СОДЕРЖАНИЕ

ПРИЛОЖЕНИЯ

ПРЕДИСЛОВИЕ

Сегодня знание цифровой техники требуется в очень многих областях. Невозможно понять принципы работы мобильной связи, микрокомпьютерных технологий, систем цифрового управления и различного телекоммуникационного оборудования без знания методов цифровой обработки информации, а широкое применение интегрированных механико-электронных систем лишь усиливает эту тенденцию. Особое значение цифровые технологии приобрели во встраиваемых системах. Речь идет о цифровых схемах, которые «встраиваются» в те или иные технические решения. Встраиваемые системы вы можете встретить в самых разных устройствах — стиральных машинах, автомобилях, холодильниках, развлекательной электронике, мобильных телефонах и т. д.

В данной книге отлично изложены основы цифровой техники, включая устройство и программирование простых микропроцессоров. Помимо прочной теоретической базы, читатель получит знания, позволяющие понять принципы работы большинства цифровых схем. В это 8-е издание внесены многочисленные дополнения. В частности, в главе о микропроцессорах в качестве примера рассматривается микропроцессор, широко используемый в современной промышленности. Также добавлен материал о представлении чисел с плавающей запятой.

Эта книга предназначена, в первую очередь, для студентов колледжей и университетов, изучающих инженерию и информационные технологии. Поскольку для понимания книги не требуется никаких специальных знаний, она также подходит для широкого круга заинтересованных читателей. Для работы с главой «Схемотехника» читателю понадобятся базовые знания в области электроники, однако эта глава не обязательна для понимания других глав книги, и ее можно пропустить. Представление булевой алгебры и используемые символы в основном соответствуют действующим стандартам DIN[1].

[1] *DIN* (Deutsches Institut für Normung) – Немецкий институт по стандартизации — *Примеч. науч. ред.*

Для облегчения самостоятельной работы с книгой каждая глава сопровождается упражнениями, с помощью которых можно проверить понимание материала. Предлагаемые решения можно найти в приложении. Одна из основных задач данной книги — это подробное изложение основ цифровой техники. В частности, много внимания уделено синтезу коммутационных схем. Рассмотрены примеры широко применяемых стандартных коммутационных схем, таких как мультиплексоры и преобразователи кода. Поскольку для понимания работы процессоров важно знание арифметических операций, подробно рассматриваются арифметика с фиксированной запятой и аппаратная реализация арифметико-логических устройств. Сегодня в распоряжении разработчиков схем находятся совершенные инструменты проектирования, позволяющие разрабатывать сложные цифровые схемы, реализовывать их на основе полупроводников, тестировать и настраивать работу схемы. Это обусловило бурное развитие интегральных схем специального назначения (ASIC), которые заказчик может настроить самостоятельно. Поэтому одна из глав посвящена таким интегральным схемам. Следующая глава знакомит с VHDL – языком программирования для описания, синтеза и моделирования интегральных цифровых схем, который фактически стал стандартом и часто используется при проектировании схем ASIC. Такой язык описания оборудования (HDL) становится все более популярным при проектировании схем, поскольку он дает значительные преимущества по сравнению с традиционными методами графически ориентированного проектирования, особенно при работе над сложными проектами. Опираясь на знание коммутационных схем, можно переходить к изучению синхронных и асинхронных методов коммутации. В книге представлена методика разработки синхронных и асинхронных контроллеров. При этом в новой редакции больше внимания уделено синхронным контроллерам. Методы асинхронной коммутации сейчас применяются, главным образом, в триггерах, наиболее распространенные типы которых представлены в книге. Кроме того, рассмотрено проектирование счетчиков и регистров сдвига и представлены некоторые примеры их промышленной реализации. Отдельная глава посвящена сравнению различных технологий и свойств модулей памяти, которые существенно влияют на характеристики современных вычислительных систем. Приведены типовые временные диаграммы, иллюстрирующие принципы работы различных модулей памяти. Последняя глава представляет простое введение в микропроцессорную технику. Для знакомства с устройством вычислительной машины поясняется принцип работы компьютера фон Неймана, на примере которого описываются процессы выполнения команд. В качестве практического примера в одном из разделов представлен современ-

ный микроконтроллер ATmega16 производства компании AVR, описаны его структура и режимы работы. Подробно рассматривается программирование на ассемблере, знание которого будет полезно при работе с различными типами процессоров.

Фульда,
март 2018 года,
Клаус Фрике

ГЛАВА 1

ВВЕДЕНИЕ

В последние годы значение цифровой техники все более и более возрастает. Причина этого заключается в значительных преимуществах цифровой техники при создании очень сложных систем. Это достигается путем представления сигнала двумя значениями, которые могут обрабатываться логическими вентилями с сильно нелинейными передаточными характеристиками без сбоев, накопления и дальнейшего распространения искажений сигнала. Благодаря такому представлению сигналов удалось создать полупроводниковую технологию, позволяющую реализовать до 10^7 элементов на одном кристалле.

Поскольку целью систем цифровой техники является обработка сигналов, то следует несколько подробнее рассмотреть понятие «сигнал». Сигналы служат для переноса информации. Они описываются такими физическими величинами, как напряжение, ток, давление, сила и т. д. Амплитуды таких величин зависят от времени. Передаваемая информация заключается в изменяющихся амплитудных значениях. Пусть, например, измеряется зависящий от времени уровень жидкости F в резервуаре. На рис. 1.1a показана зависимость уровня жидкости от времени. Если датчик выдает электрический сигнал, напряжение которого пропорционально величине заполнения, получаем временную зависимость напряжения U_s, показанную на рис. 1.1 b. Этот сигнал имеет непрерывную величину, то есть в измеряемом диапазоне могут появиться все значения амплитуды.

Системы, которые могут обрабатывать непрерывные по величине сигналы, называются аналоговыми системами.

Дискретные по величине сигналы, которые также называют цифровыми сигналами, могут, в отличие от этого, принимать только определенные дискретные значения в фиксированные моменты времени. Примером цифрового сигнала является сигнал, который с помощью двух различных уровней напряжения показывает, закрыта или открыта дверь. Подобные сигналы могут непосредственно обрабатываться цифровыми системами.

