\sinh x, \cosh x, \tanh x$
- $\operatorname{arsinh} x, \operatorname{arcosh} x, \operatorname{artanh} x$

Функции asinPi, acosPi и tanPi не были частью стандарту IEEE 754-2008, потому что казалось, что они менее необходимы. Первые два были хотя бы упомянуты в абзаце, но это считалось ошибкой, пока они не были добавлены в редакцию 2019 года.

Эти операции также включают в себя установку направления округления в динамическом режиме и доступ к нему, а также операции уменьшения вектора, определенные реализацией, такие как сумма, масштабированное произведение и скалярное произведение , точность которых не указана стандартом.

С 2019 года также рекомендуются расширенные арифметические операции для двоичных форматов. Эти операции, указанные для сложения, вычитания и умножения, производят пару значений, состоящую из результата, правильно округленного до ближайшего в формате, и члена ошибки, который можно точно представить в этом формате. На момент публикации стандарта аппаратные реализации не были известны, но очень похожие операции уже были реализованы в программном обеспечении с использованием хорошо известных алгоритмов. История и мотивация их стандартизации объясняются в справочном документе.

По состоянию на 2019 год ранее необходимые minNum, maxNum, minNumMag и maxNumMag в IEEE 754-2008 теперь удалены из-за их неассоциативности. Вместо этого рекомендуется два набора новых минимальных и максимальных операций. Первый набор содержит минимум, минимум, максимум и максимум. Второй набор содержит minimumMagnitude, minimumMagnitudeNumber, maximumMagnitude и maximumMagnitudeNumber. История и мотивация этого изменения объясняются в справочном документе.

Оценка выражений

Стандарт рекомендует, как языковые стандарты должны определять семантику последовательностей операций, и указывает на тонкости буквального смыслы и оптимизации, которые меняют ценность результата. Напротив, предыдущая версия стандарта 1985 оставляла аспекты языкового интерфейса неопределенными, что приводило к несогласованному поведению между компиляторами или различным уровням оптимизации в одном компиляторе.

Языки программирования должны позволять пользователю указывать минимальную точность для промежуточных вычислений выражений для каждого основания. В стандарте это называется «предпочтительной шириной», и должна быть возможность устанавливать ее для каждого блока. Промежуточные вычисления в выражениях должны быть рассчитаны, а любые временные значения должны быть сохранены с использованием максимальной ширины операндов и предпочтительной ширины, если она установлена. Таким образом, например, компилятор, ориентированный на оборудование с плавающей запятой x87 , должен иметь средства указания, что промежуточные вычисления должны использовать двойной расширенный формат . Сохраненное значение переменной должно всегда использоваться при вычислении последующих выражений, а не любого предшественника до округления и присвоения переменной.

Воспроизводимость

IEEE 754-1985 допускает множество вариантов реализации (например, кодирование некоторых значений и обнаружение определенных исключений). IEEE 754-2008 усилил многие из них, но некоторые варианты все еще остаются (особенно для двоичных форматов). Пункт о воспроизводимости рекомендует, чтобы языковые стандарты предоставляли средства для написания воспроизводимых программ (т. Е. Программ, которые будут давать одинаковый результат во всех реализациях языка), и описывает, что необходимо сделать для достижения воспроизводимых результатов.

Символьное представление

Стандарт требует операций для преобразования между базовыми форматами и внешними форматами последовательности символов. Преобразование в формат десятичных символов и обратно требуется для всех форматов. Преобразование во внешнюю последовательность символов должно быть таким, чтобы обратное преобразование с округлением до четного восстановило исходное число. Нет требования сохранять полезную нагрузку в виде тихого NaN или сигнального NaN, а преобразование из внешней последовательности символов может превратить сигнальный NaN в тихий NaN.

Исходное двоичное значение будет сохранено путем преобразования в десятичное и обратно с использованием:

5 десятичных цифр для binary16,
9 десятичных цифр для binary32,
17 десятичные цифры для двоичного64,
36 десятичных цифр для двоичного 128.
Для других двоичных форматов необходимое количество десятичных цифр составляет

$1 + [p \log_{10}(2)]$
где p - количество значащих битов в двоичном формате, например 237 бит для двоичного 256.

(Примечание: в качестве ограничения реализации правильное округление гарантируется только для количества десятичных цифр выше плюс 3 для наибольшего поддерживаемого двоичного формата. Например, если binary32 является наибольшим поддерживаемым двоичным форматом, то преобразование из десятичной внешней последовательности с 12 десятичными цифрами гарантированно правильно округляется при преобразовании в binary32; но преобразование последовательности из 13 десятичных цифр - нет; однако стандарт рекомендует, чтобы реализации не налагали такого ограничения.)

