Серия основана в 2000 году

А.Я.Савельев

Основы информатики

Допущено Министерством образования
Российской Федерации
в качестве учебника для студентов
высших учебных заведений,
обучающихся по направлению подготовки
дипломированных специалистов
«Информатика и вычислительная техника»

Москва
Издательство МГТУ имени Н.Э.Баумана
2001

УДК 002 (075.8)
ББК 73
С12

Рецензенты:

Кафедра «Автоматизированные системы и вычислительная техника» Московского государственного университета (зав. кафедрой академик МАН ВШ, д-р техн. наук, проф. В. И. Карпов);
д-р техн. наук, проф. В. В. Семенов (МАН)

С12 **Савельев А. Я.** Основы информатики: Учеб. для вузов. — М.: Изд-во МГТУ им. Н. Э. Баумана, 2001. — 328 с.,ил. (Сер. Информатика в техническом университете).

ISBN 5–7038–1515–0

Излагаются основные понятия информатики, методы измерения и представления информации, способы представления числовой информации в информационных системах, методы и алгоритмы выполнения арифметических и логических операций в различных системах счисления. Уделяется внимание методам контроля правильности функционирования цифровых автоматов как основному элементу информационных систем, а также способам защиты информации в вычислительных и информационных системах. Рассматриваются методы логического описания и проектирования на основе использования аппарата теории булевых функций, теории автоматов и автоматных языков.

Содержание учебника соответствует курсу лекций, который автор читает в МГТУ им. Н. Э. Баумана.

Для студентов высших технических учебных заведений.

УДК 002 (075.8)
ББК 73

ISBN 5–7038–1515–0

Оглавление

ПРЕДИСЛОВИЕ

Представленная вниманию читателей книга «Введение в информатику» — первая книга из многотомной серии «Информатика в техническом университете».

В учебных планах подготовки дипломированного специалиста по направлению «Информатика и вычислительная техника» дисциплина «Информатика» входит в состав фундаментального цикла дисциплин. Она создает теоретическую базу для изложения и понимания таких дисциплин, как «Организация ЭВМ и систем», «Алгоритмические языки и программирование», «Теория автоматов» и других специальных курсов.

Основная цель книги — познакомить студента с понятиями информатики, изложить методы и средства представления информации в компьютерах и информационных системах, методы реализации арифметических и логических операций в цифровых автоматах, а также основы анализа и синтеза логических схем ЭВМ и информационных систем.

Следует напомнить читателю, что термин «информатика» впервые был использован французскими специалистами для определения комплекса задач, связанных с применением ЭВМ для обработки, хранения и преобразования разнообразного вида информации. Однако этот термин не является общепризнанным, так как ряд стран, а именно — США, Канада и некоторые латиноамериканские государства — придерживаются термина «компьютерные науки». Вместе с тем, на Международном Конгрессе «Образование и информатика» (1996 год, Москва) отмечалось, что «информатика сегодня — это одна из важных и перспективных «точек роста» мировой науки, вокруг которой формируется новый комплекс наук об информации».

Главным понятием курса «Информатика» является понятие цифрового автомата как средства для представления и обработки любых видов информации. Автор здесь придерживается классического определения, введенного академиком В. М. Глушковым: «Электронные цифровые машины с программным управлением представляют собой пример одного из наиболее распространенных в настоящее время преобразователей дискретной информации, называемых дискретными или цифровыми автоматами».

В главах 1, 3, 8 и 9 изложены способы представления и кодирования информации, методы контроля и защиты информации. Этот материал основан на результатах работ К. Шеннона, В. А. Котельникова, Р. Хэминга и др.

Главы 4, 5, 6 и 7 содержат описание методов и алгоритмов выполнения арифметических операций на двоичных и десятичных сумматорах. Их основой послужили работы С. А. Лебедева, Р. К. Ричардса, В. А. Мельникова и др. Материал изложен таким образом, чтобы достичь максимальной общности содержания при возможности практического использования результатов теории. При этом используется некая гипотетическая ЭВМ, обладающая определенной структурой, описанной в примерах.

Главы 2, 10 и 11 посвящены логическим основам анализа и синтеза цифровых автоматов. Этот материал книги базируется на трудах Д. Буля, В. И. Шестакова, Э. Мак-Класки и др.

В главах 2, 11 и 12 изложены некоторые положения теории цифровых автоматов, которые найдут дальнейшее развитие в специальном курсе «Теория цифровых автоматов». Основой материала этих глав являются труды В. М. Глушкова, Дж. фон-Неймана, Э. Поста, А. Тьюринга, С. В. Яблонского и др. Читатель, интересующийся более глубоко вопросами анализа и синтеза цифровых автоматов, найдет эти сведения в специальной литературе. Для лучшего усвоения материала глава 12 дополнена конкретными примерами синтеза двух управляющих автоматов.

В основу книги положен курс лекций по данной дисциплине, читаемый автором в течение нескольких лет в Московском государственном техническом университете имени Н. Э. Баумана.

Автор

1. БАЗОВЫЕ ПОНЯТИЯ ИНФОРМАТИКИ

1.1. Общие сведения об информации

Прежде всего определим, что такое вычислительная машина. Интуитивно понятно, что это — средство для автоматизации вычислений. Однако вычислительные машины используются настолько широко и для решения такого обширного круга задач (от вычислений до составления меню в ресторане и даже сочинения музыки), что поневоле возникает сомнение в правильности интуитивного определения.

В «Энциклопедии кибернетики» [20] приведено следующее определение: «*Вычислительная машина (ВМ) — физическая система (устройство или комплекс устройств), предназначенная для механизации или автоматизации процесса алгоритмической обработки информации и вычислений*». Таким образом, понятие «вычислительная машина» самым тесным образом связано с понятиями «информация» и «алгоритмическая обработка».

Объект передачи и преобразования в вычислительных системах (машинах) — информация. В этом смысле вычислительную машину (систему) можно называть информационной, в отличие, например, от энергетической системы, где объект передачи и преобразования — энергия. Все процессы, происходящие в вычислительной системе, связаны непосредственно с различными физическими носителями информационных сообщений, и все узлы и блоки этой системы являются физической средой, в которой осуществляются информационные процессы. Специфика информационных процессов состоит не только в передаче информационных сообщений посредством заданной физической среды, но и в преобразовании, переработке и хранении информации. Все это составляет предмет науки информатики. **Информатика** представляет собой неразрывное единство трех составных частей: теории передачи и преобразования информации; алгоритмических средств обработки информации и вычислительных средств.

Информация определяет многие процессы в вычислительной машине. В самой общей форме процесс решения задачи на вычислительной машине проходит через следующие этапы:

— ввод информации или установка исходных данных;

— переработка или преобразование введенной информации;

— определение результатов и вывод переработанной информации.

Вычислительная машина получает информацию, запоминает ее, обрабатывает по заданным алгоритмам и направляет потребителю (пользователю) или в другие системы обработки (рис. 1.1).

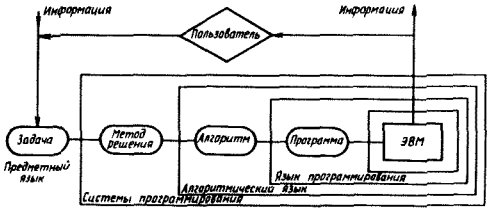


Рис. 1.1. Схема взаимодействия пользователя с компьютером

Термин «информация» имеет много определений. В широком смысле *информация — отражение реального мира.* Существует определение термина в узком смысле: *информация — любые сведения, являющиеся объектом хранения, передачи и преобразования.* Оба определения важны для понимания процессов функционирования вычислительной машины.

Важный вопрос теории передачи и преобразования информации — установление меры, количества и качества информации.

Информационные меры, как правило, рассматриваются в трех аспектах: структурном, статистическом и семантическом.

В **структурном** аспекте рассматривается строение массивов информации и их измерение простым подсчетом информационных элементов или комбинаторным методом. Структурный подход применяется для оценки возможностей информационных систем вне зависимости от условий их применения.

При **статистическом** подходе используется понятие энтропии как меры неопределенности, учитывающей вероятность появления и информативность того или иного сообщения. Статистический подход учитывает конкретные условия применения информационных систем.

Семантический подход позволяет выделить полезность или ценность информационного сообщения.

1.2. Структурная мера информации

Информация всегда представляется в виде сообщения. Элементарная единица сообщений — *символ*. Символы, собранные в группы, — *слова*. Сообщение, оформленное в виде слов или отдельных символов, всегда передается в материально-энергетической форме (электрический, световой, звуковой сигналы и т. д.).

Различают информацию *непрерывную* и *дискретную*.

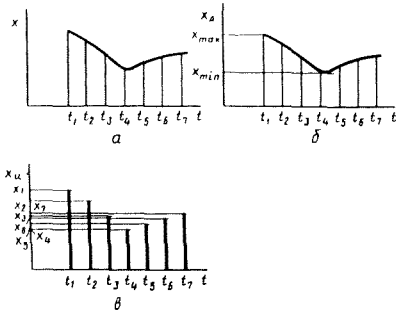


Рис. 1.2. Способы представления информации

Функция $x(t)$, изображенная на рис. 1.2, *а*, может быть представлена в непрерывном (рис. 1.2, *б*) и дискретном (рис. 1.2, *в*) видах. В непрерывном виде эта функция может принимать любые вещественные значения в данном диапазоне изменения аргумента t, т. е. множество значений непрерывной функции бесконечно. В дискретном виде функция $x(t)$ может принимать вещественные значения только при определенных значениях аргумента. Какой бы малый интервал дискретности (т. е. расстояние между соседними значениями аргумента) не выбирался, множество значений дискретной функции для заданного диапазона изменений аргумента (если он не бесконечный) будет конечно (ограничено).

При использовании структурных мер информации учитывается только дискретное строение сообщения, количество содержащихся в нем информационных элементов, связей между ними. При структурном подходе различаются геометрическая, комбинаторная и аддитивная меры информации.

Геометрическая мера предполагает измерение параметра геометрической модели информационного сообщения (длины, площади, объема и т. п.) в дискретных единицах. Например, геометрической моделью информации может быть линия единичной длины (рис 1.3, *а* — одноразрядное слово, принимающее значение 0 или 1), квадрат (рис. 1.3, *б* — двухразрядное слово) или куб (рис 1.3, *в* — трехразрядное слово). Максимально возможное количество информации в заданных структурах определяет информационную емкость модели (системы), которая определяется как сумма дискретных значений по всем измерениям (координатам).

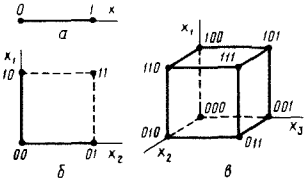


Рис. 1.3. Геометрическая модель информации

В комбинаторной мере количество информации определяется как число комбинаций элементов (символов). Возможное количество информации совпадает с числом возможных сочетаний, перестановок и размещений элементов. Комбинирование символов в словах, состоящих только из 0 и 1, меняет значения слов. Рассмотрим две пары слов 100110 и 001101, 011101 и 111010. В них проведена перестановка крайних разрядов (изменено местоположение знакового разряда в числе — перенесен слева направо).

Аддитивная мера (*мера Хартли*), в соответствии с которой количество информации измеряется в двоичных единицах — битах, — наиболее распространена. Вводятся понятия глубины q и длины n числа.

Глубина q числа — количество символов (*элементов*), *принятых для представления информации.* В каждый момент времени реализуется только один какой-либо символ.

Длина n числа — количество позиций, необходимых и достаточных для представления чисел заданной величины.

В гл. 3 понятие глубины числа будет трансформировано в понятие основания системы счисления. При заданных глубине и длине числа количество чисел, которое можно представить, $N = q^n$. Величина N не удобна

для оценки информационной емкости. Введем логарифмическую меру, позволяющую, вычислять количество информации, — бит:

$$I(g) = \log_2 N = n \log_2 q . \qquad (1.1)$$

Следовательно, 1 бит информации соответствует одному элементарному событию, которое может произойти или не произойти. Такая мера количества информации удобна тем, что она обеспечивает возможность оперировать мерой как числом. Количество информации при этом эквивалентно количеству двоичных символов 0 или 1. При наличии нескольких источников информации общее количество информации

$$I(q_1, q_2, \ldots, q_k) = I(q_1) + I(q_2) + \ldots + I(q_k), \qquad (1.2)$$

где $I(q_k)$ – количество информации от источника k.

Логарифмическая мера информации позволяет измерять количество информации и используется на практике.

1.3. Статистическая мера информации

В статистической теории информации вводится более общая мера количества информации, в соответствии с которой рассматривается не само событие, а информация о нем. Этот вопрос глубоко проработан К. Шенноном в работе «Избранные труды по теории информации». Если появляется сообщение о часто встречающемся событии, вероятность появления которого близка к единице, то такое сообщение для получателя малоинформативно. Столь же малоинформативны сообщения о событиях, вероятность появления которых близка к нулю.

События можно рассматривать как возможные исходы некоторого опыта, причем все исходы этого опыта составляют ансамбль, или полную группу событий. К. Шенион ввел понятие неопределенности ситуации, возникающей в процессе опыта, назвав ее энтропией. *Энтропия* ансамбля есть количественная мера его неопределенности и, следовательно, информативности, количественно выражаемая как средняя функция множества вероятностей каждого из возможных исходов опыта.

Пусть имеется N возможных исходов опыта, из них k разных типов, а i-й исход повторяется n_i раз и вносит информацию, количество которой оценивается как I_i. Тогда средняя информация, доставляемая одним опытом,

$$I_{\text{ср}} = (n_1 I_1 + n_2 I_2 + \ldots + n_k I_k)/N . \qquad (1.3)$$

Но количество информации в каждом исходе связано с его вероятностью p_i и выражается в двоичных единицах (битах) как $I_i = \log_2(1/p_i) = -\log_2 p_i$. Тогда

$$I_{ср} = [n_1(-\log_2 p_1) + \ldots + n_k(-\log_2 p_k)]/N. \quad (1.4)$$

Выражение (1.4) можно записать также в виде

$$I_{ср} = \frac{n_1}{N}(-\log_2 p_1) + \frac{n_2}{N}(-\log_2 p_2) + \ldots + \frac{n_k}{N}(-\log_2 p_k). \quad (1.5)$$

Но отношения n/N представляют собой частоты повторения исходов, а следовательно, могут быть заменены их вероятностями: $n_i/N = p_i$, поэтому средняя информация в битах

$$I_{ср} = p_1(-\log_2 p_1) + \ldots + p_k(-\log_2 p_k),$$

или

$$I_{ср} = -\sum_{i=1}^{k} p_i \log_2 p_i = H. \quad (1.6)$$

Полученную величину называют энтропией и обозначают обычно буквой H. Энтропия обладает следующими свойствами:

1. Энтропия всегда неотрицательна, так как значения вероятностей выражаются величинами, не превосходящими единицу, а их логарифмы — отрицательными числами или нулем, так что члены суммы (1.6) — неотрицательны.

2. Энтропия равна нулю в том крайнем случае, когда одно из p_i равно единице, а все остальные — нулю. Это тот случай, когда об опыте или величине все известно заранее и результат не дает новую информацию.

3. Энтропия имеет наибольшее значение, когда все вероятности равны между собой: $p_1 = p_2 = \ldots = p_k = 1/k$. При этом

$$H = -\log_2(1/k) = \log_2 k.$$

4. Энтропия объекта AB, состояния которого образуются совместной реализацией состояний A и B, равна сумме энтропий исходных объектов A и B, т. е. $H(AB) = H(A) + H(B)$.

Если все события равновероятны и статистически независимы, то оценки количества информации, по Хартли и Шеннону, совпадают. Это свидетельствует о полном использовании информационной емкости систе-

мы. В случае неравных вероятностей количество информации, по Шеннону, меньше информационной емкости системы. Максимальное значение энтропии достигается при $p = 0{,}5$, когда два состояния равновероятны. При вероятностях $p = 0$ или $p = 1$, что соответствует полной невозможности или полной достоверности события, энтропия равна нулю.

Количество информации только тогда равно энтропии, когда неопределенность ситуации снимается полностью. В общем случае нужно считать, что количество информации есть уменьшение энтропии вследствие опыта или какого-либо другого акта познания. Если неопределенность снимается полностью, то информация равна энтропии: $I = H$.

В случае неполного разрешения имеет место частичная информация, являющаяся разностью между начальной и конечной энтропией: $I = H_1 - H_2$.

Наибольшее количество информации получается тогда, когда полностью снимается неопределенность, причем эта неопределенность была наибольшей — вероятности всех событий были одинаковы. Это соответствует максимально возможному количеству информации I^1, оцениваемому мерой Хартли:

$$I^1 = \log_2 N = \log_2 (1/p) = -\log_2 p,$$

где N — число событий; p — вероятность их реализации в условиях равной вероятности событий.

Таким образом, $I^1 = H_{max}$.

Абсолютная избыточность информации $D_{абс}$ представляет собой разность между максимально возможным количеством информации и энтропией: $D_{абс} = I^1 - H$, или $D_{абс} = H_{max} - H$.

Пользуются также понятием относительной избыточности

$$D = (H_{max} - H)/H_{max}. \qquad (1.7)$$

1.4. Семантическая мера информации

Вычислительные машины обрабатывают и преобразуют информацию разного содержания — от числовых данных до сочинения музыки и стихов. Вся эта информация изображается соответствующими символами. Оценка содержания разнохарактерной информации — весьма сложная проблема.

Среди семантических мер наиболее распространены содержательность, логическое количество, целесообразность и существенность информации.

Содержательность события i выражается через функцию меры $m(i)$ — содержательности его отрицания. Оценка содержательности основана на математической логике, в которой логические функции истинности $m(i)$ и ложности $m(\bar{i})$ имеют формальное сходство с функциями вероятностей события $p(i)$ и антисобытия $q(i)$ в теории вероятностей.

Как и вероятность, содержательность события изменяется в пределах $0 \le m(i) \le 1$.

Логическое количество информации I_{mf}, сходное со статистическим количеством информации, вычисляется по выражению:

$$I_{mf} = \log_2[1/m(i)] = -\log_2 m(\bar{i}).$$

Отличие статистической оценки от логической состоит в том, что в первом случае учитываются вероятности реализации тех или иных событий, что приближает к оценке смысла информации.

Если информация используется в системах управления, то ее полезность целесообразно оценивать по тому эффекту, который она оказывает на результат управления.

Мера *целесообразности* информации определяется как изменение вероятности достижения цели при получении дополнительной информации. Полученная информация может быть пустой, т. е. не изменять вероятности достижения цели, и в этом случае ее мера равна нулю. В других случаях полученная информация может изменять положение дела в худшую сторону, т. е. уменьшить вероятность достижения цели, и тогда она будет дезинформацией, измеряющейся отрицательным значением количества информации. Наконец, в благоприятном случае получается добротная информация, которая увеличивает вероятность достижения цели и измеряется положительной величиной количества информации.

Мера целесообразности в общем виде может быть аналитически выражена в виде соотношения

$$I_{\text{цел}} = \log_2 p_1 - \log_2 p_0 = \log_2 \frac{p_1}{p_0}, \qquad (1.8)$$

где p_0 и p_1 — начальная (до получения информации) и конечная (после получения информации) вероятности достижения цели.

Следует различать существенность самого события; существенност времени совершения события или его наблюдения (рано—поздно— момент); существенность координаты совершения события.

Измерение некоторого параметра X можно характеризовать несколь кими функциями величины x: вероятностью $p(x)$, погрешностью измере ния $\varepsilon(x)$ и существенностью $c(x)$. Каждой из этих функций можно поста вить в соответствие определенную меру информации. Мерой Хартл оценивается функция погрешности ε при фиксированных значениях функ ции вероятности ($p = \text{const}$) и существенности ($c = \text{const}$). Мерой Шеннон оценивается функция вероятности ($p = \text{var}$) при фиксированных значения функций погрешности ($\varepsilon = \text{const}$) и существенности ($c = \text{const}$). Мера су щественности относится к ситуации с фиксированными функциями пс грешности ($\varepsilon = \text{const}$) и вероятности ($p = \text{const}$). Можно ввести функци существенности: c_x, зависящие от x; c_T, c_N, зависящие от времени T пространства (канала) N.

1.5. Преобразование информации

Информационное сообщение всегда связано с источником информа ции, приемником информации и каналом передачи.

Дискретные сообщения состоят из конечного множества элементо создаваемых источником последовательно во времени. Набор элементо (символов) составляет алфавит источника.

Непрерывные сообщения задаются какой-либо физической величиной изменяющейся во времени. Получение конечного множества сообщений з конечный промежуток времени достигается путем *дискретизации* (во вр мени), *квантования* (по уровню) (см. рис 1.2).

В большинстве случаев информация о протекании того или иного ф зического процесса вырабатывается соответствующими датчиками в вид сигналов, непрерывно изменяющихся во времени. Переход от аналогово представления сигнала к цифровому дает в ряде случаев значительные пр имущества при передаче, хранении и обработке информации. Преобразов ние осуществляется с помощью специальных устройств — преобразоват лей непрерывных сигналов и может быть выполнено дискретизацией в времени и квантованием по уровню.

Рассмотрим разновидности сигналов, которые описываются фун цией $x(t)$.

1. Непрерывная функция непрерывного аргумента. Значения, которые могут принимать функция $x(t)$ и аргумент t, заполняют промежутки (x_{min}, x_{max}) и $(-T, T)$ соответственно.

2. Непрерывная функция дискретного аргумента. Значения функции $x(t)$ определяются лишь на дискретном множестве значений аргумента t_i, $i = 0 \pm 1 \pm 2, \ldots$ Величина $x(t_i)$ может принимать любое значение в интервале (x_{min}, x_{max}).

3. Дискретная функция непрерывного аргумента. Значения, которые может принимать функция $x(t)$, образуют дискретный ряд чисел x_1, $x_2, \ldots, x_k$. Значение аргумента t может быть любым в интервале $(-T, T)$

4. Дискретная функция дискретного аргумента. Значения, которые могут принимать функция $x(t)$ и аргумент t, образуют дискретные ряды чисел $x_1, x_2, \ldots, x_k$ и $t_1, t_2, \ldots, t_k$, заполняющие интервалы (x_{min}, x_{max}) и $(-T, T)$ соответственно.

Первая из рассмотренных разновидностей принадлежит непрерывным сигналам, вторая и третья — дискретно-непрерывным, а четвертая — дискретным сигналам.

Операцию, переводящую информацию непрерывного вида в информацию дискретного вида, называют квантованием по времени, или дискретизацией. Следовательно, дискретизация состоит в преобразовании сигнала $x(t)$ непрерывного аргумента t в сигнал $x(t_i)$ дискретного аргумента t_i.