В том случае, если сигналы с непрерывной величиной должны передаваться цифровыми системами, эти сигналы должны быть предварительно квантизированы. В соответствии с данным процессом, с определенными временными точками (точками считывания) соотнесена амплитуда сиг-

нала дискретной амплитудной ступени. Вышеупомянутый пример продемонстрирован на рис. 1.1c, показывающем датчик уровня заполнения. При этом получаем напряжение с дискретными величинами U_q. При квантизировании необходимо принимать во внимание ошибку округления.

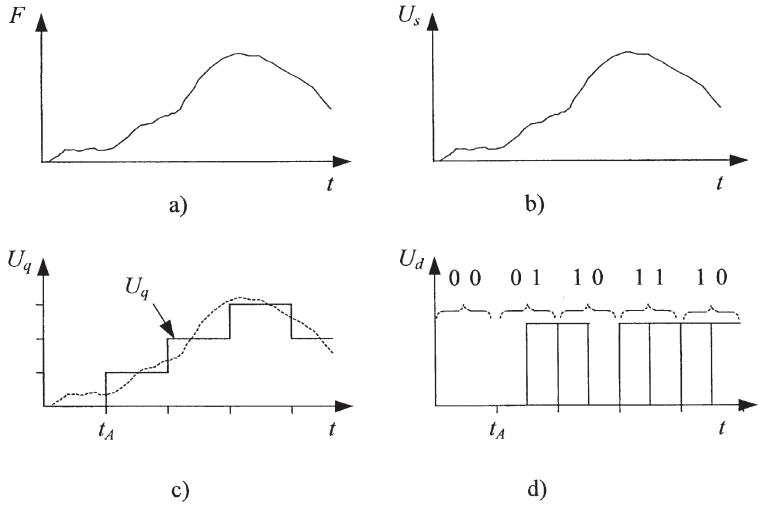


Рис. 1.1. Пример «оцифровывания» сигналов.
a) Зависимость от времени *t* уровня наполнения резервуара *F*.
b) Зависимость от времени выходного напряжения датчика U_s.
c) Квантизированная зависимость от времени напряжения U_q при 4 амплитудных ступенях.
d) Приведение амплитуд в соответствие с записью величин в виде кода 00, 01, 10 и 11

При цифровой передаче сигнал вначале должен быть переведен в цифровую форму. В этом случае амплитуда будет предъявляться с помощью последовательности цифр. Каждая цифра представляет собой сигнал с дискретным значением. На рис. 1.1d приведен пример кодирования с помощью нескольких следующих друг за другом цифр. Амплитудная ступень 0 представлена двумя цифрами 00. Амплитуды 1, 2 и 3 становятся сочетаниями 01, 10, 11. Следует различать также дискретные во времени сигналы и непрерывные во времени сигналы. Дискретные во времени сигналы могут изменять свою амплитуду только в определенные моменты времени, в то время как непрерывные во времени сигналы могут изменять свою амплитуду в произвольные моменты времени. Цифровые системы могут быть дискретными во времени, в этом случае их называют синхронными. Синхронизация осуществляется с помощью тактового сигнала.

Благодаря ограничению в использовании конечного числа амплитудных ступеней, цифровая система обладает высокой помехозащищенностью. Подвергнувшиеся воздействию помех цифровые сигналы могут быть однозначно приведены к первоначальным дискретным амплитудным значениям. Но чтобы не возникла ошибка, помеха не должна превышать половину расстояния между двумя амплитудными ступенями.

Цифровые системы имеют ряд преимуществ перед аналоговыми системами:

– При использовании цифровых сигналов не происходит накопления их искажений — появляется возможность реализации систем любой степени сложности, например микропроцессоров. Это свойство цифровых систем определяет их превосходство и при передаче на большие расстояния.

– Цифровые системы сравнительно легко проектировать, поскольку способ их описания, представляющий собой булеву алгебру, — аппарат очень удобный для автоматизации. Сегодня разработка сложных цифровых систем автоматизирована посредством применения высокопроизводительных алгоритмов.

– Цифровые системы можно относительно просто тестировать.

Недостаток цифровых систем:

– Цифровые системы являются более медленными системами, чем аналоговые. Поэтому в области высоких частот доминирует аналоговая техника.

ГЛАВА 2

КОДИРОВАНИЕ И СИСТЕМЫ СЧИСЛЕНИЯ

2.1. Коды

В цифровой технике коды используются для оптимального представления сигнала в случае его применения. Код отображает символы одного множества через символы второго множества. При этом должна существовать возможность декодирования, при которой вероятно получить исходные символы из кодированных.

Известным примером кода является код Морзе. Определение кода производится с помощью таблицы соответствия. Для кода Морзе подобное соответствие отображено в табл. 2.1. Этот код является обратимым, так как из буквы можно получить символ Морзе, а из него можно вновь образовать букву. Но это справедливо только для текста, написанного строчными буквами, поскольку код Морзе не делает различия между прописными и строчными буквами. Строго говоря, из кода Морзе нельзя восстановить путем декодирования текст, записанный с помощью строчных и прописных букв.

Таблица 2.1. Код Морзе

Алфавит	Код Морзе	Алфавит	Код Морзе	Алфавит	Код Морзе
a	· –	j	· – – –	s	· · ·
b	· – · –	k	– · –	t	–
c	– · · ·	l	· – · ·	u	· · –
d	– · ·	m	– –	v	· · · –
e	·	n	– ·	w	· – –
f	· · – ·	o	– – –	x	– · · –
g	– – ·	p	· – – ·	y	– · – –
h	· · ·	q	– – · –	z	– – · ·
I	· ·	r	· – ·		

Для каждого применения имеется более или менее подходящий код. Так, для проведения операций над числами в компьютере рационально применять иной код, чем для передачи чисел по линии связи. В данной главе исследуются различия между отдельными кодами и даны указания по их специфическому применению.