При использовании десятичного формата с плавающей запятой десятичное представление будет сохранено с использованием:

7 десятичных цифр для decimal32,
16 десятичных цифр для decimal64,
34 десятичных цифр для decimal128 .

Алгоритмы с кодом для правильно округленного преобразования из двоичного в десятичное и из десятичного в двоичное обсуждают Гей, а для тестирования - Паксон и Кахан.Википедия site:wiki5.ru

Воспроизводимость

IEEE 754-1985 допускает множество вариантов реализации (например, кодирование некоторых значений и обнаружение определенных исключений). IEEE 754-2008 усилил многие из них, но некоторые варианты все еще остаются (особенно для двоичных форматов). Пункт о воспроизводимости рекомендует, чтобы языковые стандарты предоставляли средства для написания воспроизводимых программ (т. Е. Программ, которые будут давать одинаковый результат во всех реализациях языка), и описывает, что необходимо сделать для достижения воспроизводимых результатов.

NaN

Два вида NaN: тихий NaN (qNaN) и сигнализационный NaN (sNaN). NaN может нести полезную нагрузку, предназначенную для диагностической информации, указывающей источник, вызвавший NaN. Знак NaN не имеет никакого значения, но может быть предсказуемым в некоторых случаях.
NaN = s 111 1111 1xxx xxxx xxxx xxxx xxxx xxxx
где s - это знак (чаще всего игнорируемый в приложениях), а последовательность x представляет ненулевое число (нулевое значение кодирует бесконечности). Первый бит из x используется для определения типа NaN: «тихий NaN» или «сигнальный NaN». Остальные биты кодируют полезную нагрузку (чаще всего игнорируются в приложениях).
Операции с плавающей запятой, отличные от упорядоченных сравнений, обычно передают тихий NaN (qNaN). Большинство операций с плавающей запятой в сигнальном NaN (sNaN) сигнализируют об исключительной ситуации недопустимой операции; тогда действие исключения по умолчанию такое же, как и для операндов qNaN, и они производят qNaN, если производят результат с плавающей запятой.

Распространение тихих NaN посредством арифметических операций позволяет обнаруживать ошибки в конце последовательности операций без тщательного тестирования на промежуточных этапах. Например, если кто-то начинает с NaN и добавляет 1 пять раз подряд, каждое добавление приводит к NaN, но нет необходимости проверять каждое вычисление, потому что можно просто отметить, что окончательный результат - NaN. Однако, в зависимости от языка и функции, NaN можно незаметно удалить из цепочки вычислений, в которой одно вычисление в цепочке даст постоянный результат для всех других значений с плавающей запятой. Например, вычисление x может привести к результату 1, даже если x равно NaN, поэтому проверка только окончательного результата может скрыть тот факт, что вычисление перед x привело к NaN. В общем, тогда необходим более поздний тест на наличие установленного недопустимого флага, чтобы обнаружить все случаи, когда вводятся NaN.

В разделе 6.2 старого стандарта IEEE 754-2008 есть две аномальные функции (функции maxNum и minNum, которые возвращают максимум два операнда, которые, как ожидается, будут числами), которые предпочитают числа - если только один из операндов является NaN, то возвращается значение другого операнда. В версии IEEE

754-2019 эти функции были заменены, поскольку они не являются ассоциативными (когда в операнде появляется сигнальное NaN).

Сравнение с NaN

Сравнение с NaN всегда возвращает неупорядоченный результат даже при сравнении с самим собой. Предикаты сравнения либо сигнализируют, либо не сигнализируют о тихих операндах NaN; версии сигнализации сигнализируют об исключении недопустимой операции для таких сравнений. Предикаты равенства и неравенства не сигнализируют, поэтому x = x, возвращающий false, можно использовать для проверки, является ли x тихим NaN. Все другие стандартные предикаты сравнения сигнализируют о получении операнда NaN. Стандарт также предоставляет несигнальные версии этих других предикатов. Предикат isNaN (x)определяет, является ли значение NaN, и никогда не сигнализирует об исключении, даже если x является сигнальным NaN.

Сравнение NaN и любого значения x с плавающей запятой (включая NaN и ± ∞)						
Сравнение	NaN ≥ x	NaN ≤ x	NaN>x	NaN < x	NaN = x	NaN ≠ x
Результат	Всегда Ложь	Всегда Ложь	Всегда Ложь	Всегда Ложь	Всегда Ложь	Всегда Истина

Операции, генерирующие NaN

Есть три типа операций, которые может возвращать NaN:

- Большинство операций хотя бы с одним операндом NaN.
- Неопределенные формы :
  - Деления (± 0) / (± 0) и (± ∞) / (± ∞).
  - Умножения (± 0) × (± ∞) и (± ∞) × (± 0).
  - Остаток x% y, когда x равен бесконечности или y равен нулю.
  - Сложение (+ ∞) + (−∞), (−∞) + (+ ∞) и эквивалентные вычитания (+ ∞) - (+ ∞) и (−∞) - (−∞).
  - В стандарте есть альтернативные функции для степеней:
    - Стандартная функция pow и функция целочисленной экспоненты pown определяют 0, 1 и ∞ как 1.
    - Функция powrопределяет все три неопределенных для мс как недопустимые операции и поэтому возвращает NaN.
- Действительные операции с сложными результатами, например:
  - Квадратный корень из отрицательного числа.
  - логарифм отрицательного числа.
  - обратный синус или обратный косинус числа, которое меньше -1 или больше 1.