Квантование по уровню состоит в преобразовании непрерывного множества значений сигнала $x(t_i)$ в дискретное множество значений x_k, $k = 0, 1, \ldots, (m-1)$; $x_k \in (x_{min}, x_{max})$ (третий вид сигнала).

Совместное применение операций дискретизации и квантования по уровню позволяет преобразовать непрерывный сигнал $x(t)$ в дискретный по координатам x и t (четвертая разновидиость).

В результате дискретизации исходная функция $x(t)$ заменяется совокупностью отдельных значений $x(t_1)$. По значениям функции $x(t_i)$ можно восстановить исходную функцию $x(t)$ с некоторой погрешностью. Функцию, полученную в результате восстановления (интерполяции) по значениям $x(t_i)$, будем называть *воспроизводящей* и обозначать $V(t)$.

При дискретизации сигналов приходится решать вопрос о том, как часто следует проводить отсчеты функции, т. е. каков должен быть шаг дискретизации $\Delta t_i = t_i - t_{i-1}$. При малых шагах дискретизации количество от-

счетов функции на отрезке обработки будет большим и точность воспроизведения — высокой. При больших шагах дискретизации количество отсчетов уменьшается, но при этом, как правило, снижается точность восстановления. Оптимальной является такая дискретизация, которая обеспечивает представление исходного сигнала с заданной точностью при минимальном количестве отсчетов.

Методы дискретизации и восстановления информации классифицируются в зависимости от регулярности отсчета, критерия оценки точности дискретизации и восстановления, вида базисной функции, принципа приближения.

Регулярность отсчета определяется равномерностью дискретизации.

Дискретизация называется равномерной (рис. 1.4, *а*), если длительность интервалов $\Delta t_i = \text{const}$ на всем отрезке обработки сигнала. Методы равномерной дискретизации широко применяют, так как алгоритмы и аппаратура для их реализации достаточно просты. Однако при этом возможна значительная избыточность отсчетов.

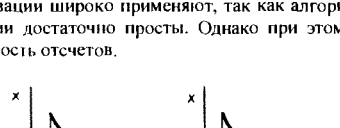


Рис. 1.4. Способы дискретизации информации

Дискретизация называется неравномерной (рис. 1.4, *б*), если длительность интервалов между отсчетами Δt_i различна, т. е. $\Delta t_i = \text{var}$. Выделяют две группы неравномерных методов: адаптивные и программируемые. При адаптивных методах интервалы Δt_i изменяются в зависимости от текущего изменения параметров сигналов. При программируемых методах интервалы Δt_i изменяются либо оператором на основе анализа поступающей информации, либо в соответствии с заранее установленной программой работы.

Критерии оценки точности дискретизации сигнала выбираются получателем информации и зависят от целевого использования сигнала и возможностей аппаратной (программной) реализации. Чаще других используются критерий наибольшего отклонения, среднеквадратический интегральный и вероятностный.

Тип базисных (приближающих, воспроизводящих) функций в основном определяется требованиями ограничения сложности устройств (программ) дискретизации и восстановления сигналов.

Воспроизводящие функции $v(t)$ обычно совпадают с *приближающими* функциями $p(t)$, хотя в общем случае они могут отличаться друг от друга. Чаще всего для дискретизации и восстановления используют ряды Фурье и Котельникова, полиномы Чебышева и Лежандра, степенные полиномы, функции Уолша и Хаара, гипергеометрические функции.

При равномерной дискретизации шаг Δt и частота отсчетов F_0 — постоянные величины (рис. 1.4, *а*). Модель равномерной дискретизации очень хорошо подходит к модели синхронных автоматов. Теорема Котельникова позволяет осуществлять выбор шага дискретизации, что существенным образом может повлиять на количество и скорость поступления информации для обработки.

Квантование по уровню состоит в преобразовании непрерывных значений сигнала $x(t_i)$ в моменты отсчета t_i, в дискретные значения (рис. 1.5). В соответствии с графиком изменения функции $x(t)$ ее истинные значения представляются в виде заранее заданных дискретных уровней *1*, *2*, *3*, *4*, *5* или *6*. Функция в моменты отсчета может задаваться или точно (значение x_2 — уровень *4*), или с некоторой погрешностью (значение x_1 — уровень *2*, значение x_3 — уровень *6*).

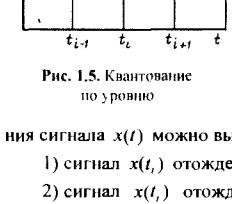


Рис. 1.5. Квантование по уровню

Квантование по уровню может быть равномерным и неравномерным в зависимости от величины шага квантования. Под шагом (интервалом) квантования δ_m понимается разность $\delta_m = x_m - x_{m-1}$, где x_m, x_{m-1} — соседние уровни квантования.

Уровень квантования для заданного значения сигнала $x(t)$ можно выразить двумя способами:

1) сигнал $x(t_i)$ отождествляется с ближайшим уровнем квантования;

2) сигнал $x(t_i)$ отождествляется с ближайшим меньшим (или большим) уровнем квантования.

Так как в процессе преобразования значение сигнала $x(t)$ отображается уровнем квантования x_m, а каждому уровню m может быть поставлен в соответствие свой номер (число), то при передаче или хранении информа-

ции можно вместо истинного значения величины x_m использовать соответствующее число m. Истинное значение уровня квантования легко восстановить, зная масштаб по шкале x. Для представления m уровней квантования с помощью неизбыточного равномерного кода потребуется $n = \log_2 m$ разрядов. Такое преобразование сопровождается шумами или погрешностью квантования. Погрешность квантования связана с заменой истинного значения сигнала $x(t_i)$ значением, соответствующим уровню квантования x_m. Максимальная погрешность квантования зависит от способа отождествления сигнала с уровнем квантования. Для первого из рассмотренных способов она равна $0{,}5\delta_m$, для второго — δ_m.

Чем меньше шаг квантования, тем меньше погрешность квантования. Можно принять, что погрешность квантования в пределах шага квантования имеет равновероятный закон распределения, т. е. любое значение функции в пределах шага будет равновероятным.

Наиболее часто используются степенные алгебраические полиномы вида $V(t) = \sum_{i=0}^{n} a_i t^i$, где n — степень полинома, a_i — действительные коэффициенты. Из этого класса функций наиболее полно исследовано применение полиномов нулевой и первой степени. Алгебраические полиномы удобны для программирования и обработки на ЭВМ.

Выбор оптимальной системы функции представляет определенные трудности, так как при решении задачи минимизации числа дискретных характеристик для описания сигнала с заданной точностью должны учитывать сложность аппаратуры (программ), допустимое время задержки в выдаче информации и другие факторы.

Метод дискретизации при преобразовании непрерывной информации в дискретную влияет на количество информации, которую надо хранить или преобразовывать в ЭВМ. Важна теорема Котельникова, согласно которой функция, имеющая ограниченный спектр частот, полностью определяется дискретным множеством своих значений, взятых с частотой отсчетов:

$$F_0 = 2f_m, \tag{1.9}$$

где $f_m = 2\pi\omega_m$ — максимальная частота в спектре частот $S(j\omega)$ сигнала $x(t)$; ω_m — угловая скорость.

Функция $x(t)$ воспроизводится без погрешностей по точным значениям $x(t_i)$ в виде ряда Котельникова:

$$x(t) = \sum_{k=-\infty}^{0} x(k_{\Delta t}) \frac{\sin \omega_m (t - k_{\Delta t})}{\omega_m (t - k_{\Delta t})}, \qquad (1.10)$$

где Δt — шаг дискретизации.

Теорема Котельникова справедлива для сигналов с ограниченным спектром. Реальные сигналы — носители информации — имеют конечную длительность. Спектр таких сигналов не ограничен, т. е. реальные сигналы не соответствуют в точности модели сигнала с ограниченным спектром, и применение теоремы Котельникова к реальным сигналам связано с погрешностями при восстановлении сигналов по формуле (1.10) и неопределенностью выбора шага дискретизации или частоты отсчетов.

Для практических задач, однако, идеально точное восстановление функций не требуется, необходимо лишь восстановление с заданной точностью. Поэтому теорему Котельникова можно рассматривать как приближенную для функций с неограниченным спектром. На практике частоту отсчетов часто определяют по формуле

$$F_0 = 2 f_{\max} k_3, \qquad (1.11)$$

где k_3 — коэффициент запаса (обычно $1{,}5 \le k_3 \le 6$); $f_{\max}$ — максимальная допустимая частота в спектре сигнала $x(t)$, например, с учетом доли полной энергии, сосредоточенной в ограниченном частотой спектре сигнала.

Из вышеизложенного следует, что преобразование непрерывной информации в дискретную может сопровождаться сжатием информации (уменьшением ее количества). Квантование по уровню — один из способов сжатия информации.

Квантование и дискретизация находят широкое применение в преобразователях информации, используемых при связи ЭВМ с конкретными объектами (процессами).

1.6. Формы представления информации

Информация всегда представляется в виде сообщения, которое передается некоторой физической средой. Носителем информации может быть любая предметная среда, которая меняет состояние в зависимости от передаваемой информации. Это может быть бумага, на которой информация изображается либо знаками, либо специальными отметками (например, перфорация); магнитный материал (лента, диск и т. п.), состояние которого

меняется с помощью магнита; электрический сигнал, у которого изменяется какой-либо параметр (частота, амплитуда).

Различают две формы представления информации: **статическую** I_c, (рис. 1.6, *а*) и **динамическую** $I_д$ (рис. 1.6, *б*). Возможность передачи сообщения посредством электрического сигнала реализуется с помощью канала связи, соединяющего источник и приемник информации (рис. 1.7). Чтобы передать информацию, необходимо ее предварительно преобразовать.

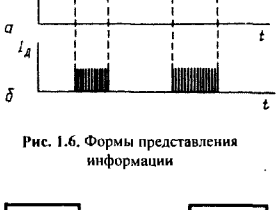


Рис. 1.6. Формы представления информации

Кодирование — преобразование сообщения в форму, удобную для передачи по данному каналу. В качестве простого примера можно привести передачу сообщения в виде телеграммы. Все символы кодируются с помощью телеграфного кода.

Декодирование — операция восстановления принятого сообщения. В систему связи необходимо ввести устройства для кодирования и декодирования информации (рис. 1.8). Теоретическое обоснование таких систем дал в своих работах К. Шеннон. Рядом теорем он показал эффективность введения кодирующих и декодирующих устройств, назначение которых состоит в согласовании свойств источника информации со свойствами канала связи. Одно из них (кодирующее устройство, или кодер) должно обеспечить такое кодирование, при котором путем устранения избыточности информации существенно снижается среднее число символов, приходящееся на единицу сообщения. При отсутствии помех это непосредственно дает выигрыш во времени передачи или в объеме запоминающего устройства. Такое кодирование называют *эффективным* (или *оптимальным*), так как оно повышает эффективность системы. При наличии помех в канале передачи оно позволяет преобразовать входную информацию в последовательность символов, наилучшим образом отвечающую задачам дальнейшего преобразования. Другое кодирующее устройство (кодер канала) обеспечивает заданную достоверность при передаче или хранении информации путем введения дополнительно избыточности информации. Такое кодирование называют *избыточным* или *помехоустойчивым*. Помехоустойчивость

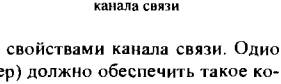


Рис. 1.7. Информационная модель канала связи

достигается учетом не только интенсивности помехи, но и ее статистических закономерностей.

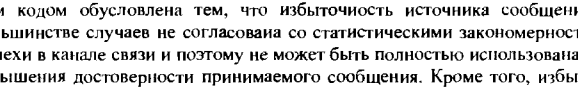


Рис. 1.8. Информационная модель канала связи с шумами

Целесообразность устранения избыточности сообщения методами эффективного кодирования с последующим перекодированием помехоустойчивым кодом обусловлена тем, что избыточность источника сообщения в большинстве случаев не согласована со статистическими закономерностями помехи в канале связи и поэтому не может быть полностью использована для повышения достоверности принимаемого сообщения. Кроме того, избыточность источника сообщений иногда является следствием ряда причин.

Если избыточность источника сообщений мала и помехи в канале связи практически отсутствуют, то введение как кодера источника, так и кодера канала нецелесообразно. Если избыточность источника сообщений высока, а помехи весьма малы, то целесообразно ввести кодер источника. Если избыточность источника мала, а помехи велики, то целесообразно ввести кодер канала. При большой избыточности и высоком уровне помех целесообразно ввести оба дополнительных кодирующих (и декодирующих) устройства. Большинство кодов, используемых при кодировании информации без учета статистических свойств источника и помехи в канале связи, основано на системах счисления.

1.7. Передача информации

Современные вычислительные средства (персональные ЭВМ, микро-, мини- и макси-ЭВМ) часто используются в составе вычислительных систем или сетей. В этих случаях необходимо решать вопросы не только эффективного представления информации, но также вопросы передачи информации по каналам связи без искажений. В качестве каналов связи (рис. 1.9) могут использоваться: непосредственная связь *НС* пользователя вычислительными средствами, телефонный канал *ТлК*, телеграфный канал *ТгК*, радиоканал *РК*, телевизионный канал *ТвК*, другие виды связи.

Вид канала определяет характер и величину помех, которые при этом появляются. Поэтому инженер-проектировщик ЭВМ должен учитывать эти обстоятельства при разработке технических, математических и программных средств.

Рассмотрим некоторые общие вопросы, возникающие при передаче информации.

Передача информации по каналу без помех. Если через канал связи без помех передается последовательность дискретных сообщений длительностью T, то скорость передачи информации по каналу связи (бит/с)

$$v = \lim_{I \to \infty} (I/T), \qquad (1.12)$$

где I — количество информации, содержащейся в последовательности сообщений.

Предельное значение скорости передачи информации называется пропускной способностью канала связи без помех $c = v_{\max}$.

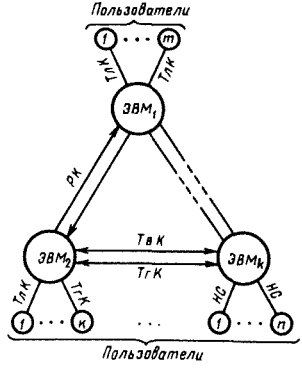


Рис. 1.9. Каналы связи в вычислительных сетях

Количество информации в сообщениях максимально при равной вероятности состояний. Тогда

$$c = \lim_{T \to \infty} \log_2 k/T . \qquad (1.13)$$

Скорость передачи информации в общем случае зависит от статистических свойств сообщений и параметров канала связи.

Пропускная способность — характеристика канала связи, которая не зависит от скорости передачи информации. Количественно пропускная способность канала связи выражается максимальным количеством двоичных единиц информации, которое данный канал связи может передать за одну секунду.

Для наиболее эффективного использования канала связи необходимо, чтобы скорость передачи информации была как можно ближе к пропускной способности канала связи. Если скорость поступления информации на вход канала связи превышает пропускную способность канала, то по каналу будет передана не вся информация. Основное условие согласования источника информации и канала связи

$$v \le c . \qquad (1.14)$$

Согласование осуществляется путем соответствующего кодирования сообщений. Доказано, что если скорость информации, вырабатываемой источ-

ником сообщений $v_и$, достаточно близка к пропускной способности канала c, т. е. $v_и = c - \varepsilon$, где ε — сколь угодно малая величина, то всегда можно найти такой способ кодирования, который обеспечит передачу сообщений, вырабатываемых источником, причем скорость передачи информации будет весьма близка к пропускной способности канала.

Верно и обратное утверждение: невозможно обеспечить длительную передачу всех сообщений, если поток информации, вырабатываемый источником, превышает пропускную способность канала.

Если ко входу канала подключен источник сообщений с энтропией, равной пропускной способности канала связи, то считается, что источник согласован с каналом. Если энтропия источника меньше пропускной способности канала, что может быть в случае неравновероятности состояний источника, то источник не согласован с каналом, т. е. канал используется не полностью.

Согласование в статистическом смысле достигается с помощью статистического кодирования. Оно позволяет повысить энтропию передаваемых сообщений до величины, которая получается, если символы новой последовательности равновероятны. При этом число символов в последовательности будет сокращено. В результате источник информации согласуется с каналом связи.

Передача информации по каналу с помехами. При передаче информации через канал с помехами сообщения искажаются, и на приемной стороне нет уверенности в том, что принято именно то сообщение, которое передавалось. Следовательно, сообщение недостоверно, вероятность правильности его после приема не равна единице. В этом случае количество получаемой информации уменьшается на величину неопределенности, вносимой помехами, т. е. вычисляется как разность энтропии сообщения до и после приема: $I' = H(i) - H_1(i)$, где $H(i)$ — энтропия источника сообщений; $H_1(i)$ — энтропия сообщений на приемной стороне.

Таким образом, скорость передачи по каналу связи с помехами

$$v' = \lim_{T \to \infty} \frac{H(i) - H_1(i)}{T}. \qquad (1.15)$$

Пропускной способностью канала с шумами называется максимальная скорость передачи информации при условии, что канал связи без помех согласован с источником информации:

$$c = \lim_{T \to \infty} \frac{I_{max}}{T}.$$

Если энтропия источника информации не превышает пропускной способности канала ($H \le c$), то существует код, обеспечивающий передачу информации через канал с помехами со сколь угодно малой частотой ошибок или сколь угодно малой недостовериостью. Пропускная способность канала связи при ограниченной средней мощности аналогового сигнала

$$c = F_m \log_2(1 + W_c / W_ш), \qquad (1.16)$$

где F_m — полоса частот канала (Гц); W_c — средняя мощность сигнала; $W_ш$ — средняя мощность помех (равномерный спектр) с нормальным законом распределения амплитуд в полосе частот канала связи.

Следовательно, можно передавать информацию по каналу с помехами без ошибок, если скорость передачи информации меньше пропускной способности канала, определяемой формулой (1.16). Для скорости $v > c$ при любой системе кодирования частота ошибок принимает конечное значение, причем оно растет с увеличением значения v. Из выражения (1.16) следует, что для канала с весьма высоким уровием шумов ($W_ш >> W_c$) максимальная скорость передачи близка к нулю.

Задания для самоконтроля

1. Являются ли словами следующие комбинации:

а) корабль, б) аааа, в) 12345; г) 1а25АВС

2. Что такое «бит»?

3. Какое количество информации можно изобразить с помощью двадцатиразрядных десятичных чисел?

4. В каких случаях статистическая мера информации совпадает с аддитивной мерой?

5. Определите пропускную способность канала связи, в котором отношение полезного сигнала к сигналу помехи составляет:

а) 1,2, б) 0,2

Полоса частот канала 1 МГц

2. АВТОМАТ КАК ОСНОВНОЙ ЭЛЕМЕНТ ИНФОРМАЦИОННЫХ СИСТЕМ

2.1. ЭВМ как автомат

Здесь полезно вспомнить слова В. М. Глушкова о том, что «электронные цифровые машины с программным управлением представляют собой пример одного из наиболее распространенных в настоящее время типов преобразователей дискретной информации, называемых дискретными или цифровыми автоматами» [6]. С точки зрения пользователя вычислительная машина представляется в виде некоего «черного ящика», выполняющего достаточно сложные и многочисленные операции при решении самых разнообразных задач. О том, как устроен этот «черный ящик», можно получить достаточно подробные сведения, проведя тщательный анализ процессов представления, преобразования и переработки информации. Необходимо выделить несколько важных положений.

1. Прежде всего, **любая вычислительная машина (ВМ) работает автоматически** (будь то большая или малая ЭВМ, персональный компьютер или Супер-ЭВМ). В этом смысле ВМ как автомат может быть описана структурной схемой, представленной на рис. 2.1. Элементы этой структурной схемы применительно к электроиным вычислительным машинам (ЭВМ) универсального назначения могут быть определены следующим образом (рис. 2.2).

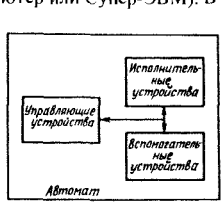


Рис. 2.1. Обобщенная блок-схема автомата

В качестве исполнительных элементов в автомат включаются:

— арифметико-логическое устройство (АЛУ);

— память;

— устройства ввода—вывода информации.

Управляющим элементом автомата является устройство управления (УУ), которое собственно обеспечивает автоматический режим работы.

Вспомогатсльными устройствами автомата могут быть всевозможные дополнительные средства, улучшающие или расширяющие возможности автомата.

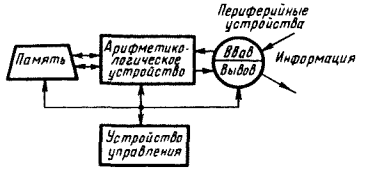


Рис. 2.2. Структурная схема ЭВМ

2. ЭВМ — программно-управляемый цифровой автомат.

Это положение подкрепляет две фундаментальные идеи. Первая состоит в том, что ЭВМ — автомат для переработки и преобразования *цифровой* или дискретной информации. Это означает, что вся подаваемая на вход ЭВМ информация (текстовая, графическая, числовая и т. п.) должна быть преобразована в набор цифр или чисел, представленных в выбранной *системе счисления*. Следовательно, выбор системы счисления является очень ответственной задачей для разработчика. Вторая идея заключается в том что ЭВМ управляется специальной программой, которая может либо вводиться в ЭВМ, либо храниться в ее памяти. Следует подчеркнуть очень важные функции памяти ЭВМ.

Память (запоминающее устройство) — функциональная часть ЭВМ предназначенная для хранения и (или) выдачи входной информации, промежуточных и окончательных результатов, вспомогательной информации. В памяти машины находятся также программы решения задач, через команды которых осуществляется управление работой всей машины.

Основные параметры, характеризующие память, — емкость и время обращения к памяти.

Емкость памяти — количество слов информации, которое можно записать в памяти. При этом *словом* является упорядоченная последовательность символов алфавита конечной длины. Ячейка памяти — часть памяти, содержащая слово.

Емкость памяти можно выразить количеством содержащихся в ней слов или ячеек. Длина ячейки памяти измеряется количеством битов (один бит равен одному двоичному разряду) или байтов (один байт содержит восемь битов). Ячейка памяти может вмещать информацию разной длины или разного *формата*. Формат измеряется словом, двойным словом или полу

словом в зависимости от принятого для данной ЭВМ способа представления информации.

Время обращения — интервал времени между началом и окончанием ввода (вывода) информации в память (из памяти). Оно характеризует затраты времени на поиск места и запись (чтение) слова в память (из памяти).