Комбинацию нескольких символов кода называют словом (word). В последующем мы ограничимся технически важным случаем, в котором все слова одного кода имеют одинаковую длину n. В коде Морзе это не происходит. Если в код входит множество символов N, то N^n различных слов могут иметь длину n. Когда используются все N^n возможных слов одного кода, то в этом случае говорят о минимальном коде. При использовании менее чем N^n слов его называют избыточным кодом. Ниже можно найти описание наиболее употребительных кодов, их полное описание можно получить из [7–9].

2.2. Двоичный код

Будучи универсальным, двоичный код является важнейшим в цифровых системах. В соответствии с определением, при котором применяются только символы 1 и 0, становится возможной обработка сигналов с помощью схемных элементов, работающих как переключатели. Также двоичный код позволяет использование арифметики, аналогичной арифметике десятичных систем. При этом двоичную систему счисления можно рассматривать как кодирование десятичной системы, где двоичное число состоит из слова, образованного символом $c_i \in \{0, 1\}$. Символы ci одного слова называют в цифровой технике битами. Слово z_2 в двоичном представлении формируется путем последовательного присоединения отдельных битов, как это показано ниже:

$$z_2 = c_{n-1} c_{n-2} \cdots c_1 c_0 \cdots c_{-m+1} c_{-m}. \tag{2.1}$$

Двоичное число имеет n разрядов перед запятой и m разрядов после запятой. Отдельным битам присвоены, в соответствии с их позицией i в слове, весовые коэффициенты 2^i. На основе этого можно рассчитать эквивалентное десятичное число z_{10}:

$$\begin{gathered} z_2 = g(z_{10}) = \\ = c_{n-1}\, 2^{n-1} + c_{n-2} 2^{n-2} + \cdots + c_1 2^1 + c_0 2^0 + \\ + \cdots + c_{-m+1} 2^{-m+1} + c_{-m} 2^{-m}. \end{gathered} \tag{2.2}$$

Рассмотрим в качестве примера двоичное число $10110{,}001_2$, которое в качестве такового отмечено символом 2. Оно интерпретируется как:

$$g(z_2) = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} = \\ = z_{10} = 22{,}375_{10}$$

Двоичный (или дуальный) код обозначается как взвешенный, поскольку стоящие дальше влево биты обладают более высокими весовыми коэффициентами. Уравнение (2.2) можно рассматривать как правило, в соответствии с которым производится преобразование двоичных чисел в десятичные числа.

Преобразование же десятичных чисел в двоичные числа является более сложным. Его можно описать различными алгоритмами для целочисленной и дробной частей. В приведенном выше примере с числом 22,37510 алгоритм должен быть представлен следующим образом:

- Вначале формируется целочисленная часть двоичного числа. Для чего целочисленная часть десятичного числа последовательно делится на 2 и записывается остаток, пока не будет получен 0.

$22 : 2 = 1$	остаток 0
$11 : 2 = 5$	остаток 1
$5 : 2 = 2$	остаток 1
$2 : 2 = 1$	остаток 0
$1 : 2 = 0$	остаток 1

↑ целочисл. часть двоичного числа

Соответствующее числу 22_{10} двоичное число представляет собой 10110_2.

- Второй шаг заключается в преобразовании дробной части десятичного числа в дробную часть двоичного числа. Вначале дробная часть десятичного числа умножается на 2. Целочисленная часть отделяется, она образует разряды двоичного числа с наименьшими значениями.

Процесс повторяется, как это показано ниже.

$0{,}375 \cdot 2 = 0{,}75$	+0
$0{,}75 \cdot 2 = 0{,}5$	+1
$0{,}5 \cdot 2 = 0$	+1

↓ дробная часть двоичного числа

В данном примере мы видим, что остаток равен 0. Но не обязательно так всегда бывает. В нормальном случае дробная часть эквивалентного двоичного числа имеет бесконечно большое количество разрядов. В данном случае необходимо удовольствоваться определенным числом разрядов после запятой и этим ограничить точность.

В нашем случае $0,375_{10}$ точно соответствует $0,011_2$. На основе целочисленной и дробной частей получаем искомое двоичное число $10110,011_2$.

2.3. Арифметические операции с фиксированной запятой в двоичной системе

В данной главе описываются арифметические операции с числами с фиксированной запятой. Арифметические операции с фиксированной запятой означают, что в них запятая всегда стоит на фиксированном месте. При этом место, на котором стоит запятая, ориентируется на позицию в запоминающем устройстве (ЗУ), на которой находится число. В этом случае нет необходимости реализовать запятую в аппаратуре компьютера. Она существует только в голове программиста. Мы ограничиваемся постоянной длиной слова n, как это имеет место в компьютерах. На основе этого можно обсудить проблему переполнения допустимой области.

2.3.1. Целочисленное сложение в двоичной системе

Целочисленное сложение двух чисел A и B производится в двоичной системе точно так же, как и в десятичной системе — по разрядам. Как и там, в каждом разряде должны быть просуммированы обе двоичных цифры a_n и b_n и перенос из предыдущего разряда C_{n-1}.

При сложении возникают (табл. 2.2) новая сумма S_n и новый перенос C_n.

В этой таблице дискретной линией разделены входные и выходные величины. Например:

$$\begin{array}{rr} & 0\,1\,1\,1\,1\,1\,1\,0 \\ + & 0\,0\,1\,1\,0\,1\,0\,1 \\ \hline \text{перенос} & 1\,1\,1\,1\,1\,0\,0 \\ \hline = & 1\,0\,1\,1\,0\,0\,1\,1 \end{array}$$

В приведённом выше примере, чтобы не было переполнения, необходимо суммировать два числа длиной 8 бит, итог которых также должен иметь длину 8 бит.

Таблица 2.2. Сложение в двоичной системе со слагаемыми a_n, b_n и переносом из предыдущего разряда C_{n-1}. Сумма равна S_n, новый перенос C_n

a_n	b_n	c_{n-1}	c_n	s_n
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

2.3.2. Сложение чисел с фиксированной запятой

В случае когда суммируются два числа с фиксированной запятой, важно, как и в обычной процедуре, чтобы обе запятые стояли друг над другом. Так, при сложении двух чисел длиной в 8 бит запятая у обоих чисел должна стоять, например, на третьем месте.