NaN также могут быть явно присвоены переменным, обычно как представление пропущенных значений. До стандарта IEEE программисты часто использовали специальные значения (например, -99999999) для представления неопределенных или отсутствующих значений, но не было гарантии, что они будут обрабатываться согласованно или правильно.

NaN не обязательно генерируются во всех перечисленных случаях. Если операция может вызвать исключительную ситуацию и ловушки не замаскированы, тогда операция вызовет ловушку. Если операнд является тихим NaN, а также нет сигнального операнда NaN, тогда нет условия исключения и результатом является тихий NaN. Явные присвоения не вызовут исключения даже для сигнализации NaN.

Тихие NaN

Тихие NaN, или qNaN, не вызывают никаких дополнительных исключений, поскольку они распространяются через большинство операций. Исключение составляют случаи, когда NaN нельзя просто передать в неизмененном виде на вывод, например, при преобразовании формата или некоторых операциях сравнения.

NaN сигнализации

NaN сигнализации или sNaN - это особые формы NaN, которые при использовании большинством операций должны вызывать исключение недопустимой операции, а затем, при необходимости, «подавляться» в qNaN, который затем может распространяться. Они были введены в IEEE 754. Было несколько идей, как их можно использовать:

- Заполнение неинициализированной памяти сигнальными NaN приведет к исключению недопустимой операции, если данные используются до инициализации
- Использование sNaN в качестве заполнителя для более сложный объект , например:
    - Представление числа, у которого отсутствует
    - Представление числа, которое имеет переполнение
    - Число в формат с более высокой точностью
    - А комплексное число

При обнаружении обработчик прерывания может декодировать sNaN и вернуть индекс для вычисленного результата. На практике такой подход сталкивается со многими сложностями. Обработка знакового бита NaN для некоторых простых операций (таких как абсолютное значение) отличается от обработки для арифметических операций. Стандарт не требует ловушек. Есть и другие подходы к такого рода проблемам, которые были бы более переносимыми.

Определение функции

Существуют разногласия по поводу правильного определения результата числовой функции, которая принимает тихий NaN в качестве входных данных. Одна точка зрения состоит в том, что NaN должно распространяться на выход функции во всех случаях, чтобы распространять индикацию ошибки. Другой взгляд, принятый стандартами ISO C99 и IEEE 754-2008 в целом, заключается в том, что если функция имеет несколько аргументов и вывод однозначно определяется всеми не -NaN (включая бесконечность), тогда это значение должно быть результатом. Таким образом, например, значение, возвращаемое hypot (± ∞, qNaN)и hypot (qNaN, ± ∞), равно + ∞.

Проблема особенно остро стоит для функции возведения в степень pow (x, y)= x. Выражения 0, ∞ и 1 считаются неопределенными формами, когда они встречаются как пределы (точно так же, как ∞ × 0), и вопрос о том, является ли от нуля до нулевой степени должен быть определен как 1 разделились мнения.

Если вывод считается неопределенным, когда параметр не определен, то pow (1, qNaN)должно давать qNaN. Однако математические библиотеки обычно возвращают 1 для pow (1, y)для любого действительного числа y, и даже когда y равно бесконечности. Точно так же они производят 1 для pow (x, 0), даже если x равен 0 или бесконечности. Обоснованием для возврата значения 1 для неопределенных форм было то, что значение функций в особых точках может быть принято как конкретное значение, если это значение находится в предельном значении для всех, кроме исчезающе малой части шара вокруг предельного значения. параметров. В версии стандарта IEEE 754 2008 года говорится, что оба pow (1, qNaN)и pow (qNaN, 0)должны возвращать 1, поскольку они возвращают 1 независимо от else используется вместо тихого NaN. Более того, ISO C99, а затем IEEE 754-2008, решили указать pow (−1, ± ∞)= 1 вместо qNaN; причина этого выбора указана в обосновании языка C: «Как правило, C99 избегает результата NaN, когда полезно числовое значение.... Результат pow (−2, ∞)равен + ∞, потому что все большие положительные значения с плавающей запятой являются целыми числами ".