Для построения запоминающих устройств в качестве физических элементов используют электроиные схемы, ферритовые магнитные материалы, магнитные ленты и диски, барабаны с магнитным покрытием, оптические запоминающие элементы и т. д.

Основным преобразователем цифровой информации является арифметико-логическое устройство.

Арифметико-логическое устройство (АЛУ) — функциональная часть ЭВМ, которая выполняет логические и арифметические действия, необходимые для переработки информации, хранящейся в памяти. Оно характеризуется временем выполнения элементарных операций; средним *быстродействием*, т. е. количеством арифметических или логических действий (операций), выполняемых в единицу времени (секунду); набором элементарных действий, которые оно выполняет. Важной характеристикой АЛУ является также *система счисления*, в которой осуществляются все действия.

В современных вычислительных устройствах основным исполнительным элементом является процессор (П) или микропроцессор (МП), который содержит в себе АЛУ, память (как правило, оперативную память), блок управления.

В микропроцессорах важную роль играют шины данных и адресные шины или адресные магистрали. Структурная схема микропроцессора представлена на рис. 2.3. Числа в скобках указывают разрядность шин и устройств.

Вычислительные машины, построенные на основе микропроцессора, называются микроЭВМ [18] и отличаются тем, что обычно имеют два вида памяти: RAM (Random-Access-Memory) — память с произвольной выборкой (ППВ) и ROM (Read-Only-Memory) — постоянная память (ПП) на интегральных схемах. В постоянную память можно вложить уже готовый транслятор с алгоритмического языка или готовый пакет программ, выполняющий определенную функцию. Это позволяет расширить возможности микроЭВМ путем изготовления модулей расширения в виде ROM. Структурная схема микроЭВМ представлена на рис. 2.4.

Наличие входного и выходного каналов, а также средств и методов взаимодействия (интерфейса) ЭВМ с внешними устройствами позволяет существенно повысить скорость работы всего комплекса от ввода информации в машину до вывода ее. Фактически для осуществления подобного

принципа работы необходимо иметь несколько ЭВМ, выполняющих разные функции: управление работой всего комплекса устройств, выполнение арифметических и логических действий, ввод и вывод информации. Все это свидетельствует о существенном усложнении структуры ЭВМ, и эта тенденция сохраняется для персоиальных ЭВМ, к которым уже в полной мере можно применять термин «вычислительные системы» (рис. 2.5).

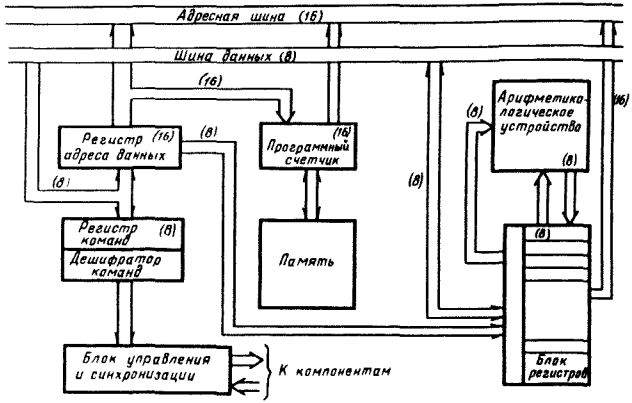


Рис. 2.3. Структурная схема микропроцессора

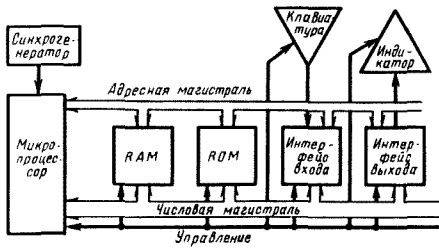


Рис. 2.4. Структурная схема микроЭВМ

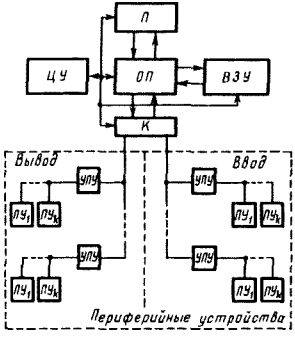


Рис. 2.5. Структурная схема вычислительной системы

2.2. Абстрактные автоматы и понятие алгоритма

Классические примеры абстрактных автоматов — машины Тьюринга или машины Поста*. Как правило, такая машина содержит бесконечную ленту, разделенную на отдельные секции (ячейки), в которые можно либо заносить метку, либо считывать метку с помощью записывающей или считывающей головки (рис. 2.6). Лента (или головка) может передвигаться в левую или правую стороны на один шаг в зависимости от команды. Лента всегда останавливается так, чтобы напротив головки находилась секция (ячейка). Команды абстрактного автомата обычно включают в себя одно из следующих действий (на примере машины Поста): движение головки вправо, движение головки влево, запись метки, стирание метки, передача управления, остановка (стоп).

Рис. 2.6. Структурная схема автомата Поста

Каждая команда имеет свой номер i. Стрелка указывает направление движения. Второе число j, стоящее в конце команды, называется

* Английский математик А. М. Тьюринг в работе «О вычислимых числах с приложением к проблеме разрешения» и американский математик Э. Х. Пост в работе «Финитные комбинаторные процессы» почти одновременно в 1936 г. дали уточнения понятия «алгоритм» на примере гипотетической машины с бесконечной лентой. Машина Тьюринга отличается от машины Поста тем, что ячейки заполняются не просто меткой, а символами из заданного множества.

отсылкой. У команды передачи управления могут быть две отсылки. Поэтому программа абстрактного автомата должна обладать двумя свойствами:

1) на первом месте в списке всегда стоит команда с номером 1, на втором месте — с номером 2 и т. д.;

2) отсылка любой из команд всегда находится в списке команд программы.

После передвижения ленты влево или вправо головка считывает состояние секции (пустая или записана метка). Информация о том, какие секции пусты, а какие отмечены, образует *состояние ленты* или *состояние автомата.* Таким образом, обладая указанным выше набором команд, автомат может осуществлять определенные действия, которые будут задаваться *программой.* Программой абстрактного автомата будем называть *конечный непустой список* команд.

Для «работы» абстрактного автомата необходимо задать программу и начальное состояние, т. е. положение головки и состояние ячеек ленты. После этого автомат приступает к выполнению команды номер 1. Все секции (ячейки) ленты нумеруются в определенном порядке. Порядок нумерации ячеек может совпадать с порядком, в котором расположены натуральные целые числа.

Каждая команда выполняется за один шаг, после чего начинается выполнение команды, номер которой указан в отсылке. Если эта команда имеет две отсылки, то команда с номером верхней отсылки выполняется, если под головкой находится пустая ячейка. Если же под головкой находится ячейка с меткой, то выполняется команда с номером нижней отсылки. Выполнение команды передачи управления не изменяет состояния автомата (ни одна из меток не уничтожается и не ставится, и лента остается неподвижной). При запуске автомата может возникнуть одна из следующих ситуаций:

автомат дошел до выполнения невыполнимой команды (запись метки в занятую ячейку, стирание метки в пустой ячейке); выполнение программы прекращается, автомат останавливается (назовем это состояние поломкой автомата), происходит безрезультатная остановка;

автомат дошел до команды стоп, программа считается выполненной, происходит результатная остановка;

автомат не доходит ни до результатной, ни до безрезультатной остановки, происходит бесконечная работа (автомат «завис»).

Рассмотрим работу автомата, начальное состояние которого задано рис. 2.7 при выполнении следующей программы:

1. → 3;
2. → 4;
3. ▽ 2;
4. ?<2;5;
5. стоп.

Если начальное состояние соответствует рис. 2.7, *а*, то выполнение программы приводит к результатной остановке. Если же исходное состояние автомата соответствует рис. 2.7, *б*, то программа не дает результата, автомат «зависает». Таким образом, начальное состояние автомата влияет на результативность его работы.

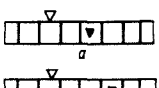


Рис. 2.7. Начальное состояние автомата

С другой стороны, различные программы, примененные к одному и тому же начальному состоянию, дают разный результат.

На таких элементарных автоматах, как машина Поста или машина Тьюринга, можно проводить различные действия над числами. Для этого необходимо представлять числа в абстрактном автомате.

Назовем последовательность секций (ячеек), содержащих метку, массивом, а число секций в нем — *длиной массива*. Условимся число n представлять на ленте массивом длины $n+1$. Тогда этот массив будем называть *автоматным изображением числа*. Например, числа 6, 3 и 2 представлены соответственно на рис. 2.8 автоматными изображениями.

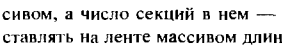


Рис. 2.8. Автоматное изображение чисел

Представим себе, что на машине Поста надо прибавить 1 к любому числу. Для этого требуется написать программу для машины Поста, обладающую следующим свойством: для любого числа n, записанного на ленте, программа должна дать результатную остановку с записью числа $n+1$ в произвольном месте ленты.

Программа может выглядеть следующим образом:

1-й вариант	2-й вариант
1. → 2;	1. ← 2;
2. ?<3;1;	2. ?<3;1;
3. ▽ 4;	3. ▽ 4;
4. стоп.	4. стоп.

В качестве начального состояния может быть выбрано любое состояние, при котором головка находится на одной из отмеченных ячеек ленты

(т. е. над набором числа). Если же головка будет находиться в произвольном месте ленты, то программа усложнится. Читателю предлагается самостоятельно написать такую программу.

С помощью абстрактного автомата можно реализовать и другие преобразования числовой информации. Рассмотрим, например, сложение двух чисел. В самой общей постановке эта задача формулируется так: составить программу сложения двух чисел n_1 и n_2 записанных на ленте на произвольном расстоянии друг от друга. Начальное состояние автомата показано на рис. 2.9.

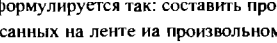


Рис. 2.9. Начальное состояние автомата для программы сложения чисел

Для написания программы можно, например, передвигать левый массив направо до слияния с правым массивом. Передвижение массива осуществляется перенесением (стиранием) самой левой метки в ближайшую пустую секцию справа (команды № 1 и № 7 программы, приведенной ниже). Когда массивы сольются (команды № 5 и № 6), то оказывается, что результат равен $n_1 + n_2 + 2$. Значит, надо стереть одиу лишнюю метку (команда № 4). В окончательном состоянии головка стоит левее образовавшейся суммы.

1. $\underset{\vee}{3}$ 2;
2. $\rightarrow$ 3;
3. $?\begin{matrix}\swarrow 12;\\ \searrow 4;\end{matrix}$
4. $\underset{\vee}{3}$ 5;
5. $\rightarrow$ 6;
6. $?\begin{matrix}\swarrow 7;\\ \searrow 5;\end{matrix}$
7. $\underset{\vee}{\vee}$ 8;
8. $\rightarrow$ 9;
9. $?\begin{matrix}\swarrow 10;\\ \searrow 12;\end{matrix}$
10. $\leftarrow$ 11;
11. $?\begin{matrix}\swarrow 2;\\ \searrow 10;\end{matrix}$
12. стоп.

Представленные примеры программ машины Поста не исчерпывают всех ее возможностей. Можно составить программы для умножения, деления чисел. Есть ли ограничения на вычисления, производимые на машине Поста? Ответ на этот вопрос был сформулирован самим Э. Постом в следующем виде: «*Задача на составление программы, приводящей от исходного данного к результирующему числу, тогда и только тогда имеет решение, когда имеется какой-нибудь общий способ, позволяющий по произвольному и одному данному выписать результирующее число*».

Формулировка постулата Поста подводит к понятию алгоритма*. Существует много определений термина «алгоритм». Например, по определению акад. А. Н. Колмогорова, *алгоритм или алгорифм — это всякая сис-*

* Термин «алгоритм» произошел от имени узбекского математика Аль-Хорезми, который еще в IX в. сформулировал правила выполнения четырёх арифметических действий. Появившееся несколько позже слово «алгорифм» связано с Евклидом, древнегреческим математиком, сформулировавшим правила нахождения наибольшего общего делителя двух чисел. В современной математике употребляют термин «алгоритм».

тема вычислений, выполняемых по строго определенным правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи.

В инженериой практике часто используется следующее определение: алгоритм — конечиая совокупность точно сформулированиых правил решения какой-то задачи [1].

По форме задания алгоритмы могут быть словесными и математическими. Пример *словесной формы* алгоритма — алгоритм Евклида для нахождения наибольшего общего делителя двух чисел a и b.

1. Обозревая два числа a и b, переходи к следующему пункту.

2. Сравни обозреваемые числа (a равно b, a меньше, больше b) и переходи к следующему пункту.

3. Если a и b равны, то прекрати вычислеиие: каждое из чисел дает искомый результат. Если числа не равны, то переходи к следующему пункту.

4. Если первое число меиьше второго, то переставь их местами и переходи к следующему пункту.

5. Вычти второе число из первого и сделай обозрение двух чисел: вычитаемого и остатка; переходи к п. 2.

По указаииям этого алгоритма можно найти наибольший общий делитель для любой пары целых чисел.

Характеристиками алгоритма являются:

— детерминированность, определяющая однозначность результата решения задачи при заданных исходных данных;

— дискретность определяемого алгоритмом процесса, означающая расчленеиность его на отдельные элементарные шаги;

— массовость, позволяющая применять один и тот же алгоритм для некоторого множества одиотипных задач.

Эти характеристики не дают точного описания алгоритма, а лишь объясняют смысл этого термиина в математике.

Пример алгебраической формы алгоритма — любая математическая формула для нахождения какой-то величины. Например, значение корней уравнеиия вида $ax^2 + bx + c = 0$ можно найти по формуле $x_{1,2} = (-b \pm \sqrt{(b^2 - 4ac)})/2a$, которая представляет собой алгоритм нахождения этих корней. Однако для того, чтобы реализовать математическую форму алгоритма, требуется дать еще ряд словесных указаиий, показать область примеиеиия алгоритма.

Детерминированный алгоритм — алгоритм, имеющий место при четкой и ясной системе правил и указаний и однозначных действиях.

Случайный алгоритм — алгоритм, предусматривающий возможность случайного выбора тех или иных правил.

Алгоритм должен обеспечивать получение результата через конечное число шагов для любой задачи определенного класса. В противном случае задача неразрешима. Нахождение алгоритма решения задачи называется *алгоритмизацией*.

Процесс выполнения алгоритма называется *алгоритмическим процессом*. Для некоторых исходных данных он заканчивается получением искомого результата после конечного числа шагов. Однако возможны случаи, когда искомый результат не достигается или безрезультатно обрывается. Тогда говорят, что к таким исходным данным алгоритм неприменим.

Таким образом, алгоритм дает возможность ответить на вопрос «что делать?» в каждый момент времени, однако создать алгоритм не всегда возможно.

Численный алгоритм — алгоритм, соответствующий решению поставленной задачи с помощью арифметических действий.

Логический алгоритм — алгоритм, используемый в случае, если при решении задачи приходится применять некоторые логические действия.

Процесс решения задачи на ЭВМ прежде всего должен быть выражен каким-то алгоритмом. Разработка алгоритмов решения задач — задача программиста, а разработка алгоритмов функционирования цифрового автомата для решения поставленных задач — задача инженера-разработчика.

2.3. Основные понятия алгебры логики

Понятие автомата было введено в гл. 1 в качестве модели для описания функционирования устройств, предназначенных для переработки дискретной информации.

Для формального описания цифрового автомата широко применяют аппарат алгебры логики, являющейся одним из важных разделов математической логики*.

Основное понятие алгебры логики — высказывание. *Высказывание* — некоторое предложение, о котором можно утверждать, что оно истинно или ложно. Например, высказывание «Земля — это планета Солнечной системы» истинно, а о высказывании «на улице идет дождь» можно ска-

* Создатель алгебры логики — английский математик Дж. Буль (1815–1864). Поэтому алгебру логики называют также алгеброй Буля. В последние годы алгебра Буля получила значительное развитие благодаря работам таких ученых, как Э. Пост, К. Шеннон, Г. Л. Шестаков, В. М. Глушков [6], С. В. Яблонский [21] и др.

зать, истинно оно или ложно, если указаны дополнительные сведения о погоде в данный момент.

Любое высказывание можно обозначить символом x и считать, что $x = 1$, если высказывание истинно, а $x = 0$ — если высказывание ложно.

Логическая (булева) переменная — такая величина x, которая может принимать только два значения: $x = \{0, 1\}$.

Высказывание абсолютно истинно, если соответствующая ей логическая величина принимает значение $x = 1$ при любых условиях. Пример абсолютно истинного высказывания — высказывание «Земля — планета Солнечной системы».

Высказывание абсолютно ложно, если соответствующая ей логическая величина принимает значение $x = 0$ при любых условиях.

Например, высказывание «Земля — спутник Марса» абсолютно ложно.

Логическая функция (функция алгебры логики) — функция $f(x_1, x_2, \ldots, x_n)$, принимающая значение, равное нулю или единице на наборе логических переменных $x_1, x_2, \ldots, x_n$.

Логические функции от одной переменной представлены в таблице 2.1.

Таблица 2.1

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	1	0	0	1
1	1	0	1	0

В соответствии с введенными определениями функция $f_1(x)$ является абсолютно истинной (константа единицы), а функция $f_2(x)$ — абсолютно ложной функцией (константа нуля).

Функция $f_3(x)$, повторяющая значения логической переменной, — *тождественная функция* $[f_3(x) \equiv x]$, а функция $f_4(x)$, принимающая значения, обратные значениям x, — *логическое отрицание*, или функция НЕ $[f_4(x) = \neg x = \bar{x}]$.

Логические функции от двух переменных представлены в таблице 2.2.

Дизъюнкция (логическое сложение) — функция $f_7(x_1, x_2)$, которая истинна тогда, когда истинны или x_1, или x_2, или обе переменные.

Дизъюнкцию часто называют также функцией ИЛИ и условно обозначают так: $f_7(x_1, x_2) = x_1 + x_2 = x_1 \vee x_2$.

Таблица 2.2

Функция	x_1x_2				Примечание
	00	01	10	11	
f_0	0	0	0	0	f_0 — абсолютная ложь
f_1	0	0	0	1	$x_1 \wedge x_2$ (конъюнкция)
f_2	0	0	1	0	$x_1 \wedge \bar{x}_2$ (запрет x_2)
f_3	0	0	1	1	$x_1\bar{x}_2 \vee x_1x_2 = x_1$ (переменная x_1)
f_4	0	1	0	0	$\bar{x}_1x_2$ (запрет x_1)
f_5	0	1	0	1	$\bar{x}_1x_2 \vee x_1x_2 = x_2$ (переменная x_2)
f_6	0	1	1	0	$x_1 \oplus x_2$ (сложение по модулю 2)
f_7	0	1	1	1	$x_1 \vee x_2$ (дизъюнкция)
f_8	1	0	0	0	$x_1 \downarrow x_2$ (фуикция Пирса)
f_9	1	0	0	1	$x_1 \equiv x_2$ (равнозначность)
f_{10}	1	0	1	0	$\bar{x}_1\bar{x}_2 \vee x_1\bar{x}_2 = \bar{x}_2$ (переменная $\bar{x}_2$)
f_{11}	1	0	1	1	$x_2 \rightarrow x_1$ (импликация)
f_{12}	1	1	0	0	$\bar{x}_1\bar{x}_2 \vee \bar{x}_1x_2 = \bar{x}_1$ (переменная $\bar{x}_1$)
f_{13}	1	1	0	1	$x_1 \rightarrow x_2$ (импликация)
f_{14}	1	1	1	0	x_1/x_2 (функция Шеффера)
f_{15}	1	1	1	1	f_1 — абсолютная истина

От дизъюнкции следует отличать функцию $f_6(x_1, x_2)$, которая называется *функцией сложения по модулю 2* (*функцией исключительное* ИЛИ) и является истинной, когда истинны или x_1, или x_2, в отдельности. Условное обозначение этой функции $f_6(x_1, x_2) = x_1 \oplus x_2$.

Конъюнкция (*логическое умножение*) — функция $f_1(x_1, x_2)$, которая истинна только тогда, когда и x_1, и x_2 истинны. Конъюнкцию часто называют также функцией И; условно обозначают так: $f_1(x_1, x_2) = x_1 \& x_2 = x_1 \wedge x_2$.

Пример 2.1. Имеются два высказывания: «Завтра будет холодная погода», «Завтра пойдет снег».

Дизъюнкция этих высказываний — новое высказывание: «Завтра будет холодная погода или пойдет снег» Соединительный союз, который образовал новое предложение, — ИЛИ.

Конъюнкция образуется следующим образом: «Завтра будет холодная погода и пойдет снег» Это высказывание образовано с помощью союза И.

Штрих Шеффера — функция $f_{14}(x_1, x_2)$, которая ложна только тогда, когда x_1 и x_2 истинны. Условное обозначение функции Шеффера:

$f_{14}(x_1, x_2) = x_1/x_2$. Немецкий математик Д. Шеффер на основе этой функции создал алгебру, названную алгеброй Шеффера.

Функция Пирса (Вебба) — функция $f_8(x_1, x_2)$, которая истинна только тогда, когда x_1 и x_2 ложны. Условное обозначение этой функции: $f_8(x_1, x_2) = x_1 \downarrow x_2 = x_1 \circ x_2$.

Математики Ч. Пирс и Д. Вебб, независимо друг от друга изучавшие свойства этой функции, создали алгебру, названную алгеброй Пирса (Вебба).

Импликация — функция $f_{13}(x_1, x_2)$, которая ложна тогда и только тогда, когда x_1 истинно и x_2 ложно. Условное обозначение: $f_{13}(x_1, x_2) = x_1 \to x_2$.

Все логические функции, приведенные в таблице 2.2, — элементарные функции.

Две функции равносильны друг другу, если принимают на всех возможных наборах переменных одни и те же значения:

$$f_1(x_1, x_2, \ldots, x_n) = f_2(x_1, x_2, \ldots, x_n).$$

Булевы переменные могут быть действительными или фиктивными. Переменная x_i *действительна*, если значение функции $f(x_1, \ldots, x_i, \ldots, x_n)$ изменяется при изменении x_i. Переменная x_i *фиктивна*, если значение функции $f(x_1, \ldots, x_i, \ldots, x_n)$ не изменяется при изменении x_i.

Пример логической функции от трех переменных представлен в таблице 2.3.

Из таблицы видно, что переменные x_1 и x_2 — действительные, а переменная x_3 — фиктивная, так как $f(x_1, x_2, 0) = f(x_1, x_2, 1)$ для всех наборов x_1, x_2.