Например:

$$\begin{array}{r} +\ 0\,1\,1\,0\,0,\,0\,1\,0 \\ \quad 0\,0\,1\,1\,0,\,1\,1\,1 \\ \hline \text{перенос}\ 1\,1\,0\,0\,1\ 1\,0 \\ \hline = 1\,0\,0\,1\,1,\,0\,0\,1 \end{array}$$

2.3.3. Представление с помощью обратного кода

Для того, чтобы иметь малые затраты на аппаратурную часть (hardware) компьютера, были предприняты усилия по сведению к одному

алгоритму вычитания и сложения. Этого можно добиться, если применять двоичные цифры в их дополняющей форме. Различают единичное дополнение (обратный код, поразрядное дополнение) и двойное дополнение (точное дополнение).

Поразрядное дополнение формируется путем замены всех нулей на единицы и обратно. Следовательно, поразрядным дополнением (one's complement) 0001 является 1110. Ниже поразрядное дополне ние двоичного числа A обозначено $\neg A$. Очевидно, что при представлении n-битового слова имеем:

$$\neg A + A = 2^n - 1. \tag{2.3}$$

Например, при представлении 8-битового слова имеем:

$$10110011 + 01001100 = 11111111 = 2^8 - 1.$$

Можно так преобразовать уравнение (2.3), чтобы получить формулу для расчета поразрядного дополнения:

$$\neg A = 2^n - 1 - A. \tag{2.4}$$

2.3.4. Представление с помощью двойного дополнения (точное дополнение, two's complement)

Точное дополнение A_{K2} образуется из поразрядного дополнения $\neg A$ путем прибавления 1:

$$A_{K2} = \neg A + 1. \tag{2.5}$$

Следовательно, с учетом (2.4) будет справедливо:

$$A_{K2} = 2^n - A. \tag{2.6}$$

Мы видим, что в данном представлении содержится «$-A$», благодаря чему оно удобно для проведения вычитания. Учтем также, что 2^n в двоичном представлении имеет $n + 1$ разрядов. Ниже пример точного дополнения для 10101100:

$$A_{K2} = \neg A + 1 = 01010011 + 1 = 01010100$$

Представление 4-битовых двоичных слов в круговой форме, приведенное на рис. 2.1, позволяет показать числовой диапазон. Соответствующее данным значениям наибольшее представляемое положительное число равно 7_{10}, соответствующее данным значениям наибольшее отрицательное число равно -8_{10}. Следовательно, числовой диапазон построен несимметрично, поскольку отрицательное число занимает больше места, чем положительное. Наибольшее и наименьшее представляемые числа можно выразить как:

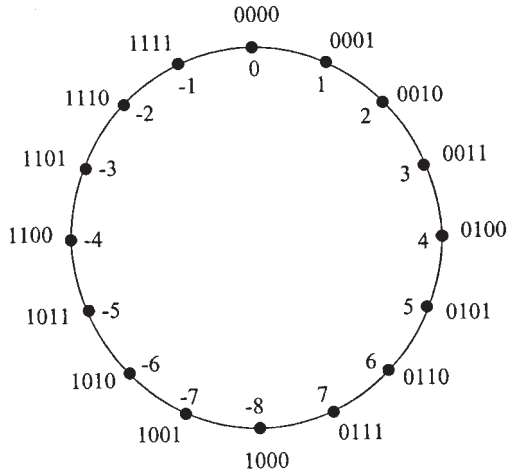
$$z_{max} = 2^{n-1} - 1, \tag{2.7}$$

$$z_{min} = -2^{n-1}. \tag{2.8}$$

На рис. 2.1 можно видеть, что малые числа, сформированные на основе точного дополнения, содержат много ведущих единиц, когда они отрицательны, и содержат много ведущих нулей, когда они положительны. В числах, представленных в формате дополнения до двух, на крайней позиции слева стоит нуль, когда они отрицательны, и единица, когда они положительны. Их собственным точным дополнением является число

1000_2 (-8_{10}). Важно установить, что при представлении на основе точного дополнения имеем только один 0. Это облегчает запрос, равен ли результат 0. Напротив, при представлении на основе поразрядного дополнения имеют место двоичное число 0000_2, соответствующее $+0_{10}$, и двоичное число 1111_2, соответствующее -0_{10}.

Рис. 2.1. Представление 4-битовых слов с помощью 4-битового дополнения



2.3.5. Вычитание при представлении с помощью точного дополнения

Пусть будут вычтены одно из другого два положительных двоичных числа A и B. При условии применения точного дополнения в соответствии с уравнением (2.6) вычитание можно провести следующим образом:

$$A - B = A - B + B_{K2} - B_{K2} = \mathrm{A} - B + B_{K2} - (2^n - B). \qquad (2.9)$$

Раскрытие скобок в правой части уравнения дает:

$$A - B = A + B_{K2} - 2^n. \qquad (2.10)$$

Что означает вычитание 2^n? Поясним это на примере операции вычитания $7 - 3 = 4$ в 4-битовой двоичной системе. Сумма двоичного эквивалента числа 7 и дополнение двоичного эквивалента числа 3 равняется:

$$\begin{array}{rr} & 0\,1\,1\,1 \quad 7_{10} \\ + & 1\,1\,0\,1 \quad -3_{10} \\ \hline = & 1\,0\,1\,0\,0 \\ \hline - & 1\,0\,0\,0\,0 \\ = & 0\,1\,0\,0 \quad 4_{10} \end{array}$$

Вычитание числа 10000_2, проведенное в соответствии с уравнением (2.10), дает правильный результат 0100_2. Это может произойти в 4-битовом компьютере просто потому, что высший результат игнорируется. Итак, при проведении вычитания с помощью точного дополнения нет необходимости учитывать высший перенос c_4. Но необходимо соблюдать осторожность в связи с переполнением числового диапазона. Исследуем это ниже.