Чтобы удовлетворить тех, кто желает более строгой интерпретации того, как должна действовать степенная функция, стандарт 2008 определяет две дополнительные степенные функции: pown (x, n), где показатель степени должен быть целое число и powr (x, y), которое возвращает NaN всякий раз, когда параметр является NaN, или возведение в степень даст неопределенную форму .

Integer NaN

Наиболее фиксированный -size целочисленные форматы не могут явно указывать недопустимые данные. В таком случае при преобразовании NaN в целочисленный тип стандарт IEEE 754 требует, чтобы сообщалось о недопустимой операции исключение. Например, в Java такие операции вызывают экземпляры java.lang.ArithmeticException. В C они приводят к неопределенному поведению, но если приложение F поддерживается, операция приводит к «недопустимому» исключению с плавающей запятой (в соответствии с требованиями стандарта IEEE) и неопределенному ценность.

Пакет Perl Math :: BigIntиспользует "NaN" для результата строк, которые не представляют действительные целые числа.

>perl -mMath :: BigInt -e "print Math :: BigInt->new ('foo') "NaN

Display

Различные операционные системы и языки программирования могут иметь разные строковые представления NaN.

nan NaN NaN% NAN NaNQ NaNS qNaN sNaN 1. # SNAN 1. # QNAN -1. # IND + nan.0

Поскольку на практике закодированные NaN имеют знак, тихий / сигнальный бит и необязательная «диагностическая информация» (иногда называемая полезной нагрузкой), они также часто встречаются в строковых представлениях NaN, например:

-NaN NaN12345 -sNaN12300 -NaN (s1234)

(существуют другие варианты).

Кодирование

В форматах хранения с плавающей запятой, соответствующихстандарту IEEE 754, NaN идентифицируются с помощью определенных заранее заданных битовых шаблонов, уникальных для NaN. Знаковый бит не имеет значения. Двоичный формат NaN представлены экспоненциальным полем, заполненным единицами (например, значениями бесконечности), и некоторым ненулевым числом в значении поля (чтобы отличить их от значений бесконечности). Исходный стандарт IEEE 754 1985 года (IEEE 754-1985) описывал только двоичные форматы с плавающей запятой и не определял, как должно быть помечено состояние сигнализации / молчания. На практике старший бит значимого поля определяет, является ли NaN сигнальным или нет. В результате были реализованы две разные реализации с обратным смыслом:

- большинство процессоров (включая те из Intel и AMD семейства x86, Семейство Motorola 68000, семейство AIM PowerPC, семейство ARM, семейство Sun SPARC и, необязательно, новые процессоры MIPS) устанавливают бит сигнализации / молчания в ненулевое значение, если NaN не используется, и в ноль, если NaN сигнализирует. Таким образом, на этих процессорах бит представляет собой флаг is_quiet;
- в NaN, генерируемых PA-RISC и старыми процессорами MIPS, бит сигнализации / молчания равен ноль, если NaN тихо, и ненулевое значение, если NaN сигнализирует. Таким образом, на этих процессорах этот бит представляет собой флаг is_signaling.

Первый вариант предпочтительнее, поскольку он позволяет реализации заглушить сигнальный NaN, просто установив бит сигнализации / молчания в 1. обратное невозможно с последним выбором, потому что установка бита сигнализации / молчания в 0 может дать бесконечность.

Версия стандарта IEEE 754 от 2008 г. (IEEE 754-2008) дает формальные рекомендации по кодированию состояния сигнализации / молчания.

- Для двоичных форматов наиболее значимым битом значимого поля должен быть флаг is_quiet. То есть, этот бит не равен нулю, если NaN неактивен, и равен нулю, если NaN сигнализирует.
- Для десятичных форматов, как в двоичном, так и в десятичном кодировании, NaN идентифицируется по пяти старшим битам поле комбинации после бита знака установлено в единицы. Шестой бит поля - это флаг is_quiet. Стандарт следует интерпретации как флаг is_signaling. То есть бит сигнализации / молчания равен нулю, если NaN является тихим, и ненулевым, если NaN сигнализирует. Сигнализация NaN подавляется очисткой этого шестого бита.

Для соответствия IEEE 754-2008 значение бита сигнализации / молчания в последних процессорах MIPS теперь можно настраивать через поле NAN2008 регистра FCSR. Эта поддержка является необязательной в MIPS версии 3 и требуется в версии 5.

Состояние / значение остальных битов значимого поля не определены стандартом. Это значение называется «полезной нагрузкой» NaN. Если операция имеет единственный вход NaN и передает его на выход, полезная нагрузка результата NaN должна быть полезной нагрузкой входного NaN (это не всегда возможно для двоичных форматов, когда состояние сигнализации / молчания кодируется с помощью is_signalingфлаг, как описано выше). Если имеется несколько входов NaN, полезная нагрузка результата NaN должна быть из одного из входных NaN; в стандарте не указано, какие именно.