Таблица 2.3

x_1	x_2	x_3	$f(x_1, x_2, x_3)$	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0	1	0	0	1
0	0	1	0	1	0	1	1
0	1	0	1	1	1	0	0
0	1	1	1	1	1	1	0

Использование фиктивных функций дает возможность сокращать или расширять количество переменных для логических функций.

Так как число значений переменных x_i ограничено, то можно определить количество функций N от любого числа переменных n: $N = 2^{2^n}$.

Рассмотрим некоторые практические примеры использования алгебры логики.

Пример 2.2. В школе произошла неприятная история: разбито окно в одном из классов. Подозревают четырех учеников: Леню, Диму, Толю и Мишу. При опросе каждый из детей сделал по три заявления:

Леня:

1) я не виноват — $Л_1$;

2) я не подходил к окну — $Л_2$;

3) Миша знает, кто разбил, — $Л_3$.

Дима:

1) стекло разбил не я — $Д_1$;

2) с Мишей я не был знаком до поступления в школу — $Д_2$;

3) это сделал Толя — $Д_3$.

Толя:

1) я не виноват — $Т_1$;

2) это сделал Миша — $Т_2$;

3) Дима говорит неправду, утверждая, что я разбил окно, — $Т_3$.

Миша:

1) я не виноват — $М_1$;

2) стекло разбил Леня — $М_2$;

3) Дима может поручиться за меня, так как знает меня со дня рождения, — $М_3$.

В дальнейшем все признали, что одно из трех заявлений является неверным. Это пригодится при построении более сложных формул, поскольку показания каждого ученика в целом истинны только при условии, что два заявления истинны, а одно ложно. Используя элементарные логические функции, можно описать показания всех учеников в таком виде:

$$Л = Л_1Л_2\overline{Л}_3 + Л_1\overline{Л}_2Л_3 + \overline{Л}_1Л_2Л_3;$$

$$Д = Д_1Д_2\overline{Д}_3 + Д_1\overline{Д}_2Д_3 + \overline{Д}_1Д_2Д_3;$$

$$Т = Т_1Т_2\overline{Т}_3 + Т_1\overline{Т}_2Т_3 + \overline{Т}_1Т_2Т_3;$$

$$М = М_1М_2\overline{М}_3 + М_1\overline{М}_2М_3 + \overline{М}_1М_2М_3.$$

Теперь остается решить эту систему уравнений и определить, какие показания истинны. Для этого надо упростить выражения, используя аксиомы.

Рассмотрим третье уравнение. По условию, $Т_1 = Т_3$, а значит, $\overline{Т}_1 = \overline{Т}_3$, но $Т_1\overline{Т}_1 = 0$, $Т_1Т_1 = Т_1$ или $Т = Т_1\overline{Т}_2$. Поэтому оно верно тогда, когда $Т_1 = 1$; $Т_2 = 0$. Значит, Толя не виноват и Миша не виноват. Отсюда следует, что $Д_3$ ложно, т. е. $Д_3 = 0$ ($\overline{Д}_3 = 1$). Следовательно, $Д = Д_1Д_2\overline{Д}_3$. Отсюда $Д_1 = 1$; $Д_2 = 1$. Дима не виноват.

$Д_2$ противоположно $М_3$, т. е. $\overline{Д}_2 = М_3$. Значит, $М_3 = 0$ или $М = М_1М_2\overline{М}_3$. Оно верно только тогда, когда $М_1 = 1$; $М_2 = 1$.

Ответ стекло разбил Леня.

Пример 2.3. Предположим, что имеется система кондиционирования воздуха для помещения, где установлена ЭВМ, состоящая из двух кондиционеров малой и большой мощности и работающая при таких условиях: кондиционер малой мощности включается, если температура воздуха в помещении достигает 19° С ; кондиционер большой мощности включается, если температура воздуха достигает 22° С (малый кондиционер при этом отключается), оба кондиционера включаются при температуре воздуха 30° С

Пусть информация о температуре воздуха поступает от датчиков, которые срабатывают при достижении температуры соответственно 19, 22, 30° С. Каждый из этих датчиков выдает входную информацию для устройства управления кондиционерами. Первые три датчика определяют рабочие режимы, и их можно представить как входы управляющего автомата. Используя двоичный алфавит для задания состояний датчика (0 — нет сигнала о достижении заданного уровня температуры, 1 — есть сигнал), функционирование системы управления кондиционерами можно описать следующим образом

z_3	z_2	z_1	ω_2	ω_1
0	0	0	0	0
0	0	1	0	1
0	1	1	1	0
1	1	1	1	1

Здесь z_1 — сигнал датчика, срабатывающего при $t = 19°$ С ; $z_1 = 0$, если температура меньше 19° С ; $z_1 = 1$, если температура равна или больше 19° С ; z_2 — датчик, срабатывающий при $t = 22°$ С , $z_2 = 0$, если $t < 22°$ С , $z_2 = 1$ при $t \geq 22°$ С ; z_3 — датчик, срабатывающий при $t = 30°$ С ; $z_3 = 0$ при $t < 30°$ С , $z_3 = 1$ при $t \geq 30°$ С ; ω_1 и ω_2 — соответственно сигналы управления маломощным и мощным кондиционерами ($\omega = 0$ — кондиционер выключен, $\omega = 1$ — кондиционер включен).

Таблица описывает функционирование системы управления без нарушений работы.

Впервые теория Дж. Буля была применена П. С. Эренфестом к анализу контактных цепей (1910 г.). Возможность описания переключательных схем с помощью логических формул оказалась весьма ценной по двум причинам. Во-первых, с помощью формул удобнее проверять работу схем. Во-вторых, задание условий работы любой переключательной схемы в виде формул упрощает процесс построения самой переключательной схемы, так как оказалось, что существует ряд эквивалентных преобразований, в результате которых логические формулы упрощаются. При описании переключательных схем замкнутое состояние контакта принимается за истинное высказывание, а разомкнутое — за ложное, поэтому последовательное соединение контактов можно рассматривать как функцию И, а параллельное — как функцию ИЛИ.

Использование логических функций оказалось особенно полезным для описания работы логических элементов ЭВМ.

2.4. Свойства элементарных функций алгебры логики

Из таблицы 2.2 видно, что элементарные функции типа отрицания, дизъюнкции, Шеффера, Пирса, импликации и т. д. находятся в определенной связи друг с другом. Рассмотрим эти связи и свойства исходных функций.

Конъюнкция, дизъюнкция, отрицание (функции И, ИЛИ, НЕ). Используя основные положения алгебры логики, нетрудно убедиться в справедливости следующих восьми аксиом. Пусть x — некоторая логическая переменная. Тогда

1. $\bar{\bar{x}} = x$, что означает возможность исключения из логического выражения всех членов, имеющих двойное отрицание, заменив их исходной величиной;

2. $x + x = x$; $xx = x$ — правила подобных преобразований позволяют сокращать длину логических выражений;

3. $x + 0 = x$;

4. $x + 1 = 1$;

5. $x0 = 0$;

6. $x \cdot 1 = x$;

7. $x\bar{x} = 0$;

8. $x + \bar{x} = 1$.

Дизъюнкция и конъюнкция обладают рядом свойств, аналогичных свойствам обычных арифметических операций сложения и умножения:

1) свойство ассоциативности (сочетательный закон):

$$x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3, \quad x_1(x_2x_3) = (x_1x_2)x_3;$$

2) свойство коммутативности (переместительный закон):

$$x_1 + x_2 = x_2 + x_1, \quad x_1x_2 = x_2x_1;$$

3) свойство дистрибутивности (распределительный закон):

для конъюнкции относительно дизъюнкции

$$x_1 \& (x_2 + x_3) = (x_1 \& x_2) + (x_1 \& x_3);$$

для дизъюнкции относительно конъюнкции

$$x_1 + x_2x_3 = (x_1 + x_2) \& (x_1 + x_3).$$

Свойство дистрибутивности фактически определяет правила раскрытия скобок или взятия в скобки логических выражений.

Справедливость указанных свойств легко доказывается с помощью вышеизложенных аксиом.

Докажем, например, что

$$x_1 + x_2 x_3 = (x_1 + x_2) \& (x_1 + x_3).$$

В самом деле, $(x_1 + x_2)(x_1 + x_3) = x_1 x_1 + x_1 x_3 + x_1 x_2 + x_2 x_3 = x_1 + x_1 x_2 + x_1 x_3 + x_2 x_3 = x_1(1 + x_2 + x_3) + x_2 x_3 = x_1 + x_2 x_3$.

Аналогично можно доказать и другие законы.

Несложно установить правильность соотношений, известных как *законы де Моргана:*

$$\begin{aligned} \overline{x_1 x_2} &= \bar{x}_1 + \bar{x}_2, \\ \overline{x_1 + x_2} &= \bar{x}_1 \bar{x}_2. \end{aligned} \tag{2.1}$$

Из законов де Моргана вытекают следствия:

$$\begin{aligned} x_1 x_2 &= \overline{\bar{x}_1 + \bar{x}_2}, \\ x_1 + x_2 &= \overline{\bar{x}_1 \bar{x}_2}, \end{aligned} \tag{2.2}$$

с помощью которых появляется возможность выражать конъюнкцию через дизъюнкцию и отрицание или дизъюнкцию через конъюнкцию и отрицание.

Законы де Моргана и следствия из них справедливы для любого количества переменных:

$$\begin{aligned} \overline{x_1 + x_2 + \ldots + x_n} &= \bar{x}_1 \& \ldots \& \bar{x}_n, \\ \overline{x_1 x_2 \ldots x_n} &= \bar{x}_1 + \bar{x}_2 + \ldots + \bar{x}_n. \end{aligned} \tag{2.3}$$

Для логических функций устанавливаются соотношения, известные как *законы поглощения*:

$$\begin{aligned} x_1 + x_1 x_2 &= x_1, \\ x_1(x_1 + x_2) &= x_1. \end{aligned} \tag{2.4}$$

В таблице 2.4 показана справедливость законов поглощения.

Функция сложения по модулю 2 (исключительное ИЛИ) — функция, выражаемая следующим образом:

$$x_1 \oplus x_2 = x_1 \bar{x}_2 + \bar{x}_1 x_2 = (x_1 + x_2)(\bar{x}_1 + \bar{x}_2). \tag{2.5}$$

Таблица 2.4

x_1	x_2	$x_1 + x_2$	x_1x_2	$x_1 + x_1x_2$	$x_1(x_1 + x_2)$
0	0	0	0	0	0
0	1	1	0	0	0
1	0	1	0	1	1
1	1	1	1	1	1

Функция сложения по модулю 2 обладает следующими свойствами: коммутативности (переместительный закон):

$$x_1 \oplus x_2 = x_2 \oplus x_1;$$

ассоциативности (сочетательный закон):

$$x_1 \oplus (x_2 \oplus x_1) = (x_1 \oplus x_2) \oplus x_3;$$

дистрибутивности (распределительный закон):

$$x_1(x_2 \oplus x_3) = (x_1x_2) \oplus (x_1x_3).$$

Для этой функции справедливы аксиомы:

$$x \oplus x = 0; \quad x \oplus 1 = \bar{x};$$
$$x \oplus \bar{x} = 1; \quad x \oplus 0 = x.$$

На основании аксиом и свойств можно вывести правила перевода функций И, ИЛИ, НЕ через функцию сложения по модулю 2 и наоборот:

$$\bar{x}_1 = x_1 \oplus 1;$$
$$x_1 + x_2 = x_1 \oplus x_2 \oplus x_1x_2; \qquad (2.6)$$
$$x_1x_2 = (x_1 \oplus x_2) \oplus (x_1 + x_2).$$

Функция импликации ($\rightarrow$) — функция, выражаемая следующим образом: $x_1 \rightarrow x_2 = x_1 + x_2$.

Для функции импликации справедливы аксиомы:

$$x \rightarrow x = 1; \quad x \rightarrow 1 = 1; \quad 0 \rightarrow x = 1;$$
$$x \rightarrow \bar{x} = \bar{x}; \quad x \rightarrow 0 = \bar{x}; \quad 1 \rightarrow \bar{x} = x.$$

Из аксиом следует, что импликация обладает только свойством коммутативности (переместительный закон) в измененном виде: $x_1 \rightarrow x_2 = \bar{x}_2 \rightarrow \bar{x}_1$.

Свойство ассоциативности для этой функции несправедливо (таблица 2.5).

Таблица 2.5

x_1	x_2	x_3	$x_1 \to (x_2 \to x_3)$	$(x_1 \to x_2) \to x_3$
0	0	0	1	0
0	1	0	1	0
1	0	0	1	1
1	1	0	0	0
0	0	1	1	1
0	1	1	1	1
1	0	1	1	1
1	1	1	1	1

Функции И, ИЛИ, НЕ через импликацию выражаются так:

$$x_1 + x_2 = \bar{x}_1 \to x_2;$$
$$x_1 x_2 = \overline{x_1 \to \bar{x}_2}; \qquad (2.7)$$
$$\bar{x}_1 = x_1 \to 0.$$

Функция Шеффера (/) — функция, которая может быть выражена соотношением $x_1/x_2 = \overline{x_1 x_2}$.

Для нее характерны аксиомы:

$$x/x = \bar{x}; \quad x/\bar{x} = 1; \quad x/0 = 1;$$
$$x/1 = \bar{x}; \quad \bar{x}/0 = 1; \quad \bar{x}/1 = x,$$

что подтверждается таблицей 2.6.

Таблица 2.6

x	$\bar{x}$	x/x	$x/\bar{x}$	$x/0$	$x/1$	$\bar{x}/0$	$\bar{x}/1$
1	0	0	1	1	0	1	1
0	1	1	1	1	1	1	0

Эти аксиомы позволяют сформулировать следующие правила преобразования:

$$x_1 x_2 = \overline{x_1/x_2} = (x_1/x_2)/(x_1/x_2);$$
$$\bar{x} = x/x; \qquad (2.8)$$
$$x_1 + x_2 = \bar{x}_1/\bar{x}_2 = (x_1/x_1)(x_2/x_2).$$

Для фуикции Шеффера справедливо свойство коммутативности для двух переменных $x_1/x_2 = x_2/x_1$, что легко проверяется. Для трех и более переменных это свойство уже не выполняется.

В самом деле, функция Шеффера является строго *двуместной функцией*, т. е. для нее невозможны записи вида $x_1/x_2/x_3$ или $x_1/x_2/\ldots/x_n$, так как непонятно, в какой очередности применять операцию Шеффера в этом

выражении. Следовательно, очередиость применения операции Шеффера должна указываться с помощью скобок, как, иапример,

$$((x_1/x_2)/x_3)/x_4 \text{ или } (x_1/x_2)/(x_3/x_4).$$

Эти выводы подтверждаются также тем, что свойства ассоциативиости и дистрибутивности для функцни Шеффера несправедливы. В самом деле, справедливость свойства ассоциативности ведет к тому, что должны быть равны выражения $(x_1/x_2)/x_3$ и $x_1/(x_2/x_3)$.

Применяя правила преобразования к этим выражениям, получаем

$$(x_1/x_2)/x_3 = \overline{(x_1x_2)}\,x_3 = x_1x_2 + \bar{x}_3,$$
$$x_1/(x_2/x_3) = \bar{x}_1 + x_2 + x_3.$$

Проверка равнозначности этих выражений представлена в таблице 2.7.

Таблица 2.7

x_1	x_2	x_3	x_1x_2	x_2x_3	$x_1x_2 + \bar{x}_3$	$\bar{x}_1 + x_2x_3$
0	0	0	0	0	1	1
0	0	1	0	0	0	1
0	1	0	0	0	1	1
0	1	1	0	1	0	1
1	0	0	0	0	1	0
1	0	1	0	0	0	0
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Следовательно, функции $(x_1/x_2)/x_3$ и $x_1/(x_2x_3)$ не равны, значнт, функция Шеффера не обладает свойством ассоциативности в случае трех переменных, что можно распространить и на «n» переменных.

Для функции Шеффера свойство дистрибутивности означает, что должны быть равны две функции: $x_1/(x_2x_3)$ и $(x_1/x_2)/(x_1/x_3)$. Преобразуем эти функции, используя аксиому $\bar{x} = x/1$:

$$x_1/(x_2x_3) = \bar{x}_1 + x_2x_3 = \overline{(\bar{x}_1 + x_2)(\bar{x}_1 + x_3)} = (x_1/\bar{x}_2)(x_1/\bar{x}_3) =$$
$$= \overline{(x_1/(x_2/1))(x_1/(x_3/1))} = (x_1/(x_2/1))/(x_1/(x_3/1))/1;$$
$$(x_1/x_2)(x_1/x_3) = \overline{(x_1x_2)(x_1x_3)} = x_1x_2 + x_1x_3 = x_1(x_2 + x_3) = x_1\overline{(\bar{x}_2\bar{x}_3)} =$$
$$= \overline{x_1((x_2/1)(x_3/1))} = \overline{x_1/((x_2/1)/(x_3/1))} = (x_1/((x_2/1)/(x_3/1)))/1.$$

Все этапы этих преобразований показывают, что $x_1/(x_2/x_3) \neq (x_1/x_2)/(x_1/x_3)$, т. е. свойство дистрибутивности несправедливо для

функции Шеффера от трех переменных, а следовательно, несправедливо и для «*n*» переменных.

Необходимо указать на возможность возникновения следующего заблуждения: функция Шеффера равносильна функции отрицания конъюнкции. Так как конъюнкция обладает всеми свойствами (коммутативностью, ассоциативностью, дистрибутивностью), то по этой причине и функция Шеффера должна была бы иметь эти же свойства. Однако равносильность функций на всех наборах не обязательно совпадает с наличием одинаковых свойств, что и подтверждает пример функций Шеффера и НЕ—И.

Функция Пирса (Вебба) ($\downarrow$) — функция, которая описывается выражением $x_1 \downarrow x_2 = \overline{x_1 + x_2} = \bar{x}_1 \bar{x}_2$.

Для функций Пирса (Вебба) справедливы аксиомы:

$$x \downarrow x = \bar{x}; \quad x \downarrow 0 = \bar{x}; \quad x \downarrow 1 = 0;$$
$$x \downarrow \bar{x} = 0; \quad \bar{x} \downarrow 0 = x; \quad \bar{x} \downarrow 1 = 0,$$

что подтверждается таблицей 2.8.

Таблица 2.8

x	$\bar{x}$	$x \downarrow x$	$x \downarrow \bar{x}$	$x \downarrow 0$	$x \downarrow 1$	$\bar{x} \downarrow 0$	$\bar{x} \downarrow 1$
1	0	0	0	0	0	1	0
0	1	1	0	1	0	0	0

Элементарные булевы функции И, ИЛИ, НЕ выражаются через функцию Пирса (Вебба) следующим образом:

$$\begin{aligned} \bar{x} &= x \downarrow x; \\ x_1 x_2 &= (x_1 \downarrow x_1) \downarrow (x_2 \downarrow x_2); \\ x_1 + x_2 &= (x_1 \downarrow x_2) \downarrow (x_1 \downarrow x_2). \end{aligned} \tag{2.9}$$

Для функции Пирса (Вебба) справедливо свойство коммутативности только для двух переменных: $x_1 \downarrow x_2 = x_2 \downarrow x_1$. Функция Пирса (Вебба), так же как и функция Шеффера, является двуместной функцией. Следовательно, невозможны записи вида: $x_1 \downarrow x_2 \downarrow x_3 \downarrow \ldots \downarrow x_n$; для установления приоритетов обязательно должны использоваться скобки, которые определяют последовательность осуществления операций:

$$(x_1 \downarrow x_2)(x_3 \downarrow (x_4 \downarrow x_5)).$$

Аналогично с функцией Шеффера для функции Пирса (Вебба) несправедливы свойства ассоциативности и дистрибутивности. Метод доказательства такой же, как и в предыдущем случае.

Надо заметить также, что $x_1 \downarrow x_2 = \overline{x_1 + x_2}$, т. е. функция Пирса равносильна функции НЕ—ИЛИ. Здесь также равносильность функций не ведет к наличию одинаковых свойств у этих функций.

2.5. Аналитическое представление функций алгебры логики

Существует много способов задания логических функций. Ранее был рассмотрен табличный способ, при котором каждому набору значений переменных в таблице истинности указывается значение самой логической функции. Этот способ нагляден и может быть применен для записи функций от любого количества переменных. Однако при анализе свойств функций алгебры логики (ФАЛ) такая запись не является компактной. Проще выглядит аналитическая запись в виде формул.

Рассмотрим фиксированный набор переменных $\{x_1, x_2, \ldots, x_n\}$, на котором задана функция алгебры логики. Так как любая переменная $x_i = \{0, 1\}$, то набор значений переменных фактически представляет собой некоторое двоичное число. Представим, что номером набора будет произвольное двоичное число i, получаемое следующим образом:

$$i = 2^{n-1}x_1 + 2^{n-2}x_2 + \ldots + x_n . \qquad (2.10)$$

Пусть имеется функция $\Phi_i(x_1, x_2, \ldots, x_n)$:

$$\Phi_i = \begin{cases} 0, \text{ если номер набора равен } i; \\ 1, \text{ если номер набора не равен } i. \end{cases}$$

Функцию Φ_i называют *термом*.

Дизъюнктивный терм (*макстерм*) — терм, связывающий все переменные, представленные в прямой или инверсной форме, знаком дизъюнкции (иногда в литературе используется термин «конституэнта нуля»).

Например, $\Phi_7 = \bar{x}_1 \vee x_2 \vee x_3 \vee x_4$; $\Phi_2 = x_1 \vee \bar{x}_2$.

Конъюнктивный терм (минтерм) — терм, связывающий переменные, представленные в прямой или инверсной форме, знаком конъюнкции (иногда в литературе используется термин «конституэнта единицы»). Обозначается минтерм следующим образом:

$$F_i = \begin{cases} 1, \text{ если номер набора равен } i, \\ 0, \text{ если номер набора не равен } i. \end{cases}$$

Например, $F_1 = x_1x_2x_3x_4$; $F_0 = \bar{x}_1\bar{x}_2\bar{x}_3$.

Ранг терма r определяется количеством переменных, входящих в данный терм. Например, для минтерма $F_{10} = \bar{x}_1x_2\bar{x}_3x_4\bar{x}_5$ $r = 5$, и для макстерма $\Phi_2 = \bar{x}_1 + x_2 + \bar{x}_3$ $r = 3$.

На основании вышесказанного можно сформулировать следующую теорему.