2.3.6. Переполнение числового диапазона

Исходя из вышесказанного, возникает необходимость рассмотрения проблемы переполнения числового диапазона (overflow) в связи с представлением на основе точного дополнения. Переполнение числового диапазона может происходить только в двух случаях. А именно, когда суммируются два положительных числа либо суммируются два отрицательных числа. В связи с этим рассмотрим несколько примеров, относящихся к 4-битовому представлению.

- Пример переполнения числового диапазона при сложении двух положительных чисел:

$$\begin{array}{rr} & 0\,1\,0\,1 \quad 5_{10} \\ + & 0\,1\,0\,1 \quad 5_{10} \\ \hline = & (0)\,1\,0\,1\,0 \quad -6_{10} \end{array}$$

  Очевидно, что результат является неправильным. Ошибка возникает за счет переноса 3-го разряда на место 4-го разряда, что приводит к симуляции отрицательного числа. Этот перенос c_3 в представлении, использующем n бит, обычно обозначается как c_{n-1}. Перенос c_4 (в общем случае c_n) из разряда 4 в разряд 5 называется *Carry* (*Cy*). В этом примере данный перенос не имеет места.
- Пример переполнения численного диапазона при сложении отрицательных чисел:

$$\begin{array}{rr} & 1\,0\,1\,1 \quad -5_{10} \\ + & 1\,0\,1\,1 \quad -5_{10} \\ \hline = & (1)\,0\,1\,1\,0 \quad 6_{10} \end{array}$$

  В этом примере также появляется неправильный результат. Имеет место не перенос c_{n-1} из разряда 3 в разряд 4, а перенос c_n из разряда 4 в разряд 5.

- Для сравнения проведем сложение двух отрицательных чисел без переполнения числового диапазона:

$$\begin{array}{rl} & 1\,1\,1\,1 \quad -1_{10} \\ + & 1\,1\,0\,1 \quad -3_{10} \\ \hline = (1)\,1\,1\,0\,0 & \quad -4_{10} \end{array}$$

Имеются переносы c_n и c_{n-1}.

Сведем эти результаты вместе с другими, здесь не показанными случаями в таблицу. На основе результатов, представленных в табл. 2.3, для двух двоичных чисел A и B, которые лежат в числовом диапазоне, определяемом n-битовым представлением на базе точного дополнения, можно установить перенос переполнения при сложении.

Таблица 2.3. Перенос переполнения при сложении в случае n-битового представления на основе точного дополнения

	Правильный результат	Перенос переполнения
A +B	$c_n = 0$, $c_{n-1} = 0$	$c_n = 0$, $c_{n-1} = 1$
A – B	$c_n = c_{n-1}$	невозможен
– A – B	$c_n = 1$, $c_{n-1} = 1$	$c_n = 1$, $c_{n-1} = 0$

Следовательно, правильный результат имеет место тогда, когда $c_n = c_{n-1}$, неправильный результат — когда $c_n \neq c_{n-1}$.

2.3.7. Умножение

Умножение выполняется так же, как и для десятичной системы. Рассмотрим пример умножения на основе двоичной системы для чисел $10_{10} \times 11_{10} = 110_{10}$:

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

Наибольший из ожидаемых результатов E умножения двух n-битовых слов представляет собой:

$$E = (2^n - 1) \cdot (2^n - 1) = 2^{2n} - 2^{n+1} + 1 \leq 2^{2n} - 1$$

Следовательно, результат умножения двух n-битовых чисел имеет длину $2n$ бит. Но он меньше, чем максимальное представляемое с помощью $2n$ бит двоичное число $2^{2n} - 1$.

Сказанное выше справедливо для умножения положительных чисел. При вычислениях с использованием представления на основе точного дополнения могут быть применены специальные алгоритмы [41], или же числа на основе точного дополнения перед умножением следует преобразовать обратно в исходные значения, а результат перевести в соответствии со знаком в желаемое представление.

При умножении чисел с фиксированной запятой вначале числа умножаются без учета запятой. Затем запятая вводится в соответствии с правилом: умножение двух чисел с n и k разрядами после запятой даст произведение с $n + k$ разрядами после запятой.

2.3.8. Деление

Для деления можно использовать тот же самый алгоритм, что и в десятичной системе. Продемонстрируем это на примере уравнения $10_{10} : 2_{10} = 5_D$:

```
  1010 | 0010
- 10    ----
------   101
  010
-  10
------
    0
```

Соответственно, при делении числа с n разрядами после запятой на число с k разрядами после запятой частное имеет $n - k$ разрядов после запятой. Так, в соответствии с верхним примером имеем ($1{,}25_{10} : 0{,}5_{10} = 2{,}5_{10}$): 1,010 : 00,10 = 10,1. Деление чисел с точным дополнением также можно свести к умножению и сложению [41].

2.4. Представление вещественных чисел с плавающей запятой

2.4.1. Введение: представление с плавающей запятой в десятичной системе счисления

Большие числа обычно представляются в экспоненциальном виде с плавающей запятой. Примером может служить представление скорости света: $2{,}99792458 \cdot 10^8$ м/с. В общем виде:

$$z = m \cdot b^e. \tag{2.11}$$

Число m называется мантиссой, а e — показателем степени. Здесь e — целое число, а m — число с фиксированной запятой со знаком. В этом примере мантисса равна 2,99792458, а показатель степени равен 8. Для представления чисел в формате с плавающей запятой характерно наличие основания b. В данном примере основание $b = 10$. И показатель степени, и мантисса имеют знак. В компьютерах используется представление в двоичной системе с основанием $b = 2$.

2.4.2. Представление с плавающей запятой в двоичной системе счисления

В этой главе описывается представление с плавающей запятой в двоичной системе счисления. В информатике эту форму представления числа называют «реальной», поскольку ее можно использовать для представления действительных чисел. Согласно стандарту IEEE-754 в общем случае представление чисел с плавающей запятой имеет следующий вид:

$$z = (-1)^s \cdot 1, m \cdot 2e = (-1)^s \cdot 1, m \cdot 2^{(c-q)}. \tag{2.12}$$

- s — знак мантиссы, он представлен одним битом.
- 1, m — абсолютное значение мантиссы. Оно хранится как двоичное число с фиксированной запятой и перед местом, обозначенным запятой. Оно всегда равно единице. Число m после запятой называется дробной частью. m выбирается согласно выбору показателя степени так, чтобы удовлетворять условию:

  $$2 > |1, m| \geq 1. \tag{2.13}$$

  Условие верно только для чисел $z \neq 0$. Следовательно, нельзя представить нуль. Форма записи с плавающей запятой, удовлетворяющая условию (2.13), называется нормализованной. Мантисса располагает в компьютере фиксированным количеством цифровых позиций n_m.

- Вместо показателя степени e сохраняется характеристика $c = e + q$. Она имеет фиксированное количество цифр n_c. При добавлении остатка показатель степени e сдвигается так, чтобы сохранять только положительные числа. Остаток равен

  $$q = 2^{nc-1} - 1. \tag{2.14}$$

Таким образом, например, для 8 цифр ($n_c = 8$) получим остаток 127. Это означает, что могут быть представлены показатели от −127 до 128. Наи-

меньшая характеристика 0 получается, когда наименьший показатель степени равен –127, и становится равной нулю при добавлении остатка 127. Наибольшая характеристика, а именно наибольшее число 255, которое может быть представлено 8 битами, получается из суммы наибольшего показателя 128 и остатка 127.

Таблица 2.4. Количество цифровых позиций для знака, показателя степени и мантиссы согласно стандарту IEEE-754

	Количество бит на позицию		
	знак n_s	характеристика n_c	мантисса n_m
Одинарная точность	1	8	23
Двойная точность	1	11	52

Согласно стандарту IEEE-745 часто используются две степени точности — одинарная точность и двойная точность.

Два часто используемых числовых формата согласно IEEE-745 — это одинарная точность и двойная точность. Согласно табл. 2.4 используются двоичные цифры $1 + n_c + n_m$. Для чисел в числовом формате одинарной точности требуется 32 бита, в формате двойной точности — 64 бита.

Пример:
Действительное число $-0{,}171875_{10}$ должно быть преобразовано в формат одинарной точности согласно IEEE-745. Первая цифра — это знак *s*. $s = 1$ означает отрицательную мантиссу, $s = 0$ — положительную. В данном случае $s = 1$.

Сначала вы конвертируете десятичное число в двоичное:

$$-0{,}171875_{10} = -0{,}001011_2$$

Затем число нормализуется, и тогда оно будет иметь вид 1, $m \cdot 2^n$:

$$-0{,}0010112 = -1{,}011 \cdot 2^{-3}$$

2^{-3} означает, аналогично десятичной системе счисления, сдвиг на 3 позиции. Мантисса сохраняется без 1 перед десятичной запятой и расширяется до 23 бит с нулями или округляется до 23 разрядов, если необходимо:

$$m = +1{,}011\ 0000\ 0000\ 0000\ 0000\ 0000$$

Избыток *q* добавляется к показателю степени $e = -3$ (для одинарной точности $q = 127$), чтобы получить характеристику *c*. Он сохраняется как 8-значное двоичное число:

$$c = e + q = -3 + 127 = 124_{10} = 01111100_2$$

Теперь число, которое нужно сохранить, формируется путем совмещения *s*, *c* и *m*

s *c* *m*
1 01111100011 0000 0000 0000 0000 0000

Итак, двоичное число, которое мы ищем:
$1011\ 1110\ 0011\ 0000\ 0000\ 0000\ 0000\ 0000_2$.

2.4.3. Особые представления чисел

- По определению нуль не может быть представлен, так как мантисса равна 1, *m*, и следовательно всегда не равна 0. В качестве альтернативы было определено число с мантиссой 1,00 ... и характеристикой 0. Это наименьшее число, которое можно представить, и оно определено в качестве нуля. Поскольку разрешены оба знака, получается два представления нуля.
- Числа с мантиссой 1,00. . . и максимально возможной характеристикой определены для представления ± бесконечности. При представлении бесконечности с одинарной точностью это число 255.
- Кроме того, существует представление под названием *NaN* (Not a Number — не число). Это число не считается числом с плавающей запятой и обычно приводит к отмене вычисления.

Таблица 2.5. Специальные представления нуля, бесконечности и NaN согласно IEEE-754 для одинарной точности

	Знаковая характеристика		Дробь
Ноль	0 или 1	0	0
± бесконечно	0 или 1	255	0
Без числа (*NaN*)	0 или 1	255	≠ 0

2.5. Шестнадцатеричный код

На практике наряду с двоичным кодом внедрился шестнадцатеричный код, поскольку он обеспечивает лучшее обозрение длинных двоичных чисел. Шестнадцать шестнадцатеричных цифр определены в табл. 2.4. Шестнадцатеричные цифры больше девяти представлены буквами *A*–*F*. Для пре-

образования двоичных в шестнадцатеричные числа объединяют по четыре цифры двоичного числа, которые интерпретируются как шестнадцатеричный разряд. Благодаря этому шестнадцатеричное число занимает только четверть разрядов, занимаемых двоичным числом одинаковой величины.

Например:

1011	1110	0011	0000
B	E	3	0

Итак, справедливо выражение $1011111000110000_2 = BE30_{16}$.

Таблица 2.6. Шестнадцатеричные числа

десятичные	двоичные	шестнадцатеричные	десятичные	двоичные	шестнадцатеричные
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

В качестве обозначения шестнадцатеричного числа используется индекс *H* и $. Преобразование шестнадцатеричного числа в десятичное число и обратно проще всего производить через соответствующее двоичное число. Также возможно производить преобразование с помощью алгоритма, как при преобразовании двоичного числа в десятичное число. Обратное преобразование производится аналогично представленному уравнением (2.2).

2.6. Восьмеричный код

Восьмеричный код применяется аналогично шестнадцатеричному, только объединяются лишь по 3 разряда двоичного числа. Для восьмеричного кода применяются цифры десятичного кода от 0 до 7, он обозначается индексом 0.

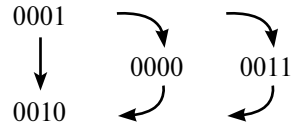
Например:

110	101	100	011
6	5	4	3

Следовательно, справедливо $11010100011_2 = 6543_8$.