Теорема. *Любая таблично заданная ФАЛ может быть представлена аналитически в виде*

$$f(x_1, x_2, \ldots, x_n) = F_1 \vee F_2 \vee \ldots \vee F_n = \bigvee_i F_i, \qquad (2.11)$$

где i — номера наборов, на которых функция равна 1; $\bigvee_i$ — знак дизъюнкции, объединяющий все термы F_i, равные единице.

В самом деле, если на каком-либо наборе функция $f(x_1^*, x_2^*, \ldots, x_n^*) = 1$, то вследствие того, что $x \vee 1 = 1$, в правой части выражения (2.11) всегда найдется элемент, равный единице; если же на наборе i функция $f(x_1^*, x_2^*, \ldots, x_n^*) = 0$, то в правой части не найдется ни одного элемента, равного 1, так как $0 \vee 0 \vee \ldots \vee 0 = 0$.

Таким образом, каждому набору i, для которого $f_i = 1$, соответствует элемент $F_i = 1$, а наборам i, на которых $f_i = 0$, не соответствует ни одного элемента $F_i = 1$. Поэтому таблица истинности однозначно отображается аналитической записью вида (2.11), которую в дальнейшем будем называть *объединением термов.*

Нормальная дизъюнктивная форма (*НДФ*) — объединение термов, включающее минтермы переменного ранга.

Количество всех термов, входящих в состав (2.11), равно количеству единичных строк таблицы.

Пример 2.4. Записать в аналитическом виде функцию, заданную таблицей 2.9

Таблица 2.9

x_1	x_2	x_3	$f(x_1, x_2, x_3)$	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	1	1	0	0	1
0	0	1	0	1	0	1	0
0	1	0	0	1	1	0	0
0	1	1	1	1	1	1	0

Р е ш е н и е . На основании теоремы эту функцию можно записать в аналитической форме:

$$f(x_1, x_2, x_3) = F(0,0,0) + F(0,1,1) + F(1,0,0) = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2\bar{x}_3.$$

Ответ $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2\bar{x}_3$.

Для представления ФАЛ в (2.11) используется совокупиость термов, объединенных знаками дизъюикции ($\vee$ или +). Можио использовать также другую элементарную логическую операцию. Сформулируем основиые требования к этой операции.

Требование 1: если какой-либо терм $F_i = 1$, то фуикция f должна быть равна единице.

Требование 2: если какой-либо терм $F_i = 0$, то функция f может быть равна единице.

Необходимо, чтобы при значениях термов $F_i = 0$ функция f была равиа нулю.

Табличное представлеиие искомой логической операции имеет вид таблиц 2.10 н 2.11.

Таблица 2.10

F_i	F_j	$\Delta = \vee$
0	0	0
0	1	1
1	0	1
1	1	1

Таблица 2.11

F_i	F_j	$\Delta = \oplus$
0	0	0
0	1	1
1	0	1
1	1	0

Таким образом, получили, что искомой функцией, кроме функции ИЛИ, может быть функция разноименности и при этом справедливым становится такое следствие из теоремы (2.11):

С л е д с т в и е : любая таблично заданная ФАЛ может быть представлена в следующей аналитической форме:

$$f(x_1, x_2, \ldots, x_n) = F_1\Delta F_2\Delta \ldots \Delta F_k, \qquad (2.12)$$

где знак Δ обозначает операции $\vee$, $\oplus$.

Требования 1 и 2 можно обобщить и потребовать, чтобы аиалитическое представление нулевых и единичных строчек таблицы различалось и чтобы выполнялось взаимно-однозначное соответствие между нулевой единичной строкой и термом.

Требование 3: если какой-либо терм $\Phi_i = 0$, то фуикция f должна быть равна нуло.

Требование 4: если все термы $\Phi_i = 0$, то функция $f = 1$.

Выполняя эти требоваиия, можио прийти к двум другим возможным функциям: конъюнкции и равнозначности (табл. 2.12 и 2.13).

Таблица 2.12

Φ_1	Φ_2	&
0	0	0
0	1	0
1	0	0
1	1	1

Таблица 2.13

Φ_1	Φ_2	$\equiv$
0	0	1
0	1	0
1	0	0
1	1	1

Теорема. *Любая таблично заданная ФАЛ может быть задана в аналитической форме:*

$$f(x_1, x_2, \ldots, x_n) = \Phi_1 \,\&\, \Phi_2 \,\&\, \ldots \,\&\, \Phi_k, \qquad (2.13)$$

где k — количество двоичных наборов, для которых $\Phi = 0$.

Нормальная конъюнктивная форма (НКФ) — объединение термов (2.13), включающее в себя макстермы разных рангов.

Следствие: любая таблично заданная ФАЛ может быть представлена в аналитической форме:

$$f(x_1, x_2, \ldots, x_n) = \Phi_1 \equiv \Phi_2 \equiv \ldots \equiv \Phi_k, \qquad (2.14)$$

где k — количество нулевых значений функции.

2.6. Совершенные нормальные формы

Нормальные конъюнктивная, дизъюнктивная формы не дают однозначного представления функции. Такое представление получается только при совершенных нормальных формах (СНФ).

Введем обозначения $x^0 = \bar{x}$, $x^1 = x$.

Тогда в общем виде переменная может быть задана как некоторая функция

$$x^\alpha = \begin{cases} x, \text{ если } \alpha = 1; \\ \bar{x}, \text{ если } \alpha = 0, \end{cases} \qquad (2.15)$$

при этом

$$x^\alpha = \alpha x + \bar{\alpha}\bar{x}, \qquad (2.16)$$

где α — двоичная переменная.

Рассмотрим конъюнкцию вида $x_1^{\alpha_1}x_2^{\alpha_2}\dots x_n^{\alpha_n}$, где $\{\alpha_1, \alpha_2, \dots, \alpha_n\}$ представляет собой двоичный набор; число наборов равно 2^n, т. е.

0 0 ... 0 0 0
0 0 ... 0 0 1
0 0 ... 0 1 0
0 0 ... 0 1 1

и т. д.

Если задать всем α_1 значение 0 и 1, то можно получить, например, дизъюнкцию вида

$$\vee_1 x_1^{\alpha_1}x_2^{\alpha_2}\dots x_n^{\alpha_n} = x_1^0x_2^0\dots x_n^0 \vee x_1^0x_2^0\dots x_n^1 \vee x_1^0x_2^0\dots x_{n-1}^1x_n^0 \vee \dots \vee x_1^1x_2^1\dots x_n^1, \quad (2.17)$$

где $\vee$ — символ обобщенной дизъюнкции по единичным строкам.

Тогда справедлива следующая теорема.

Теорема. *Любая ФАЛ может быть представлена в виде*

$$f(x_1, x_2, \dots, x_k, \dots, x_n) = \vee x_1^{\alpha_1}x_2^{\alpha_2}\dots x_k^{\alpha_k} f(\alpha_1 \dots \alpha_k x_{k+1} \dots x_n). \quad (2.18)$$

Выражение (2.18) называют *разложением функции алгебры логики по k переменным.*

Доказательство. Прежде всего определим, при каких условиях выполняется равенство $x^\alpha = 1$.

Очевидно, что это имеет место при $x = \alpha$. В самом деле, если $x = \alpha$, то $\alpha^\alpha = \alpha\alpha + \bar{\alpha}\bar{\alpha} = \alpha + \bar{\alpha} = 1$; если $x = \bar{\alpha}$, то $\bar{\alpha}^\alpha = \bar{\alpha}\alpha + \bar{\alpha}\bar{\bar{\alpha}} = \alpha\bar{\alpha} = 0$.

С учетом того, что $x^\alpha = 1$ при $x = \alpha$, можно утверждать, что конъюнкция вида $x_1^{\alpha_1}x_2^{\alpha_2}\dots x_n^{\alpha_n}$ равна единице при $x_1 = \alpha_1$, $x_2 = \alpha_2 \dots x_n = \alpha_n$.

Эти равенства можно подставить в правую часть выражения (2.18). В результате $\underbrace{\vee_1 x_1^{\alpha_1}x_2^{\alpha_2}\dots x_k^{\alpha_k}}_{1} f(x_1, x_2, \dots, x_k, \dots, x_n) =$

$= f(x_1, x_2, \dots, x_k, \dots, x_n)$, что и требовалось доказать.

Следствие 1: если $k = 1$, то функция алгебры логики представляется в виде

$$f(x_1, x_2, \dots, x_n) = x_1 f(1, x_2, \dots, x_n) \vee \bar{x}_1 f(0, x_2, \dots, x_n). \quad (2.19)$$

Следствие 2: если $k = n$, то любая ФАЛ может быть представлена в виде

$$f(x_1, x_2, \dots, x_n) = \vee x_1^{\alpha_1}x_2^{\alpha_2}\dots x_n^{\alpha_n}. \quad (2.20)$$

Совершенная нормальная дизъюнктивная форма (*СНДФ*) — ФАЛ, заданная в виде (2.20).

Рассмотрим основные свойства СНДФ:

в СНДФ нет двух одинаковых миитермов;

в СНДФ ни один минтерм не содержит двух одинаковых множителей (переменных);

в СНДФ ни один минтерм не содержит вместе с переменной и ее отрицание.

На основании этих свойств можно предложить следующий алгоритм получения СНДФ из таблицы истинности:

1. Положить номер строки в таблице $i = 1$, номер элемента в строке $j = 1$.

2. Выбрать из таблицы набор с номером i. Если $f_i = 1$, то перейти к п. 3, если $f_i \neq 1$, — к п. 5.

3. Сформировать терм F_i. Выбрать элемент строки с номером j.

$$\text{Если } x_j = \begin{cases} 0, \text{ то } F_i := F_i \wedge \bar{x}_j, \\ 1, \text{ то } F_i := F_i \wedge x_j. \end{cases}$$

4. Вычислить $j := j + 1$. Если $i < n$, то перейти к п. 3, если $i \geq n$, — к п. 5.

5. Вычислить $i := i + 1$. Если $i < 2^n$, то перейти к п. 2, если $i \geq 2^n$, — к п. 6.

6. Конец.

Пример 2.5. Представить функцию, заданную таблицей 2.14, в СНДФ.

Таблица 2.14

x_1	x_2	x_3	f	x_1	x_2	x_3	f
0	0	0	0	1	0	0	1
0	0	1	0	1	0	1	0
0	1	0	0	1	1	0	1
0	1	1	1	1	1	1	0

Р е ш е н и е. Решение проводится в соответствии с вышеприведенным алгоритмом

Ответ $f(x_1, x_2, x_3) = \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 \bar{x}_3$.

Рассмотрим СНФ для других функций.

Пусть имеется терм вида $\Phi_i = x_1^{\bar{\alpha}_1} \vee x_2^{\bar{\alpha}_2} \vee \ldots \vee x_n^{\bar{\alpha}_n}$, где

$$\Phi_i = \begin{cases} 0, \text{ если номер набора равен } i; \\ 1, \text{ если номер набора не равен } i. \end{cases}$$

Если $x_i = \alpha_i$ и α_i — текущий элемент двоичного набора, то тогда возможны случаи:

1) $x_i = 0$	2) $x_i = 0$	3) $x_i = 1$	4) $x_i = 1$
$\alpha_i = 0$	$\alpha_i = 1$	$\alpha_i = 0$	$\alpha_i = 1$
$\bar{\alpha}_i = 1$	$\bar{\alpha}_i = 0$	$\bar{\alpha}_i = 1$	$\bar{\alpha}_i = 0$
$x_i^{\bar{\alpha}_i} = x_i = 0$	$x_i^{\bar{\alpha}_i} = \bar{x}_i = 1$	$x_i^{\bar{\alpha}_i} = x_i = 1$	$x_i^{\bar{\alpha}_i} = \bar{x}_i = 0$.

В случае, когда $x_i^{\bar{\alpha}_i} = 0$ для $x_i = \alpha_i$, терм Φ_i можно использовать в представлении (2.13) и (2.14).

Теорема. *Любая ФАЛ, кроме абсолютно истинной функции, может быть представлена в совершенной конъюнктивной нормальной форме (СКНФ):*

$$f(x_1, x_2, \ldots, x_n) = \underset{0}{\wedge}(x_1^{\bar{\alpha}_1} \vee x_2^{\bar{\alpha}_2} \vee \ldots \vee x_n^{\bar{\alpha}_n}). \qquad (2.21)$$

Для представления логической функции используются операции И, ИЛИ, НЕ ($\wedge$, $\vee$, $\neg$).

Следствие: любая ФАЛ, кроме конституэнты 1, может быть представлена в виде

$$f(x_1, x_2, \ldots, x_n) = \underset{0}{\equiv}(x_1^{\bar{\alpha}_1} \ldots \vee x_2^{\bar{\alpha}_2} \vee \ldots \vee x_n^{\bar{\alpha}_n}). \qquad (2.22)$$

Здесь используются операции неравнозначности, дизъюнкции, отрицания ($\equiv$, $\vee$, $\neg$).

Пример 2.6. Найти СКНФ для функции (см. табл. 2.11).

Решение. Решение проводится по формуле (2.21)

$$f(x_1, x_2, \ldots, x_n) = (x_1 \vee x_2 \vee x_3)(x_1 \vee \bar{x}_2 \vee x_3)(\bar{x}_1 \vee \bar{x}_2 \vee x_3)(\bar{x}_1 \vee x_2 \vee \bar{x}_3)(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3).$$

Это представление более громоздкое, чем представление этой функции в СНДФ, так как в таблице много строк, для которых $f = 0$.

В рассмотренных ранее объединениях термов были использованы для записи термов F_{ij} и Φ_{ij} операции ИЛИ, НЕ, а также И, НЕ. Можно использовать также набор из импликации ($\rightarrow$) и отрицания (НЕ).

Способы преобразования НФ в СНФ. Совершенная нормальная форма отличается от нормальной формы (НФ) тем, что всегда содержит термы только максимального ранга и дает однозначное представление функции.

Произвольная нормальная дизъюнктивная форма (НДФ) переводится в СНДФ следующим образом.

Пусть $f_{\text{НФ}} = F_1$. Тогда

$$f_{\text{СНФ}} = F_1 x_i \vee F_1 \bar{x}_i, \tag{2.23}$$

где x_i — переменная, которая не входит в данный терм F_1.

Пример 2.7. Логическую функцию, заданную в НДФ

$$f(x_1, x_2, x_3, x_4) = \underset{F_1}{x_1 \bar{x}_2} \vee \underset{F_2}{x_2 \bar{x}_3 \bar{x}_4} \vee \underset{F_3}{\bar{x}_1 \bar{x}_3 x_4} \vee \underset{F_4}{x_1 x_2 x_3 x_4},$$

преобразовать в СНДФ.

Р е ш е н и е. Воспользуемся приемом преобразования (2.23) поочередно к термам $F_1 = x_1 x_2 (x_3 \vee \bar{x}_3) = x_1 \bar{x}_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3$.

Оба члена полученного выражения умножим на $(x_4 \vee \bar{x}_4)$.

В результате получим

$$F_1 = x_1 \bar{x}_2 x_3 x_4 \vee x_1 \bar{x}_2 \bar{x}_3 x_4 \vee x_1 \bar{x}_2 x_3 \bar{x}_4 \vee x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4.$$

Аналогично,

$$F_2 = x_2 \bar{x}_3 x_4 \vee (x_1 \vee \bar{x}_1) = x_1 x_2 \bar{x}_3 x_4 \vee \bar{x}_1 x_2 \bar{x}_3 x_4,$$
$$F_3 = \bar{x}_1 \bar{x}_3 x_4 (x_2 \vee \bar{x}_2) = \bar{x}_1 x_2 \bar{x}_3 x_4 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4.$$

После приведения подобных членов определяем СНДФ.

Ответ: $f(x_1, x_2, x_3, x_4) = x_1 x_2 \bar{x}_3 x_4 \vee x_1 \bar{x}_2 x_3 x_4 \vee x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \vee x_1 \bar{x}_2 \bar{x}_3 x_4 \vee x_1 \bar{x}_2 \bar{x}_1 \bar{x}_4 \vee \vee \bar{x}_1 x_2 \bar{x}_3 x_4 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4 \vee x_1 x_2 x_3 x_4$.

Если максимальный ранг для функции равен r, а минимальный ранг j-го терма равен k, то преобразование (2.23) необходимо применить к j-му терму $(r - k)$ раз.

Произвольная НКФ переводится в СКНФ путем следующего преобразования.

Пусть $f_{\text{НКФ}} = \Phi_1$. Тогда

$$f_{\text{СНКФ}} = \Phi_1 \vee x_i \bar{x}_i = (\Phi_1 \vee x_i)(\Phi_1 \vee \bar{x}_i). \tag{2.24}$$

Пример 2.8. Преобразовать в СКНФ логическую функцию

$$f(x_1, x_2, x_3) = \underset{\Phi_1}{(x_1 \vee x_2)} \underset{\Phi_2}{(\bar{x}_2 \vee \bar{x}_3)} \underset{\Phi_3}{(x_1 \vee x_2 \vee x_3)}.$$

Р е ш е н и е. Применяем правило преобразований (2.24) поочередно к термам Φ_1 и Φ_2:

$$\Phi_1 = (x_1 \vee x_2) \vee x_3 \bar{x}_3 = (x_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee \bar{x}_3);$$
$$\Phi_2 = (x_2 \vee \bar{x}_3) \vee x_1 \bar{x}_1 = (x_1 \vee x_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee \bar{x}_3).$$

После упрощений СКНФ примет окончательный вид.

Ответ: $f_{\text{СКНФ}}(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee \bar{x}_3)$.

2.7. Системы функций алгебры логики

Рассмотрим теорему Жегалкина, которая играет важную роль в алгебре логики.

Теорема Жегалкина. *Любая булева функция может быть представлена многочленом вида* $f(x_1, x_2, \ldots, x_n) = k_0 \oplus k_1x_1 \oplus k_2x_2 \oplus \ldots \oplus k_{n+1}x_1x_2 \oplus k_{n+2}x_1x_3 \oplus \ldots \oplus k_{n+m}x_1x_2x_n$, *где* k_i *— коэффициенты, принимающие значения* 0 *или* 1.

Теорема Жегалкина дает возможность представить любую логическую функцию в виде полиномов разной степени.

Существует несколько классов ФАЛ, которые также важны для логического анализа.

Класс линейных функций ($\mathbf{K_л}$). Булева функция называется *линейной*, если она представляется полиномом первой степени:

$$f(x_1, x_2, \ldots, x_n) = k_0 \oplus k_1x_1 \oplus k_2x_2 \oplus \ldots \oplus k_nx_n .$$

Количество линейных функций равно 2^{n+1}. Например, для $n = 2$ количество линейных функции равно восьми, т. е.

1) $f_1(x_1, x_2) = 0$; 2) $f_2(x_1, x_2) = x_1$; 3) $f_3(x_1, x_2) = x_2$; 4) $f_4(x_1, x_2) = x_1 \oplus x_2$;
5) $f_5(x_1, x_2) = 1 \oplus x$; 6) $f_6(x_1, x_2) = 1 \oplus x_2$; 7) $f_7(x_1, x_2) = 1 \oplus x_1 \oplus x_2$;
8) $f_8(x_1, x_2) = 1$.

Класс функций, сохраняющих нуль ($\mathbf{K_0}$). Если функция на нулевом наборе переменных равна нулю, то говорят, что функция сохраняет нуль:

$$f(0, 0, \ldots, 0) = 0 .$$

Для двух переменных (табл. 2.15) такими функциями являются $f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8$.

Класс функций, сохраняющих единицу ($\mathbf{K_1}$). Если функция на единичном наборе переменных равна единице, то говорят, что такая функция сохраняет единицу: $f(1, 1, \ldots, 1) = 1$. Для двух переменных такими функциями являются $f_2, f_4, f_6, f_8, f_{10}, f_{12}, f_{14}, f_{16}$ (см. табл. 2.15).

Класс монотонных функций ($\mathbf{K_м}$). Функция алгебры логики называется *монотонной*, если при любом возрастании набора значения этой функции не убывают. Примером таких функций для двух переменных являются функции $f_1, f_2, f_4, f_6, f_8, f_{14}$ (см. табл. 2.15).

Таблица 2.15

Функция	00	01	10	11	Классы				
					$K_л$	K_0	K_1	$K_м$	$K_с$
f_1	0	0	0	0	v	v		v	
f_2	0	0	0	1		v	v	v	
f_3	0	0	1	0		v			
f_4	0	0	1	1	v	v	v	v	v
f_5	0	1	0	0		v			
f_6	0	1	0	1	v	v	v		v
f_7	0	1	1	0	v	v			
f_8	0	1	1	1	—	v	v	v	
f_9	1	0	0	0		—	–	—	–
f_{10}	1	0	0	1	v		v		
f_{11}	1	0	1	0	v				v
f_{12}	1	0	1	1			v		
f_{13}	1	1	0	0	v				v
f_{14}	1	1	0	1			v		
f_{15}	1	1	1	0	—	—	—	–	–
f_{16}	1	1	1	1			v	v	

Класс самодвойственных функций ($K_с$). Функция алгебры логики является самодвойственной, если на каждой паре противоположных наборов она принимает противоположные значения, т. е.

$$f(x_1, x_2, \ldots, x_n) = \overline{f(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)}.$$

Для двух переменных такими функциями являются f_4, f_6, f_{11}, f_{13}.

Все названные выше классы функций обладают замечательным свойством: любая функция алгебры логики, полученная с помощью операции суперпозиции и подстановки из функций одного класса, обязательно будет принадлежать этому же классу.

Базисом называется полиая система ФАЛ, с помощью которой любая ФАЛ может быть представлена суперпозицией исходных функций.

Теорема Поста–Яблонского. *Для того чтобы система ФАЛ была полной, необходимо и достаточно, чтобы она содержала хотя бы одну функцию:*

не сохраняющую нуль,
не сохраняющую единицу,
не являющуюся линейной,
не являющуюся монотонной,
не являющуюся самодвойственной.

В таблице 2.15 представлены свойства функции двух переменных*.

К базису относится система функций И, ИЛИ, НЕ (базис 1), свойства которых были изучены Дж. Булем. Поэтому алгебра высказываний, построенная на основе этих функций, названа булевой алгеброй. Базисами являются также системы, содержащие функции И, НЕ (базис 2), ИЛИ, НЕ (базис 3), состоящие из функции Шеффера (базис 4) и функции Пирса (Вебба) (базис 5). Это перечисление показывает, что базисы могут быть избыточными (базис 1) и минимальными (базисы 4 и 5).

Базис минимальный, если удаление хотя бы одной функции превращает систему ФАЛ в неполную.

Проблема простейшего представления логических функций сводится к выбору не только базиса, но и формы наиболее экономного представления этих функций.

Базис И, ИЛИ, НЕ является избыточной системой, так как возможно удаление из него некоторых функций. Например, используя законы де Моргана, можно удалить либо функцию И, заменив ее на функции ИЛИ и НЕ, либо функцию ИЛИ, заменяя ее на функции И и НЕ.

Если сравнить в смысле минимальности различные формы представлений ФАЛ, станет ясно, что нормальные формы экономичнее совершенных нормальных форм. Но, с другой стороны, нормальные формы не дают однозначного представления.

Минимальная форма представления ФАЛ — форма представления ФАЛ, которая содержит минимальное количество термов и переменных в термах (т. е. минимальная форма не допускает никаких упрощений).

Например, функция $f(x_1, x_2, \ldots, x_n) = x_1 + x_2$ является минимальной формой и, наоборот, функция $x_1 + \bar{x}_1 x_2$ может быть упрощена, если к этому выражению применить распределительный закон, т. е. $x_1 + \bar{x}_1 x_2 = (x_1 + \bar{x}_1)(x_1 + x_2) = x_1 + x_2$.

* Функции f_0 и f_{15} не принадлежат ни одному из указанных классов.

Следовательно, упрощение сложных логических выражений может быть осуществлено по основным законам и аксиомам, изложенным выше.

Пример 2.9. Упростить функцию $f(A, B, C) = AB + BC + BC + AB$ в базисе 1.

Р е ш е н и е. Сначала примем правило (2.23), а затем упростим функцию.

$$f(A, B, C) = AB + BC + BC(A + A) + AB(C + C) = AB + BC + ABC + ABC + ABC + ABC =$$
$$= A\overline{B}(1 + C) + B\overline{C}(1 + \overline{A}) + \overline{A}C(B + B) = A\overline{B} + B\overline{C} + \overline{A}C.$$

Ответ: $f(A, B, C) = A\overline{B} + B\overline{C} + \overline{A}C$.

Пример 2.10. Упростить функцию $f(A, B, C, D) = A\overline{B} + C + \overline{A}\overline{C}D + B\overline{C}D$ в базисе 1

Р е ш е н и е. Нетрудно доказать, что $x + \overline{xy} = x + y$. Используя также теорему де Моргана, получим $x + y = \overline{x}\overline{y}$.

Тогда $C + \overline{A}\overline{C}D = C + \overline{A}D$; $C + B\overline{C}D = C + BD$.

Следовательно, $f(A, B, C, D) = A\overline{B} + C + \overline{A}D + BD = A\overline{B} + D(\overline{A} + B) + C = A\overline{B} + D\overline{A}\overline{B} +$
$+ C = A\overline{B} + C + D$.

Ответ: $f(A, B, C, D) = A\overline{B} + C + D$.

Задание для самоконтроля

1. Составить таблицу истинности для заданных функций:

а) $f_1(x_1, x_2, x_3) = \bigvee_1(1, 2, 3, 5)$;

б) $f_2(x_1, x_2, x_1) = \bigvee_1(1, 2, 3, 5, 6, 7)$;

в) $f_3(x_1, x_2, x_3) = \bigvee_1(1, 3, 8, 9, 10, 11, 12, 14)$;

2. Доказать тождественность функций:

а) $\overline{AC + B\overline{C}} = \overline{A}C + \overline{B}\overline{C}$,

б) $(A + B)(\overline{A} + C) = AC + \overline{A}B$;

в) $AB + BC + AC = \overline{A}\overline{B} + \overline{B}\overline{C} + \overline{A}\overline{C}$.

3. Найти минимальную форму для функций, заданных в примере 1.

4. Найти СДНФ и СКНФ для следующих функций:

а) $f_1(x_1, x_2, x_3) = x_1 + \overline{x}_2x_3 + \overline{x}_1x_2x_3 + x_1\overline{x}_3$;

б) $f_2(x_1, x_2, x_3) = x_1 + x_2$;

в) $f_3(x_1, x_2, x_3) = \overline{(x_1x_2 + x_2x_3)} \,\&\, \overline{x_1x_2}$.

3. ПРЕДСТАВЛЕНИЕ ЧИСЛОВОЙ ИНФОРМАЦИИ В ИНФОРМАЦИОННЫХ СИСТЕМАХ

3.1. Выбор системы счисления для представления числовой информации

Информация во внешнем по отношению к ЭВМ мире представляется в непрерывном или дискретном виде. Внутри ЭВМ информация всегда представляется в виде чисел, записанных в той или иной системе счисления. Если же речь идет о текстовой информации, то обычно она кодируется также с помощью чисел. Вопрос о выборе системы счисления для цифрового автомата — один из важнейших вопросов проектирования как алгоритмов функционирования отдельных устройств, так и расчета технических характеристик этого автомата [11, 12].

Система счисления — совокупность приемов и правил для записи чисел цифровыми знаками или символами. Наиболее известна десятичная система счисления, в которой для записи чисел используются цифры 0, 1, ..., 9. Способов записи чисел цифровыми знаками существует бесчисленное множество. Любая предназначенная для практического применения система счисления должна обеспечивать:

возможность представления любого числа в рассматриваемом диапазоне величин;

единственность представления (каждой комбинации символов должна соответствовать одна и только одна величина);

простоту оперирования числами.

Все системы представления чисел делят на позиционные и непозиционные. Самый простой способ записи чисел может быть описан выражением

$$A_{(D)} = D_1 + D_2 + \ldots + D_k = \sum_{i=1}^{i=k} D_i ,$$

где $A_{(D)}$ — запись числа A в системе счисления D; D_i — символы системы, образующие базу $D = \{D_1, D_2, \ldots, D_k\}$.

По этому принципу построены непозиционные системы счисления.

Непозиционная система счисления — система, для которой значение символа не зависит от его положения в числе.

Принципы построения таких систем не сложны. Для их образования используют в основном операции сложения и вычитания. Например, система с одним символом (палочкой) встречалась у многих народов. Для изображения какого-то числа в этой системе нужно записать количество палочек, равное данному числу. Эта система неэффективна, так как запись числа получается длинной. Другим примером непозиционной системы счисления является римская система, использующая набор следующих символов: I, X, V, L, C, D, M и т. д. В этой системе существует отклонение от правила независимости значения цифры от положения в числе. В числах LX и XL символ X принимает два различных значения: $+10$ — в первом случае и -10 — во втором.

В общем случае системы счисления можно построить по следующему принципу:

$$A_B = a_1 B_1 + a_2 B_2 + \ldots + a_n B_n , \qquad (3.1)$$

где A_B — запись числа A в системе счисления с основанием B_i; a_i — цифра (символ) системы счисления с основанием B_i; B_i — база, или основание системы.

Если предположить, что $B_i = q^i$, то с учетом (3.1)

$$B_i = q_i B_{i-1} . \qquad (3.2)$$

Позиционная система счисления — система, удовлетворяющая равенству (3.2).

Естественная позиционная система счисления имеет место, если q — целое положительное число.

В позиционной системе счисления значение цифры определяется ее положением в числе: один и тот же знак принимает различное значение. Например, в десятичном числе 222 первая цифра справа означает две единицы, соседняя с ней — два десятка, а левая — две сотни.

Любая позиционная система счисления характеризуется основанием. *Основание* (*базис*) *q естественной позиционной системы счисления* — количество знаков или символов, используемых для изображения числа в данной системе. Возможно бесчисленное множество позиционных систем, так как, приняв за основание любое число, можно образовать новую систему. Например, запись числа в шестнадцатеричной системе может проводиться с помощью следующих знаков (цифр): 0, 1, ..., 9, *A*, *B*, *C*, *D*, *E*, *F* (вместо *A*, ..., *F* можно записать любые другие символы, например, $\overline{1}$, $\overline{2}$, ..., $\overline{5}$).

Для позиционной системы счисления справедливо равенство

$$A_q = \sum_{i=-m}^{i=n} a_i q^i , \qquad (3.3)$$

или $A_q = a_n q^n + \ldots + a_1 q^1 + a_0 q^0 + a_{-1} q^{-1} + \ldots + a_{-m} q^{-m}$, где A_q — произвольное число, записанное в системе счисления с основанием q; $n+1$, m — количество целых и дробных разрядов.

На практике используют сокращенную запись чисел:

$$A_q = a_n a_{n-1} \ldots a_1 a_0 a_{-1} \ldots a_{-m} .$$

В восьмеричной системе счисления числа изображают с помощью цифр 0, 1, ..., 7. Например, $124{,}537_8 = 1 \cdot 8^2 + 2 \cdot 8^1 + 4 \cdot 8^0 + 5 \cdot 8^{-1} + 3 \cdot 8^{-2} + 7 \cdot 8^{-3}$.

В двоичной системе счисления используют цифры 0, 1. Например, $1001{,}1101_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4}$.

Для записи чисел в троичной системе берут цифры 0, 1, 2. Например, $2122_3 = 2 \cdot 3^3 + 1 \cdot 3^2 + 2 \cdot 3^1 + 2 \cdot 3^0$.

В таблице 3.1 приведены эквиваленты десятичных цифр в различных системах счисления.

Таблица 3.1

Десятичная цифра	Эквиваленты в других системах счисления				
	$q=2$	$q=3$	$q=5$	$q=8$	$q=16$
0	0000	000	00	00	0
1	0001	001	01	01	1
2	0010	002	02	02	2
3	0011	010	03	03	3
4	0100	011	04	04	4
5	0101	012	10	05	5
6	0110	020	11	06	6
7	0111	021	12	07	7
8	1000	022	13	10	8
9	1001	100	14	11	9

Для любой позиционной системы счисления справедливо, что основание изображается символом 10 в своей системе, т. е. любое число можно записать в виде

$$A_q = a_n \cdot 10^n + \ldots + a_1 \cdot 10^1 + a_0 \cdot 10^0 + a_{-1} \cdot 10^{-1} + \ldots + a_{-m} \cdot 10^{-m} . \qquad (3.4)$$

В ЭВМ используют в основном позиционные системы счисления. В дальнейшем для простоты изложения будем употреблять термин «система счисления», имея в виду позиционные системы.

Вес разряда p_i, *числа в позиционной системе счисления* выражается соотношением

$$p_i = q^i / q^0 = q^i, \tag{3.5}$$

где i — номер разряда при отсчете справа налево.

Если разряд имеет вес $p_i = 10^k$, то следующий старший разряд будет иметь вес $p_{i+1} = 10^{k+1}$, а соседний младший разряд — вес $p_{i-1} = 10^{k-1}$. Такая взаимосвязь разрядов приводит к необходимости передачи информации между ними.

Если в данном разряде накопилось значение единиц, равное или большее q, то должна происходить передача единицы в соседний старший разряд. При сложении такие передачи информации называют *переносами*, а при вычитании — *заемами*. Передача переносов или заемов происходит последовательно от разряда к разряду.

Длина числа (ДЧ) — количество позиций (или разрядов) в записи числа [14]. В техническом аспекте длина числа интерпретируется как длина разрядной сетки (ДРС).

Для разных систем счисления характерна разная длина разрядной сетки, необходимая для записи одного и того же числа. Например, $96 = 120_8 = 10120_3 = 1100000_2$. Здесь одно и то же число, записанное в разных базисах, имеет разную длину разрядной сетки. Чем меньше основание системы, тем больше длина числа.

Если длина разрядной сетки задана, то это ограничивает максимальное (или минимальное) по абсолютному значению число, которое может быть записано.

Пусть длина разрядной сетки равна любому положительному числу, например n. Тогда $A_{q\,\max} = q^n - 1$; $A_{q\,\min} = -(q^n - 1)$.

Диапазон представления (ДП) *чисел в заданной системе счисления* — интервал числовой оси, заключенный между максимальным и минимальным числами, представленными длиной разрядной сетки:

$$A_{q\,\max} \geq \text{ДП} \geq A_{q\,\min}. \tag{3.6}$$

Правильный выбор системы счисления — важный практический вопрос, поскольку от его решения зависят такие технические характеристики проектируемой ЭВМ, как скорость вычислений, объем памяти, сложность

алгоритмов выполнения арифметических операций. При выборе системы счисления для ЭВМ необходимо учитывать следующее:

основание системы счисления определяет количество устойчивых состояний, которые должен иметь функциональный элемент, избираемый для изображения разрядов числа;

длина числа существенно зависит от основания системы счисления;

система счисления должна обеспечить простые алгоритмы выполнения арифметических и логических операций.

Десятичная система, столь привычная в повседневной жизни, не является наилучшей с точки зрения ее технической реализации в ЭВМ. Известные в настоящее время элементы, обладающие десятью устойчивыми состояниями (элементы на основе сегнетокерамики, декатроны и др.), имеют невысокую скорость переключения, а следовательно, не могут обеспечить соответствующее быстродействие машины.

Подавляющее большинство компонентов электронных схем, применяемых для построения ВМ, — двухпозиционные. С этой точки зрения для ЭВМ наиболее подходит двоичная система счисления. Но рационально ли использование этой системы с точки зрения затрат оборудования? Для ответа на этот вопрос введем показатель экономичности системы C — произведение основания системы на длину разрядной сетки, выбранную для записи чисел в этой системе:

$$C = qN, \tag{3.7}$$

где q — основание системы счисления; N — количество разрядов.

Если принять, что каждый разряд числа представлен некоторым элементом с q устойчивыми состояниями, а q элементами, каждый из которых имеет одно устойчивое состояние, то показатель экономичности укажет условное количество оборудования, которое необходимо затратить на представление чисел в этой системе.

Максимальное число, которое можно изобразить в системе с основанием q,

$$A_{q\max} = q^N - 1. \tag{3.8}$$

Из (3.8) можно найти требуемую длину разрядной сетки:

$$N = \log_q(A_{q\max} + 1). \tag{3.9}$$

Тогда для любой системы счисления $C = q\log_q(A_{q\max} + 1)$.

Представим, что величина q принимает любые значения (целочисленные и дробные), т. е. является непрерывной величиной. Это необходимо

для того, чтобы рассматривать величину C как функцию от величины q. Данное допущение не является строгим, однако позволяет получить интересный вывод: если за единицу измерения оборудования принят условный элемент с одним устойчивым состоянием, то для сравнения двух систем счисления можно ввести относительный показатель экономичности

$$F = q \log_q (A_{q\,\max} + 1) / [2 \log_2 (A_{2\,\max} + 1)], \qquad (3.10)$$

позволяющий сравнить любую систему счисления с двоичной.

Если функция F непрерывна, то, как видно из приведенного ниже соотношения, она имеет минимум.

q	2	3	4	6	8	10
F	1,000	0,946	1,000	1,148	1,333	1,505

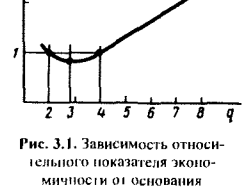


Рис. 3.1. Зависимость относительного показателя экономичности от основания системы счисления

На рис. 3.1 представлена зависимость величины F от основания системы счисления q. Нижняя точка графика соответствует минимуму функции F, определяемому из условия $dF/dq = 0$, что соответствует значению $q = e$. Следовательно, с точки зрения минимальных затрат условного оборудования, наиболее экономичной является система счисления с основанием, равным $e \approx 2{,}72$.

Используя (3.10), можно доказать, что троичная система счисления экономичнее двоичной. В подавляющем большинстве ЭВМ используют двоичную систему счисления, однако для ЭВМ это связано с преодолением дополнительных трудностей, возникающих при переводе входной информации в двоичную систему счисления и двоичной информации в выходную информацию.

3.2. Перевод числовой информации из одной позиционной системы в другую

В процессе преобразования информации в цифровом автомате возникает необходимость перевода чисел из одной позиционной системы счисления в другую. Это обусловлено тем, что в качестве внутреннего алфавита наиболее целесообразно использовать двоичный алфавит с символами 0 и 1*.

* Первым два символа для кодирования информации применил известный философ XVII в. Ф. Бэкон, который использовал символы 0, 1.

Рассмотрим задачу перевода чисел в общей постаиовке.

В соответствии с (3.3) числа в разных системах счисления можно представить следующим образом:

$$A_{q_1} = \sum_{i=-m}^{i=n} a_i q_1^i = \sum_{j=-s}^{j=k} b_j q_2^j = A_{q_2}. \qquad (3.11)$$

В общем виде задачу перевода числа из системы счисления с осиоваиием q_1 в систему счисления с основанием q_2 можно представить как задачу определения коэффициентов b_i иового ряда, изображающего число в системе с основанием q_2. Решить эту задачу можно подбором коэффициеитов b_i. Основная трудиость при этом заключается в выборе максимальиой степеии, которая еще содержится в числе A_{q_1}. Все действия должиы выполияться по правилам q_1-арифметики, т. е. по правилам исходиой системы счисления.

После нахождения максимальной степени осиоваиия проверяют «вхождение» в заданное число всех степеией иового основания, меньших максимальиого. Каждая из отмеченных степеней может «входить» в ряд не более $q_2 - 1$ раз, так как для любого коэффициеита ряда накладывается ограничение:

$$0 \le a_i \le q_1 - 1;\ 0 \le b_j \le q_2 - 1. \qquad (3.12)$$

Пример 3.1*. Перевести десятичное число $A = 96$ в троичную систему счисления ($q_2 = 3$).

Решение $96 = 0 \cdot 3^5 + 1 \cdot 3^4 + 0 \cdot 3^3 + 1 \cdot 3^2 + 2 \cdot 3^1 + 0 \cdot 3^0 = 10120_3$.

Ответ $A_3 = 10120$

Рассмотреиный в примере 3.1 прием может быть использован только при ручном переводе. Для реализации машинных алгоритмов перевода применяют следующие методы.

Перевод целых чисел делением на основание q_2 новой системы счисления. Целое число A_{q_2} в системе с осиоваиием q_2 записывается в виде

$$A_{q_2} = b_k q_2^k + b_{k-1} q_2^{k-1} + \ldots + b_1 q_2^1 + b_0 q_2^0.$$

Переписав это выражеиие по схеме Горнера, получим

$$A_{q_2} = (\ldots((b_k q_2 + b_{k-1}) q_2 + \ldots) q_2 + b_1) q_2 + b_0. \qquad (3.13)$$

* Здесь и в дальнейшем при записи десятичных чисел индекс опускается.

Правую часть выражения (3.13) разделим на величину основания q_2. В результате определим первый остаток b_0 и целую часть $(...((b_k q_2+b_{k-1})q_2 + ...)q_2 + b_1)$. Разделив целую часть на q_2, найдем второй остаток b_1. Повторяя процесс деления $k+1$ раз, получим последнее целое частное b_k, которое, по условию, меньше основания системы q_2 и является старшей цифрой числа, представленного в системе с основанием q_2.

Пример 3.2. Перевести десятичное число $A = 98$ в двоичную систему счисления ($q_2 = 2$)

Решение.

98	2									98	0	b_0
98	49	2								49	1	b_1
$b_0 = 0$	48	24	2							24	0	b_2
	$b_1 = 1$	24	12	2						12	0	b_3
		$b_2 = 0$	12	6	2					6	0	b_4
			$b_3 = 0$	6	3	2				3	1	b_5
				$b_4 = 0$	2	$1 = b_6$				1	1	b_6
					$b_5 = 1$							

Ответ $A_2 = 1100010$.

Пример 3.3. Перевести двоичное число $A_2 = 1101001$ в десятичную систему счисления ($q_2 = 10$). Основание q_2 изображается в двоичной системе эквивалентом $q_2 = 1010_2$

Решение.

1101001	1010	
1010	1010	1010
001100	1010	$b_2 = 0001$
1010	$b_1 = 0000$	
$b_0 = 0101$		

Ответ: на основании таблицы 3.1 можно записать: $b_0 = 0101_2 = 5$; $b_1 = 0000_2 = 0$; $b_2 = 0001_2 = 1$; $A = 105$.

Этот способ применяют только для перевода целых чисел.

Перевод правильных дробей умножением на основание q_2 новой системы счисления. Пусть исходное число, записанное в системе счисления с основанием q_1, имеет вид

$$A_{q_1} = a_{-1}q_1^{-1} + ... + a_{-m}q_1^{-m}.$$

Тогда в новой системе с основанием q_2 это число будет изображено как 0, b_{-1}, ..., b_{-s}, или

$$A_{q_2} = b_{-1}q_2^{-1} + \ldots + b_{-s}q_2^{-s}.$$

Переписав это выражение по схеме Горнера, получим

$$A_{q_2} = q_2^{-1}(b_{-1} + q_2^{-1}(b_{-2} + \ldots + q_2^{-1}b_{-s})). \quad (3.14)$$

Если правую часть выражения (3.14) умножить на q_2, то получится новая неправильная дробь, в целой части которой будет число b_{-1}. Умножив затем оставшуюся дробную часть на величину основания q_2, получим дробь, в целой части которой будет b_{-2}, и т. д. Повторяя процесс умножения s раз, найдем все s цифр числа в новой системе счисления. При этом все действия должны выполняться по правилам q_1-арифметики, и следовательно, в целой части получающихся дробей будут проявляться эквиваленты цифр новой системы счисления, записанные в исходной системе счисления.

Пример 3.4. Перевести десятичную дробь $A = 0{,}625$ в двоичную систему счисления ($q_1 = 2$)

Решение

0,	×	625 2
$b_{-1} = 1$,	×	250 2
$b_{-2} = 0$,	×	500 2
$b_{-3} = 1$,	×	000 2
$b_{-4} = 0$.		000

Ответ $A_2 = 0{,}1010_2$.

Пример 3.5. Перевести двоичную дробь $A_2 = 0{,}1101_2$ в десятичную систему счисления ($q_2 = 1010_2$)

Решение

	0,	×	1101 1010
$b_{-1} = 8$	1000,	×	0010 1010
$b_{-2} = 1$	0001,	×	0100 1010
$b_{-3} = 2$	0010,	×	1000 1010
$b_{-4} = 5$	0101.		0000

Ответ $A = 0{,}8125$

При переводе правильных дробей из одной системы счисления в другую можно получить дробь в виде бесконечного или расходящегося ряда. Процесс перевода можно закончить, если появится дробная часть, имеющая во всех разрядах нули, или будет достигнута заданная точность перевода (получено требуемое количество разрядов результата). Последнее означает, что при переводе дроби необходимо указать количество разрядов числа в новой системе счисления. Естественно, что при этом возникает погрешность перевода чисел, которую надо оценивать.

Для перевода неправильных дробей из одной системы счисления в другую необходим раздельный перевод целой и дробной частей по правилам, описанным выше. Полученные результаты записывают в виде новой дроби в системе с основанием q_2.

Пример 3.6. Перевести десятичную дробь $A = 98{,}625$ в двоичную систему счисления ($q_2 = 2$).