2.7. Код Грея

Часто в цифровой технике требуется для числового кода схема кодирования, в соответствии с которой при переходе от одного числа к следующему изменялась бы только одна цифра. Выполнение этого условия необходимо, когда вследствие технических неточностей момент переключения не может быть точно выдержан. Из-за этого при переключении двух цифр могут возникнуть неправильные коммутационные операции. В качестве примера подобной ошибки рассмотрим переключение от 110 к 210 в двоичном коде:



При этом переключении изменяются биты 0 и 1, при одновременном переключении непосредственно достигается новое число. Если вначале изменяется бит 0, то появляется число 0000, и только когда изменяется бит 1, получаем правильное число 0010. Если же сначала изменяется бит 1 и потом изменяется бит 0, то в промежутке возникает число 0011. Коды Грея позволяют избежать этой очень серьезной ошибки за счет того, что при переходе от одного кодового слова к следующему изменяется только один разряд. В табл. 2.7 представлен 4-разрядный код Грея. В дополнение к вышесказанному показанный код имеет свойство, которое заключается в его цикличности, также как и при переходе от высшего числа (15D) к низшему числу изменяется только один разряд. Циклические коды Грея могут быть сконструированы для всех прямых длин периодов.

Таблица 2.7. Пример 4-разрядного кода Грея

Десятичные (числа)	Код Грея	Десятичные (числа)	Код Грея
0	000	4	110
1	001	5	111
2	011	6	101
3	010	7	100

2.8. Двоично-десятичный код

Если мы хотим подвести к десятичным цифрам некоторые отметки, то для этого подойдет код, в котором отдельным десятичным цифрам при-

даны двоично-кодированные кодовые слова. Этот код обозначается как двоично-десятичный код (BCD-код). Подходящая возможность реализации заключается в том, чтобы каждой десятичной цифре сопоставить ее запись в двоичной системе счисления, то есть четырехразрядное двоичное число (см. табл. 2.6). Поскольку отдельные разряды имеют веса 8, 4, 2 и 1, данный код называют кодом типа 8-4-2-1. Существует также возможность построить двоично-десятичный код 2, 4, 2, 1 (Aiken-Code). К другим двоичнодесятичным кодам относятся код с избытком три (3-Excess-Code) и двоично-десятичный код Грея (BCD-Gray-Code) [5].

Таблица 2.8. Двоично-десятичный код

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

2.9. Алфавитно-цифровые коды

Существует большое количество кодов, осуществляющих представление алфавитно-цифровых символов с помощью двоичных цифр.

Известным примером подобных символов является код ASCII (American Standard Code for Information Interchange), который содержит также ряд управляющих символов.

Таблица 2.9. Код ASCII

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

2.10. Упражнения

Задача 2.1. Преобразуйте следующие двоичные числа в десятичные числа:

а. 1110,101

б. 10011,1101

Задача 2.2. Преобразуйте следующие десятичные числа в двоичные числа:

а. 33,125

б. 45,33

Задача 2.3. Рассчитайте приведенные ниже примеры с использованием точных дополнений при длине слова в 6 бит. Укажите, имеет ли место переполнение числового диапазона.

а. 010101 –001010

б. –010111 –011011

Задача 2.4. Рассчитайте в двоичной системе:

а. 110101 · 010101

б. 1101110 : 110

Задача 2.5. Разработайте циклический код Грея с длиной периода 6.

Задача 2.6. Какое действительное число представлено числом одинарной точности $C23A8000_{16}$ согласно IEEE-754?

ГЛАВА 3

ПЕРЕКЛЮЧАТЕЛЬНАЯ АЛГЕБРА

Цифровая техника имеет преимущество перед аналоговой техникой: она основывается на относительно простой, но при этом мощной теории, на булевой алгебре, называемой также переключательной алгеброй. В данной главе представлены теоретические основы цифровой техники. Булева алгебра может быть применена для решения почти всех проблем, возникающих при разработке цифровых схем с условием выполнения технологических предпосылок, которые рассматриваются в главе 4.

3.1. Переключательная переменная и переключательная функция

В цифровой технике применяют специальные переменные и функции. Под булевой переменной понимают переменную, которая может принимать только значения 0 и 1. С булевыми переменными могут быть образованы функции. Функцию

$$y = f x_1, x_2, x_3, \ldots x_n \quad \text{при } x_i, y \in \{0, 1\} \tag{3.1}$$

называют n-разрядной переключательной или двоичной функцией. Область значений подобной функции также определяется двоичной системой счисления с элементами 0 и 1. Функции могут быть определены таблицами, в которых величины функций связываются с возможными 2^n комбинациями n входных переменных. Эти таблицы называются таблицами истинности.

Очень простая функция, которая связывает входную переменную x с выходной переменной y, представлена в табл. 3.1. Можно видеть, что все переключательные функции могут быть определены таблицей, содержащей все входные переменные, поскольку учитывать необходимо только два элемента 0 и 1.

Определенную табл. 3.1 переключательную функцию $y = f(x)$ именуют «отрицание», «дополнение» или НЕТ (NOT). В последующем она обозначается оператором « $\neg$ » и читается «y равен не x»:

$$y = \neg x. \tag{3.2}$$

Таблица 3.1. Таблица истинности инвертора:

x	y
0	1
1	0

«Отрицание» является одноразрядной переключательной функцией, поскольку она обладает только одним входным символом. Схемный элемент «инвертор» обозначается с помощью схемного обозначения, представленного на рис. 3.1.

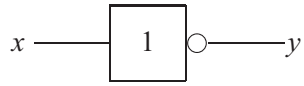
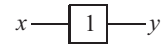
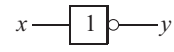


Рис. 3.1. Схемное обозначение инвертора

Имеются ли еще одноразрядные переключательные функции? Путем систематических проб находим их общим числом 4, все они сведены в табл. 3.2. Других видов одноразрядной переключательной функции y не существует. Переключательные функции $y = 0$ и $y = 1$ вырабатывают постоянные, не зависящие от входа. Поэтому существенной для переключательной алгебры является только одноразрядная двоичная функция $y = \neg x$.