Решение. Результаты перевода соответственно целой и дробной частей возьмем из примеров 3.2 и 3.4.

Ответ. $A_2 = 1100010{,}1010$.

Табличный метод перевода. В простейшем виде табличный метод заключается в следующем: имеется таблица всех чисел одной системы с соответствующими эквивалентами из другой системы; задача перевода сводится к нахождению соответствующей строки таблицы и выбору из нее эквивалента. Такая таблица очень громоздка и требует большой емкости памяти для хранения.

Другой вид табличного метода заключается в том, что имеются таблицы эквивалентов в каждой системе только для цифр этих систем и степеней основания (положительных и отрицательных); задача перевода сводится к тому, что в выражение ряда (3.3) для исходной системы счисления надо подставить эквиваленты из новой системы для всех цифр и степеней основания и произвести соответствующие действия (умножения и сложения) по правилам q_2-арифметики. Полученный результат этих действий будет изображать число в новой системе счисления.

Пример 3.7. Перевести десятичное число $A = 113$ в двоичную систему счисления, используя следующее соотношение эквивалентов и степени основания:

Десятичное число	10^0	10^1	10^2
Двоичный эквивалент	0001	1010	1100110

Решение. Подставив значения двоичных эквивалентов десятичных цифр и степеней основания в (3.11), получим

$$A = 113 = 1\cdot10^2 + 1\cdot10^1 + 3\cdot10^0 = 0001\cdot1100100 + 0001\cdot1010 + 0011\cdot0001 = 1110001_2 .$$

Ответ 1110001_2.

Пример 3.8. Перевести двоичное число $A_2 = 11001{,}1$ в десятичную систему счисления:

Двоичное число	0,1	00001	00010
Десятичный эквивалент	$2^{-1} = 0{,}5$	$2^0 = 1$	$2^1 = 2$
Двоичное число	00100	01000	10000
Десятичный эквивалент	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$

Решение. $A = 1\cdot16 + 1\cdot8 + 0\cdot4 + 0\cdot2 + 1\cdot1 + 1\cdot0{,}5 = 25{,}5$.

Ответ $A = 25{,}5$

Использование промежуточной системы счисления. Этот метод применяют при переводе из десятичной системы в двоичную и наоборот. В качестве промежуточной системы счисления можно использовать, например, восьмеричную систему.

Рассмотрим примеры, в которых перевод одного и того же числа в разные системы счисления осуществляется методом деления на основание новой системы. Запись будем вести в столбик, где справа от вертикальной черты записываются остатки деления на каждом шаге, а слева — целая часть частного.

Пример 3.9. Перевести десятичное число $A = 121$ в двоичную систему счисления, используя в качестве промежуточной восьмеричную систему счисления.

Решение.

$q_2 = 8$

121	1
15	7
1	1

3 шага

$q_2 = 2$

121	1
60	0
30	0
15	1
7	1
3	1
1	1

7 шагов

Ответ: $A = 121 = 171_8 = 1111001_2$.

Сравнивая эти примеры, видим, что при переводе числа из десятичной системы в восьмеричную требуется в два с лишним раза меньше шагов, чем при переводе в двоичную систему. Если при этом учесть, что восьмеричная

система связана с двоичной соотношением $8^k = (2^3)^k$, то перевод из восьмеричной системы в двоичную и наоборот можно осуществить простой заменой восьмеричных цифр их двоичными эквивалентами. Триада — двоичный эквивалент восьмеричных цифр.

Пример 3.10. Перевести двоичное число $A_2 = 1011{,}0111$ в восьмеричную систему счисления

Решение. Исходное число условно разбиваем на триады справа налево для целых чисел и слева направо для правильной дроби. Затем заменяем каждую триаду в соответствии с нижеприведенным соответствием.

Восьмеричная цифра....	0	1	2	3	4	5	6	7
Двоичный эквивалент..	000	001	010	011	100	101	110	111

$A_2 = 001 \quad 011{,}011 \quad 100.$

$A_8 = \quad 1 \quad\quad 3 \;,\; 3 \quad\quad 4 \;.$

Ответ $A = 13{,}34$.

В качестве промежуточных систем счисления целесообразно использовать системы с основанием $q = 2^k$. При этом существенно упрощается преобразование информации из системы счисления с основанием $q = 2^k$ в двоичную систему и наоборот. Преобразование фактически сводится к тому, что символы первоначальной информации, заданной в системе с основанием $q = 2^k$, заменяются соответствующими двоичными эквивалентами (см. табл. 3.1). Представление десятичных чисел в таком виде называется *десятично-двоичным*. Обратное преобразование из двоичной системы в систему с основанием $q = 2^k$ сводится к тому, что двоичный код разбивается на группы по k двоичных разрядов в каждой (начиная от младших разрядов для целых чисел или с первого разряда после запятой для правильных дробей); эти группы (диады, триады, тетрады (табл. 3.2) и т. д.) заменяются соответствующими символами исходной системы счисления.

Таблица 3.2

Десятичное число	Двоичный эквивалент для $q = 2^4$	Десятичное число	Двоичный эквивалент для $q = 2^4$	Десятичное число	Двоичный эквивалент для $q = 2^4$
0	0000	6	0110	11	1011
1	0001	7	0111	12	1100
2	0010	8	1000	13	1101
3	0011	9	1001	14	1110
4	0100	10	1010	15	1111
5	0101				

Системы счисления с основанием $q = 2^k$ широко используют для записи программ решения задач, а также для ускорения выполнения арифметических операций.

3.3. Разновидности двоичных систем счисления

Двоичная система счисления — система счисления, в которой для изображения чисел используются два символа и вес разрядов в которой меняется по закону $2^{\pm k}$ (где k — произвольное целое число). Из определения следует, что для изображения чисел могут быть использованы не только символы 0, 1, но и символы 1, –1 или 0, –1. Для удобства в дальнейшем символ –1 будем изображать как $\overline{1}$, а двоичную систему, в которой используются эти символы, называть системой $(1, \overline{1})$.

Рассмотрим такую систему. Если для ряда (3.3) положить, что a_i принимает значения 1 или $\overline{1}$, то в случае $q = 2$ число 99 запишется в виде $A = 99 = 1\,1\,1\,\overline{1}\,\overline{1}\,\overline{1}\,1_2$.

Система $(1, \overline{1})$ отличается от естественной двоичной системы тем, что среди используемых символов отсутствует нуль. Это обстоятельство делает невозможным представление в системе $(1, \overline{1})$ некоторых чисел в виде конечного множества. В то же время существуют числа, которые не имеют единственного изображения, например, число 1 может быть представлено в виде

$$1 = 1\underbrace{\overline{1}\,\overline{1}\ldots\overline{1}}_{k} = 1\underbrace{00\ldots0}_{k} - 1\underbrace{\ldots1}_{k}1 = 2^k - (2^k - 2^0), \qquad (3.15)$$

где $k = 1, 2, \ldots, n$.

Соотношение (3.15) выражает связь между естественной двоичной системой и системой $(1, \overline{1})$. Невозможность представить в виде конечного множества некоторые целые и дробные числа, например $20 = 1\,1\,\overline{1}\,1\,\overline{1},\overline{1}\,\overline{1}\,\overline{1}\ldots_2$, приводит к использованию бесконечных дробей, что обусловливает погрешность при представлении этих чисел в системе $(1, \overline{1})$.

Для получения конечного представления как четных, так и нечетных чисел в системе $(1, \overline{1})$ польским ученым И. Баньковским была предложена следующая запись чисел:

$$A_{1,\bar{1}} = \sum_{i=1}^{n} a \cdot 2^i - 2^{-1}, \qquad (3.16)$$

где $a_i = \{1, \bar{1}\}$.

Пример 3.11. Перевести число $A_2 = 100101$ в систему $(1, \bar{1})$.

Решение. Используя соотношение (3.15), перевод свести к замене комбинаций 001 и 01 комбинациями соответственно $1\bar{1}\bar{1}$ и $1\bar{1}$.

Ответ $A_{1\bar{1}} = 11\bar{1}\bar{1}1\bar{1}$.

Для перевода чисел в систему $(1, \bar{1})$ по методу Баньковского необходимо учитывать следующее: в случае нечетного числа перевод осуществляют по правилу (3.15), а затем в разряд $i = -1$ записывают единицу; при переводе четного числа его сначала превращают в нечетное добавлением единицы в младший разряд и только после этого переводят в систему $(1, \bar{1})$ по правилу (3.15) как нечетное число. Затем к полученному результату в разряд $i = -1$ записывают 1.

Пример 3.12. Перевести в систему $(1, \bar{1})$ двоичное число $A_2 = 11000$.

Решение. В соответствии с правилом Баньковского это число превращаем в нечетное число 11001. После этого заменяем в изображении числа комбинацию 001 на комбинацию $1\bar{1}\bar{1}$ и приписываем в разряд после запятой цифру 1.

Ответ $A_{1,\bar{1}} = 11\bar{1}\bar{1},1$.

Избыточная система счисления с основанием q — система, где для записи чисел используется количество символов, большее, чем q, например избыточная двоичная система с символами 0, 1, $\bar{1}$ или избыточная троичная система с символами 2, 1, 0, $\bar{1}$, $\bar{2}$.

Избыточная двоичная система связана с обычной двоичной системой соотношением

$$\underbrace{111\ldots1}_{k} = \sum_{i=1}^{k} 2^i = 2^{k+1} - 2^1 = \underbrace{100\ldots\bar{1}}_{k}. \qquad (3.17)$$

Формула (3.17) позволяет осуществлять переход от одной системы к другой, например $A = 11110001_2 = 1000\bar{1}0001_{1,0,\bar{1}}$.

В избыточной двоичной системе одни и те же числа можно представить несколькими способами, например $A = 0{,}01110011_2 = 0{,}100\bar{1}0011 = 0{,}100\bar{1}010\bar{1}$. Этот пример показывает, что при переходе к избыточной системе можно уменьшить количество единиц в изображении числа.

Избыточность системы счисления, характеризуемая симметричностью символов, дает возможность в ряде случаев упростить выполнение арифметических действий. Например, избыточную двоичную систему счисления используют в некоторых алгоритмах ускорения операции умножения (подробно она будет описана в § 5.7).

3.4. Системы счисления с отрицательным основанием

В работах К. Шеннона [19] было показано, что в системе счисления с основанием $q < -1$ и символами $0, -1, \ldots, -(q-1)$ можно представить любое действительное число, для чего используется выражение (3.3). Если некоторое число представляется в системе с целочисленным отрицательным основанием, то такое представление будет единственным для всех чисел, кроме чисел, равных X:

$$X = (\pm 1)q^k/(-q+1) + rq^{k+1},$$

где q — основание системы ($q < 0$); r и k — любые целые числа.

В самом деле, для системы счисления с основанием $q = -2$ десятичное число $X = 1/3$ может быть представлено бесконечными дробями в виде

$$X_{(-2)} = \begin{cases} 0, & 010101\ldots; \\ 1, & 101010. \end{cases}$$

Двоичную систему счисления с основанием $q = -2$, в которой используются символы 0, 1, назовем минус-двоичной системой счисления. В этой системе можно представлять как положительные, так и отрицательные числа (табл. 3.3).

Таблица 3.3

Десятичное число	Двоичный эквивалент для $q = 2$	Десятичное число	Двоичный эквивалент для $q = 2$
0	000000	−10	011110
1	000001	+15	010011
−1	000011	−15	010011
+5	000101	+21	010101
−5	001111	−21	111111
+10	001010		

Из таблицы видно, что методы перевода десятичных чисел в систему счисления с основанием $q = -2$ аналогичны методам перевода, рассмотренным ранее, но при переводе десятичных чисел необходимо учитывать следующее: при использовании метода последовательного деления на основание новой системы все остатки от деления на каждом шаге должны быть положительными числами, которые не превышают абсолютного значения нового основания q. Это правило распространяется и на случай перевода правильных дробей методом последовательного умножения на основе q, где появляющиеся целые части дробей также должны быть положительными числами, значение которых меньше величины q.

Пример 3.13. Перевести десятичное число $A = 21$ в минус-двоичную систему счисления методом деления на основание системы ($q = -2$).

Решение

21	1
−10	0
5	1
−2	0
1	1

Ответ: $A = 21 = 10101_{-2}$.

В случае перевода правильной дроби (или дробной части смешанной дроби) необходимо, чтобы дробь на каждом шаге удовлетворяла требованию

$$|q|/(|q|+1) \le A \le 1(|q|+1), \text{ или } -2/3 \le A \le 1/3, \qquad (3.18)$$

где A — дробь, переводимая в систему счисления с основанием $q = -2$.

Если ограничение (3.18) не выполняется, то дробь A представляют в виде $A = 1 - \Delta$, где Δ должно лежать в указанных пределах.

Пример 3.14. Перевести десятичную дробь $A = 0{,}625$ в минус-двоичную систему счисления с точностью до трех знаков после запятой.

Решение. Исходная дробь не удовлетворяет неравенству (3.18). Преобразуем ее к виду $0{,}625 = 1 - 0{,}375$.

(1) × −0,375
−2
0,750 неравенство (3.18) не удовлетворяется.

(1) × −0,250
−2
0,500 неравенство (3.18) не удовлетворяется.

(1) $\begin{array}{r} -0{,}500 \\ \times\ \ -2 \\ \hline 1{,}000 \end{array}$ неравенство (3.18) удовлетворяется.

Ответ $0{,}625 = 1{,}111_{-2}$.

Система счисления с отрицательным основанием может найти применение в специальных вычислительных устройствах, где отпадает необходимость иметь знаковую часть в изображении числа.

3.5. Формы представления числовой информации

Число 0,028 можно записать так: $28 \cdot 10^{-3}$, или 0,03 (с округлением), или $2{,}8 \cdot 10^{-2}$ и т. д. Разнообразие форм в записи одного числа может послужить причиной затруднений для работы цифрового автомата. Во избежание этого нужно либо создать специальные алгоритмы распознавания числа, либо указывать каждый раз форму его записи. Второй путь проще.

Существуют две формы записи чисел: естественная и нормальная. При естественной форме число записывается в естественном натуральном виде, например 12560 — целое число; 0,003572 — правильная дробь; 4,89760 — неправильная дробь. При нормальной форме запись одного числа может принимать разный вид в зависимости от ограничений, накладываемых на ее форму. Например, число 12560 может быть записано так: $12560 = 1{,}256 \cdot 10^4 = 0{,}1256 \cdot 10^5 = 125600 \cdot 10^{-1}$ и т. д.

Автоматное (*машинное*) *изображение числа* — представление числа A в разрядной сетке цифрового автомата. Условно обозначим автоматное изображение числа символом $[A]$. Тогда справедливо соотношение: $A = [A]K_A$, где K_A — коэффициент, величина которого зависит от формы представления числа в автомате.

Представление чисел с фиксированной запятой (точкой). Естественная форма представления числа в цифровом автомате характеризуется тем, что положение его разрядов в автоматном изображении остается всегда постоянным независимо от величины самого числа. Существует также другое название этой формы записи чисел — представление чисел с фиксированной запятой (точкой).

Чтобы упростить функционирование цифрового автомата, необходимо ограничить входную информацию какой-то одной областью чисел (на вход автомата желательно подавать либо целые числа, либо правильные дроби,

либо любые числа), что позволит определить значения масштабного коэффициента K_A. Например, если на вход цифрового автомата поступают только правильные дроби, то

$$-1 < [A]_ф < 1, \qquad (3.19)$$

где $[A]_ф$ — машинное изображение числа для формы представления с фиксированной запятой.

Тогда число A будет представлено в виде $A = [A]_ф K_A$.

Величина масштабного коэффициента K_A, удовлетворяющего условию (3.19), определяет тот факт, что в машинном изображении запятая всегда стоит после целой части дроби, т. е. перед ее старшим разрядом. Следовательно, можно хранить только дробную часть числа (цифровую часть), а в разряде целой части писать дополнительную информацию.

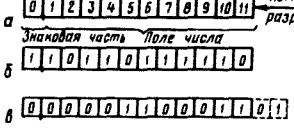


Рис. 3.2. Представление чисел в форме с фиксированной запятой

Так как числа бывают положительные и отрицательные, то формат (разрядная сетка) автоматного изображения разбивается на знаковую часть и поле числа (рис. 3.2, *а*). В знаковую часть записывается информация о знаке. Примем, что знак положительного числа «+» изображается символом 0, а знак отрицательного числа «–» изображается символом 1.

Если на вход цифрового автомата поступают целые числа, например, как в ЕС ЭВМ, то в разрядной сетке (в формате машинного изображения) один разряд отводится под знак числа, а последующие разряды образуют поле числа. Диапазон представимых чисел в этом случае от $-(2^n - 1)$ до $+(2^n - 1)$, где n — количество разрядов без знаковой части.

Задачу выбора масштабного коэффициента K_A усложняет необходимость сохранять соответствие разрядов всех чисел, которыми оперирует цифровой автомат. Пусть имеется цифровой автомат с разрядной сеткой длиной 12 двоичных разрядов (рис. 3.2, *а*). Надо определить масштабный коэффициент для чисел $A_1 = -1011{,}0111110_2$ и $A_2 = 0{,}110001101_2$.

Для того чтобы выполнить условие (3.19), необходимо число, большее по абсолютному значению, записать в виде $A_1 = -0{,}10110111110 \cdot 2^4$. Отсю-

да $[A_1]_ф = 1{,}10110111110$, что соответствует величине масштабного коэффициента $K_{A_1} = 2^4$. Число A_2 должно войти в разрядную сетку автомата с *сохранением соответствия разрядов*, т. е. $K_{A_2} = K_{A_1}$. Следовательно, $A_2 = +0{,}00001100011101 \cdot 2^4$ или $[A_2]_ф = 0{,}00001100011$ (рис. 3.2, *б*, *в*).

Из примера видно, что представление чисел в форме с фиксированной запятой может привести к погрешности представления. Так, для числа A_2 абсолютная погрешность представления оценивается величиной части числа, не уместившейся в разрядную сетку, т. е. величиной $0{,}0000000000001 \cdot 2^4$. В некоторых случаях очень малые числа представляются в машине изображением, называемым *машинным нулем*. Следовательно, ошибка представления зависит от правильности выбора масштабных коэффициентов. Вычисление последних должно проводиться таким образом, чтобы исключить возможность появления в процессе функционирования автомата чисел, машинные изображения которых не удовлетворяют условию (3.19). Если в результате операции появится число, по абсолютному значению большее единицы, то возникает переполнение разрядной сетки автомата, что нарушает нормальное функционирование цифрового автомата.

Представление чисел в форме с плавающей запятой. В нормальной форме

$$A_н = m_A q^{p_A}, \tag{3.20}$$

где m_A — мантисса числа A; p_A — порядок числа A.

Как видно из ранее изложенного, такое представление чисел не однозначно; для определенности обычно вводят некоторые ограничения. Наиболее распространено и удобно для представления в ЭВМ ограничение вида

$$q^{-1} \le |m_A| < 1, \tag{3.21}$$

где q - — основание системы счисления.

Нормализованная форма представления чисел — форма представления чисел, для которой справедливо условие (3.21).

Поскольку в этом случае абсолютное значение мантиссы лежит в пределах от q^{-1} до $1 - q^{-n}$, где n — количество разрядов для изображения мантиссы без знака, положение разрядов числа в его автоматном изображении не постоянно. Поэтому такую форму представления чисел называют также

формой представления с плавающей запятой. Формат машинного изображения числа с плавающей запятой должен содержать знаковые части и поля для мантиссы и порядка (рис. 3.3, *а*). Выделяются специальные разряды для изображения знака числа (мантиссы) и знака порядка или характеристики (рис. 3.3, *а, б*). Кодирование знаков остается таким же, как было с фиксированной запятой.

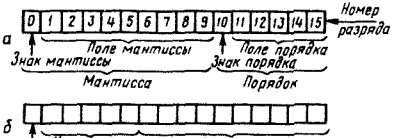


Рис. 3.3. Представление чисел в форме с плавающей запятой

Рассмотрим пример записи чисел в форме с плавающей запятой. Пусть в разрядную сетку цифрового автомата (рис. 3.3) необходимо записать двоичные числа $A_1 = -10110{,}1111_2$ и $A_2 = +0{,}000110010111_2$. Прежде всего эти числа необходимо записать в нормальной форме (рис. 3.3, *в, г*). Порядок чисел выбирают таким образом, чтобы для них выполнялось условие (3.21), т. е. $A_1 = -0{,}101101111 \cdot 2^5$ и $A_2 = +0{,}110010111 \cdot 2^{-3}$, он должен быть записан в двоичной системе счисления. Так как система счисления для заданного автомата остается постоянной, то нет необходимости указывать ее основание, достаточно лишь представить показатель степени.

Поскольку для изображения порядка выделено пять цифровых разрядов и один разряд для знака, их машинные изображения и машинные изображения их мантисс соответственно

$$[p_{A_1}] = 000101;\ [p_{A_2}] = 100011;$$

$$[m_{A_1}] = 1{,}101101111;\ [m_{A_2}] = 0{,}110010111.$$

3.6. Представленне отрицательных чисел

Один из способов выполнения операции вычитания с помощью сумматора — замена знака вычитаемого на противоположный н прибавление его к уменьшаемому:

$$A - B = A + (-B). \tag{3.22}$$

Этим операцию арифметического вычитания заменяют операцией алгебраического сложения, которая и становится основной операцией сумматора. Возникает вопрос: как представлять отрицательные числа в цифровом автомате?

Для машинного представления отрицательных чисел используют прямой, дополнительный и обратный коды. Рассмотрим применение этих кодов для чнсел, представленных в форме с фиксированной запятой. Для простоты изложения в дальнейшем будут рассматриваться числа, которые по модулю меньше единицы. Это существенно упрощает вычисление масштабных коэффициентов.

Прямой код числа $A = -0, a_1 a_2 \ldots a_n$ — машинное изображение этого числа в виде $[A]_{\text{пр}} = 1, a_1 a_2 \ldots a_n$. Из определения следует, что в прямом коде все цифровые разряды отрицательного числа остаются неизменными, а в знаковой части записывается единица. Например, если $A = -0{,}101110$, то $[A]_{\text{пр}} = 1{,}101110$. Положительное число в прямом коде не меняет своего изображения. Например, если $A = 0{,}110101$, то $[A]_{\text{пр}} = 0{,}110101$.

В прямом коде в разрядную сетку цифрового автомата можно записать следующее максимальное по абсолютному значению число $A_{\text{пр max}} = 0{,}11\ldots1 = 1 - 2^{-n}$, где n — количество разрядов разрядной сетки цифрового автомата.