Таблица 3.2. Одноразрядные двоичные функции

Таблица истинности	Функции	Схемное обозначение	Наименование
x | y 0 | 0 1 | 0	$y = 0$		
x | y 0 | 0 1 | 1	$y = x$		
x | y 0 | 1 1 | 0	$y = \neg x$		НЕТ, «дополнение», «отрицание», «инверсия»
x | y 0 | 1 1 | 1	$y = 1$		

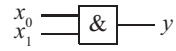
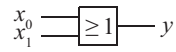
3.2. Двухразрядные переключательные функции

В принципе можно образовывать произвольные двоичные функции многих входных переменных. Но оказалось практичным вначале рассматривать только функции с одной или двумя входными переменными, а функции с большим количеством входных переменных сводить к ним.

Двоичную функцию с входными переменными x_0 и x_1 также можно определить с помощью таблицы. Комбинацию входных переменных x_0 и x_1 можно представить в виде вектора $X = [x_0, x_1]$. Двум входным переменным соответствуют 4 возможных входных вектора X, которые часто индицируются через их десятичные эквиваленты. Так, x_2 означает, что $x_1 = 1$ и $x_0 = 0$, или, выражаясь по-другому, что $X_2 = [x_1, x_0] = [1, 0]$.

Наряду с «отрицанием» технически возможными являются основные логические элементы И или ИЛИ, определяемые табл. 3.3. Также обозначают И как AND или «конъюнкция»; ИЛИ как OR или «дизъюнкция».

Таблица 3.3. Основные логические элементы И и ИЛИ

Таблица истинности	Функции	Схемное обозначение	Наименование
x_1 | x_0 | y 0 | 0 | 0 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1	$y = x_0 \wedge x_1$		И, AND, «конъюнкция»
x_1 | x_0 | y 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0	$y = x_0 \vee x_1$		ИЛИ, OR, «дизъюнкция»

Возникает вопрос о других возможных двухразрядных двоичных функциях. Чтобы системно ответить на этот вопрос, надо осуществить перестановку значений входных переменных y, которые можно получить из четырех возможных входных векторов. В общем случае функция $y(x_1, x_0)$ может быть определена таблицей истинности (табл. 3.4).

На основе этой таблицы можно сделать заключение о возможности образования $2^{2^n} = 16$ различных двоичных функций с двумя входными переменными. Все вероятные двухразрядные двоичные функции приведены в табл. 3.5. Представление двоичных функций выполнено в соответствии со стандартом DIN [10].

Таблица 3.4. Таблица истинности для двухразрядной двоичной функции

x_1	x_0	y
0	0	$y(0, 0)$
0	1	$y(0, 1)$
1	0	$y(1, 0)$
1	1	$y(1, 1)$

Технически важным функциям NAND, NOR, «эквивалентность» и EXOR (также «исключающее ИЛИ», дизъюнкция) соответствуют собственные логические символы. На практике они часто реализуются на основе специальных схем.

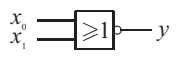
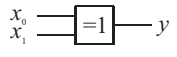
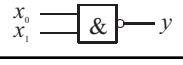
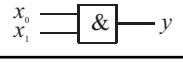
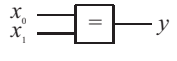
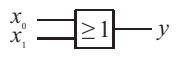
В этой таблице показано, что отдельные функции могут быть представлены только с помощью логических элементов AND, OR и NOT. Поэтому каждая логическая функция может быть представлена с помощью этих трех логических элементов. Все двоичные функции могут быть представлены как с помощью функции NOR, так и с помощью NAND. Поэтому эти функции называют совершенными.

Доказательство эквивалентности можно привести путем размещения таблиц истинности. Таким образом, например (см. табл. 3.5), может быть доказана эквивалентность для логического элемента EXOR на основе AND, OR или NOT:

$$x_0 \leftrightarrow x_1 = \neg x_0 \wedge x_1 \vee x_0 \wedge \neg x_1. \tag{3.3}$$

В табл. 3.6 вначале оцениваются оба выражения в скобках. Затем формируется логическое OR, соответствующее обоим выражениям в скобках, и вписывается в пустую колонку. Так как последняя и предпоследняя колонки совпадают, уравнение доказано, поскольку в последней колонке стоит определение функции «исключающее ИЛИ» (то есть EXOR).

Таблица 3.5. Доказательство на основе таблицы истинности

Таблица истинности					Функции	Схемное обозначение	Наименование
x_0	1	0	1	0			
x_1	1	1	0	0			
y	0	0	0	0	$y = 0$		нуль
y	0	0	0	1	$y = \neg(x_0 \vee x_1)$ $y = (x_0 \overline{\vee} x_1)$		NOR
y	0	0	1	0	$y = x_0 \wedge \neg x_1$		запрет
y	0	0	1	1	$y = \neg x_1$		дополнение
y	0	1	0	0	$y = \neg x_0 \wedge x_1$		запрет
y	0	1	0	1	$y = \neg x_0$		дополнение
y	0	1	1	0	$y = (\neg x_0 \wedge x_1) \vee$ $\vee (x_0 \neq x_1)$ $y = (x_0 \leftrightarrow x_1)$		EXOR
y	0	1	1	1	$y = \neg(x_0 \wedge x_1)$ $y = (x_0 \overline{\wedge} x_1)$		NAND
y	1	0	0	0	$y = x_0 \wedge x_1$		AND
y	1	0	0	1	$y = (x_0 \wedge x_1) \vee$ $\vee (\neg x_0 \wedge x_1)$ $y = (x_0 \leftrightarrow x_1)$		эквивалентность
y	1	0	1	0	$y = x_0$		идентичность
y	1	0	1	1	$y = x_0 \vee \neg x_1$		импликация
y	1	1	0	0	$y = x_1$		идентичность
y	1	1	0	1	$y = x_1 \vee \neg x_0$		импликация
y	1	1	1	0	$y = x_0 \vee x_1$		OR
y	1	1	1	1	$y = 1$		единица