Диапазон изменения машинных изображений для прямого кода лежит в пределах $-(1 - 2^{-n}) \le [A]_{\text{пр}} \le (1 - 2^{-n})$.

Ранее прямой код был использован для записи чисел в разрядной сетке цифрового автомата.

Правила преобразования чисел в прямой код можно сформулировать так:

$$[A]_{\text{пр}} = \begin{cases} A, \text{ если } A \ge 0, \\ 1 + |A|, \text{ если } A \le 0. \end{cases} \tag{3.23}$$

Дополнительный код числа $A = -0, a_1 a_2 \ldots a_n$ — такое машинное изображение этого числа $[A]_д = 1, \bar{a}_1 \bar{a}_2 \ldots \bar{a}_n$, для которого $\bar{a}_i = 0$ при $a_i = 1$, и $\bar{a}_i = 1$ при $a_i = 0$, за исключением последнего значащего разряда, для которого $\bar{a}_k = 1$ при $a_k = 1$. Например, число $A = -0{,}101110$ запишется в дополнительном коде так: $[A]_д = 1{,}010010$.

Дополнительный код является математическим дополнением основанию системы счисления:

$$|A| + [A]_д = q, \tag{3.24}$$

где $|A|$ — абсолютное значение числа A.

Так как положительные числа не меняют своего изображения в дополнительном коде, то правила преобразования в дополнительный код можно записать следующим образом:

$$[A]_д = \begin{cases} A, \text{ если } A \geq 0, \\ q - |A|, \text{ если } A < 0. \end{cases} \tag{3.25}$$

Максимальное дополнительное число, представляемое при этом, равно $(1 - 2^{-n})$.

Наибольшее отрицательное число, которое можно записать в дополнительном коде, определим следующим образом. Предположим, что наибольшее отрицательное число $A^1 = -0{,}11\ldots11$. Тогда изображение этого числа в дополнительном коде $[A^1]_д = 1{,}00\ldots01$. Если к числу A^1 добавить единицу в самый младший разряд, то в результате получим число $-1{,}00\ldots0$. Преобразовав это число по формальным правилам, получим $[A]_{д\,min} = 1{,}00\ldots0$.

Следовательно, диапазон изменения машинных изображений чисел для формы представления с запятой, фиксированной перед старшим разрядом, в дополнительном коде $-1 \leq |A|_д \leq (1 - 2^{-n})$.

Машинные изображения чисел — всегда целые числа. При этом наибольшее положительное число состоит из целой части, все разряды которой равны единице, и знакового разряда, равного нулю (например, в случае 16 двоичных разрядов (два байта) максимальное положительное число имеет вид 0111111111111111, т. е. равно $2^{15} - 1$); наибольшее отрицательное число состоит из целой части, все разряды которой равны нулю, и знакового

разряда, равного единице, т. е. имеет вид 1000000000000000. В этом случае говорят о *форме представления чисел с фиксированной точкой*. Таким образом, соотношение (3.24) для представления целых чисел в дополнительном коде принимает вид

$$|A| + [A]_д = q^{k+1}, \tag{3.26}$$

где k — количество разрядов в целой части машинного изображения числа ($k = 0, \bar{k}$).

Формула (3.24) — частный случай формулы (3.26) при $k = 0$.

Обратный код числа $A = -0, a_1 a_2 \ldots a_n$ — такое машинное изображение этого числа $[A]_{об} = 1, \bar{\bar{a}}_1 \bar{\bar{a}}_2 \ldots \bar{\bar{a}}_n$, для которого $\bar{\bar{a}}_i = 0$, если $a_i = 1$, и $\bar{\bar{a}}_i = 1$, если $a_i = 0$. Из определения следует, что обратный код двоичного числа является инверсным изображением самого числа, в котором все разряды исходного числа принимают инверсное (обратное) значение, т. е. все нули заменяются на единицы, а все единицы — на нули. Например, если $A = -0{,}101110$, то $[A]_{об} = 1{,}010001$. Для обратного кода чисел, представленных в форме запятой, фиксированной перед старшим разрядом, справедливо соотношение

$$|A| + |A|_{об} = q - q^{-n}, \tag{3.27}$$

где $|A|$ — абсолютная величина A; n — количество разрядов после запятой в изображении числа.

Правила преобразования чисел в обратный код можно сформулировать следующим образом:

$$[A]_{об} = \begin{cases} A, \text{ если } A \geq 0, \\ q - q^{-n} + A, \text{ если } A \leq 0. \end{cases} \tag{3.28}$$

Сравнив (3.24) и (3.27), видим, что

$$|A|_д = [A]_{об} + q^{-n}. \tag{3.29}$$

Соотношение (3.29) используют для получения дополнительного кода отрицательных чисел следующим образом: сначала инвертируется цифровая часть исходного числа, в результате получается его обратный код; затем добавляется единица в младший разряд цифровой части числа и тем самым получается дополнительный код этого изображения.

Пример 3.15. Найти обратный и дополнительный коды числа $A = -0{,}111000_2$.

Р е ш е н и е . Используя определение обратного кода, получим $[A]_{об} = 1{,}000111$.

Для нахождения дополнительного кода числа добавим единицу в младший разряд его изображения:

$$\begin{array}{r} 1{,}000111 \\ + \\ \hline 1 \\ \hline [A]_д = 1{,}001000 \end{array}$$

Ответ: $[A]_{об} = 1{,}000111$; $[A]_д = 1{,}001000$.

В обратном коде можно изображать максимальное положительное число $[A]_{об\,max} = 0{,}11\ldots1 = 1 - 2^{-n}$ и наибольшее отрицательное число $[A]_{об\,min} = -0{,}11\ldots1 = -(1 - 2^{-n})$, записываемое в виде $1{,}00\ldots0$.

При проектировании цифровых автоматов необходимо учитывать неоднозначное изображение нуля в обратном коде: +0 изображается $0{,}00\ldots0$, –0 изображается $1{,}11\ldots1$.

Использование различных способов изображения отрицательных чисел в цифровом автомате обусловливает целый ряд особенностей выполнения операции алгебраического сложения двоичных чисел.

3.7. Погрешности представления числовой информации

Представление числовой информации в цифровом автомате, как правило, влечет за собой появление погрешностей (ошибок), величина которых зависит от формы представления чисел и от длины разрядной сетки автомата.

Абсолютная погрешность представления — разность между истинным значением входной величины A и ее значением, полученным из машинного изображения $A_м$, т. е. $\Delta[A] = A - A_м$.

Относительная погрешность представления — величина

$$\delta[A] = \Delta[A]/A_м . \qquad (3.30)$$

Входные величины независимо от количества значащих цифр могут содержать грубые ошибки, возникающие из-за опечаток, ошибочных отсчетов показаний каких-либо приборов, некорректной постановки задачи или отсутствия более полной и точной информации. Например, часто принимают $\pi = 3{,}14$. Однако эта величина может быть получена с более высокой точностью. Если принять, что точное значение $\pi = 3{,}14139265$, то абсолютная погрешность равна $\Delta[\pi] = 0{,}00159265$.

Часто некоторая величина в одной системе счисления имеет конечное значение, а в другой системе счисления становится бесконечной величиной, например, дробь 1/10 имеет конечное десятичное представление, но, будучи переведена в двоичную систему счисления, становится бесконечной дробью 0,00011000110011...

Следовательно, при переводе чисел из одной системы счисления в другую неизбежно возникают погрешности, оценить которые нетрудно, если известны истинные значения входных чисел.

В соответствии с (3.19) числа изображаются в машине в виде $A_q = [A]K_A$, где масштабный коэффициент K_A выбирают так, чтобы абсолютное значение машинного изображения числа A в системе счисления с основанием $q = 2$ было всегда меньше 1: $A_q = K_A[a_{-1}q^{-1} + a_{-2}q^{-2} + \ldots + a_{-n}q^{-n} + \ldots]$.

Так как длина разрядной сетки автомата равна n двоичных разрядов после запятой, абсолютная погрешность перевода десятичной информации в систему с основанием q будет

$$\Delta[A] = a_{-(n+1)}q^{-(n+1)} + \ldots + a_{-(n+s)}q^{-(n+s)} + \ldots = \sum_{i=-(n+1)}^{-\infty} a_i q^i . \qquad (3.31)$$

Если $q = 2$, то при $a_i = 1$ максимальное значение этой погрешности

$$\Delta[A]_{\max} = \sum_{i=-(n+1)}^{\infty} 1 \cdot 2^i = 2^{-n}\sum_{i=-1}^{-\infty} 2^i = 2^{-n} . \qquad (3.32)$$

Из (3.32) следует, что максимальная погрешность перевода десятичной информации в двоичную не будет превышать единицы младшего разряда разрядной сетки автомата. Минимальная погрешность перевода равна нулю.

Усредненная абсолютная погрешность перевода чисел в двоичную систему счисления $\Delta[A] = (0 + 2^{-n})/2 = 0{,}5 \cdot 2^{-n}$.

Для представления чисел в форме с фиксированной запятой абсолютное значение машинного изображения числа

$$2^{-n} \le |[A]_\phi| \le 1 - 2^{-n} . \qquad (3.33)$$

Следовательно, относительные погрешности представления для минимального значения числа $\delta[A]_{\phi\,\min} = \Delta[A]/[A]_{\phi\,\max} = 0{,}5 \cdot 2^{-n}/(1 - 2^{-n})$.

Для ЭВМ, как правило, $n = 16 \ldots 64$, поэтому $1 >> 2^{-n}$, откуда $\delta[A]_{\phi\,\min} = 0{,}5 \cdot 2^{-n}$.

Аналогично, для максимального значения:

$$\delta[A]_{\text{ф max}} = \Delta[A]_{\text{ф max}} / [A]_{\text{ф min}} = 0{,}5 \cdot 2^{-n} / 2^{-n} . \qquad (3.34)$$

Из (3.34) видно, что погрешности представления малых чисел в форме с фиксированной запятой могут быть очень значительными.

Для представления чисел в форме с плавающей запятой абсолютное значение мантиссы

$$2^{-1} \le |[mA]_n| \le 1 - 2^{-n} . \qquad (3.35)$$

Погрешность (3.32) — погрешность мантиссы. Для нахождения погрешности представления числа в форме с плавающей запятой величину этой погрешности надо умножить на величину порядка числа p_A:

$$\begin{aligned} \delta[A]_{n\,\text{max}} &= 0{,}5 \cdot 2^{-n} p_A / 2^{-1} \cdot p_A = 2^{-n}; \\ \delta[A]_{n\,\text{min}} &= 0{,}5 \cdot 2^{-n} p_A / (1 - 2^{-n})\, p_A, \end{aligned} \qquad (3.36)$$

где n — количество разрядов для представления мантиссы числа.

Из (3.36) следует, что относительная точность представления чисел в форме с плавающей запятой почти не зависит от величины числа.

Задание для самоконтроля

1. Перевести десятичное число $A = 121$ в двоичную систему счисления.

2. Перевести двоичное число $A = 1000101011{,}01$ в десятичную систему счисления.

3. Перевести десятичное число $A = 135{,}656$ в двоичную систему счисления с точностью до пяти знаков после запятой.

4. Сколько потребуется двоичных разрядов для изображения десятичного числа $A = 10^{18}$?

5. Перевести двоичное число $A_2 = 10111011$ в десятичную систему счисления методом деления на основание.

6. Перевести восьмеричное число $A_8 = 345{,}766$ в двоичную систему счисления.

7. Записать десятичное число $A = 79{,}346$ в двоично-десятичной форме.

8. Перевести десятичную дробь $A = 63/64$ в двоичную систему счисления.

9. Перевести восьмеричную дробь $A_8 = 63/40$ в двоичную систему счисления.

10. Перевести восьмеричное число $A_8 = 326$ в троичную систему счисления.

11. Перевести восьмеричное число $A_8 = 15{,}647$ в двоичную систему счисления.

12. Перевести троичное число $A_3 = 1211$ в пятеричную систему счисления.

13. Для какой системы счисления с основанием $q_2 = x$ справедливо равенство $121 = 441_x$?

14. Записать машинное изображение в форме с плавающей запятой для десятичного числа $A \approx -3{,}375$, если для мантиссы имеется шесть двоичных разрядов со знаком и для порядков — три двоичных разряда (со знаком).

15. Определить масштабные коэффициенты для чисел $A_2 = -10110{,}111010001$ и $B_2 = 0{,}001110001110001$ при условии, что машинное изображение числа содержит десять двоичных разрядов со знаком.

16. Перевести двоичное число $A_2 = 0{,}011000100$ в систему $(1, \bar{1})$.

17. Перевести число $A_{1\bar{1}} = 1111{,}1$ из системы $(1, \bar{1})$ в двоичную систему счисления.

4. АЛГОРИТМЫ ВЫПОЛНЕНИЯ ОПЕРАЦИЙ СЛОЖЕНИЯ И ВЫЧИТАНИЯ ЧИСЕЛ НА ДВОИЧНЫХ СУММАТОРАХ

4.1. Формальные правила двоичной арифметики

Популярность двоичной системы счисления во многом определяется простотой выполнения арифметических действий:

сложение	вычитание	умножение
0 + 0 = 0	0 − 0 = 0	0 × 0 = 0
0 + 1 = 1	1 − 0 = 1	0 × 1 = 0
1 + 0 = 1	1 − 1 = 0	1 × 0 = 0
1 + 1 = (1) 0	0 − 1 = (1) 1	1 × 1 = 1
перенос в старший разряд	заем в старшем разряде	

В основу арифметико-логического устройства любой ЭВМ может быть положен либо сумматор, либо вычитатель. И в том и в другом случае могут быть разработаны алгоритмы выполнения арифметических операций. В выполнении арифметических действий всегда участвуют два числа или более. В результате арифметической операции появляется новое число

$$C = A \nabla B, \tag{4.1}$$

где ∇ — знак арифметического действия (сложение, вычитание, умножение, деление).

Операнд — число, участвующее в арифметической операции, выполняемой цифровым автоматом.

Так как цифровой автомат оперирует только автоматными изображениями чисел, то последние выступают в качестве операндов. Следовательно, для машинных операций более правильно выражение (4.1) написать в виде

$$[C] = [A] \nabla [B], \tag{4.2}$$

где в квадратных скобках [ ] — обозначения автоматных изображений операндов.

Рассмотрим формальные правила выполнения арифметических операций сложения и вычитания на уровне разрядов операндов.

На основе правил двоичной арифметики можно записать правила сложения двоичных цифр так, как показано в таблице 4.1, где a_i, b_i — разряды операндов A и B соответственно; c_i — разряд суммы; $п_i$ — перенос из данного разряда в соседний старший.

Двоичный полусумматор — устройство, выполняющее арифметические действия по правилам, указанным в таблице 4.1.

Таблица 4.1

a_i	b_i	c_i	$п_i$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Таблица 4.2

a_i	b_i	$п_{i-1}$	c_i	$п_i$
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Появление единицы переноса при сложении двух разрядов несколько изменяет правила сложения двоичных цифр (табл. 4.2).

Обобщая вышеизложенное, можно сформулировать правила поразрядных действий при сложении операндов A и B:

$$a_i + b_i + п_{i-1} = c_i + п_i, \qquad (4.3)$$

где $п_{i-1}$ — перенос из $(i-1)$-го разряда; $п_i$ — перенос в $(i+1)$-й разряд (переносы принимают значения 0 или 1).

Двоичный сумматор — устройство, выполняющее арифметические действия по правилам, указанным в таблице 4.2. Условные обозначения двоичных полусумматоров и сумматоров показаны на рис. 4.1, *а* и *б* соответственно.

На основе правил двоичной арифметики можно записать правила вычитания двоичных цифр так, как показано на таблице 4.3, где z_{i+1} — заем в старшем разряде.

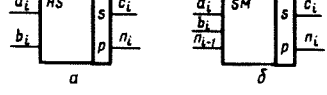


Рис. 4.1. Условное обозначение полусумматора и двоичного сумматора

Таблица 4.3

a_i	b_i	c_i	z_{i+1}
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	–1

Таблица 4.4

a_i	b_i	z_i	c_i	z_{i+1}
0	0	0	0	0
1	0	0	1	0
1	1	0	0	0
0	1	0	1	–1
0	0	–1	1	–1
1	0	–1	0	0
1	1	–1	1	1
0	1	–1	0	–1

Заем равносилен вычитанию единицы из старшего разряда. С учетом единицы заема из старшего соседнего разряда правила вычитания двоичных цифр можно записать так, как показано в таблице 4.4 (чтобы отличить заем от переноса, перед единицей поставлен знак минус).

Если A — уменьшаемое (1-й операнд), B — вычитаемое (2-й операнд), то для поразрядных действий

$$a_i - b_i + z_i = c_i + z_{i+1}. \qquad (4.4)$$

Двоичный вычитатель — устройство, выполняющее арифметическое действие по правилам, указанным в таблице 4.4.

С точки зрения технической реализации всегда проще сложить два электрических сигнала, чем вычесть их друг из друга.

В машине «Иллиак» был использован вычитатель кодов и метод избыточного кодирования чисел. При этом любой разряд числа представляется двумя битами, один из которых является знаковым s_i, а другой — значащим a_i.

Таблица кодирования выглядит следующим образом (табл. 4.5).

Таблица 4.5

Значение разряда	Кодирование	
	s_i	a_i
+0	0	0
+1	0	1
–0	1	0
–1	1	1

Вместо сложения используется операция вычитания (условное обозначение вычитателя показано на рис. 4.2).

На рис. 4.2 условно обозначены: s_i, a_i — знаковый и значащий разряды уменьшаемого соответственно; b_i — двоичный разряд вычитаемого; $п_i$, $п_{i-1}$ — нераспространяющийся перенос; NEG — управление дополнением c_i; G — управление цепями переноса; z_i, c_i — знаковый и значащий разряды результата (разностн).

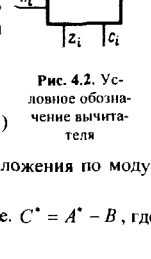


Рис. 4.2. Условное обозначение вычитателя

При этом

$$\left.\begin{aligned} z_i &= п_i \oplus NEG, c_i = п_i \oplus a_i \oplus b_i, \\ п_{i-1} &= (s_i a_i \vee a_i b_i)G, п_i = (s_{i+1}a_{i+1} \vee a_{i+1}b_{i+1})G. \end{aligned}\right\} \quad (4.5)$$

Символ $\oplus$ принят для обозначения поразрядного сложения по модулю 2 (см. § 2.3).

Результат всегда получается в избыточной форме, т. е. $C^* = A^* - B$, где C^*, A^* — избыточное представление.

Тогда операцию сложения можно осуществить путем инверсирования числа A^* и снятия результата при $NEG = 1$, т. е.

$$C^* = -(-A^* - B).$$

В подавляющем большинстве ЭВМ основное устройство — двоичный или десятичный сумматор, в зависимости от принятой системы счисления.

4.2. Сложение чисел, представленных в форме с фиксированной запятой, на двоичных сумматорах

Рассмотрим несколько видов двоичных сумматоров.

Двоичный сумматор прямого кода (ДСПК) — сумматор, в котором отсутствует цепь поразрядного переноса между старшим цифровым и знаковым разрядами (рис. 4.3, *а*). На ДСПК можно складывать только числа, имеющие одинаковые знаки, т. е. такой сумматор не может выполнять операцию алгебраического сложения. В самом деле, пусть заданы операнды

$$[A]_{пр} = Sg_A a_1 a_2 \ldots a_n, [B]_{пр} = Sg_B b_1 b_2 \ldots b_n,$$

где Sg_A, Sg_B соответственно содержимое знаковых разрядов изображений для A и B (символ происходит от английского слова *sign* — знак); a_i, b_i — цифровые разряды изображений.

Если $Sg_A = Sg_B$, то сумма чисел будет иметь знак любого из слагаемых, а цифровая часть результата получится после сложения цифровых частей операндов.

Пример 4.1. Сложить числа $A = 0{,}1011$, $B = 0{,}0100$ на сумматоре прямого кода

Решение.

$$\begin{array}{ll} [A]_{\text{пр}} = 0{,}1011 & Sg_A = 0, \\ +\ [B]_{\text{пр}} = 0{,}0100 & Sg_B = 0, \\ \hline [C]_{\text{пр}} = 0{,}1111 & Sg_C = 0. \end{array}$$

Ответ: $[C]_{\text{пр}} = 0{,}1111$.

Пример 4.2. Сложить числа $A = -0{,}0101$, $B = -0{,}1001$ на сумматоре прямого кода

Решение.

$$\begin{array}{lll} [A]_{\text{пр}} = 1{,}0101; & Sg_A = 1 & ,0101 \\ [B]_{\text{пр}} = 1{,}1001; & Sg_B = 1 & ,1001 \\ \hline & Sg_C = 1 & ,1110 \end{array}$$

Ответ. $[C]_{\text{пр}} = 1{,}1110$

При сложении чисел на ДСПК возможен случай, когда абсолютное значение суммы операндов превышает единицу. Тогда имеет место переполнение разрядной сетки автомата. Признак переполнения — наличие единицы переноса из старшего разряда *цифровой части сумматора*. В этом случае должен вырабатываться сигнал переполнения $\varphi = 1$, по которому происходит автоматический останов машины и корректировка масштабных коэффициентов с таким расчетом, чтобы избежать появления переполнения.

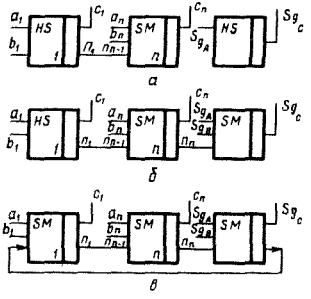


Рис. 4.3. Структурные схемы двоичных сумматоров на n разрядов

Двоичный сумматор дополнительного кода (ДСДК) — сумматор, оперирующий изображениями чисел в дополнительном коде. Характерная особенность ДСДК — наличие цепи поразрядного переноса из старшего разряда цифровой части в знаковый разряд (рис. 4.3, *б*). Определим правила сложения чисел на ДСДК.

Теорема. *Сумма дополнительных кодов чисел есть дополнительный код результата.*

Доказательство. Предположим, что числа представлены в форме с фиксированной запятой, стоящей перед старшим разрядом. Рассмотрим возможные случаи.

$$1)\ A>0,\ B>0,\ A+B<1.$$

Так как $[A]_д = A$, $[B]_д = B$, то $[A]_д + [B]_д = A + B = [A+B]_д$ — результат положительный.

$$2)\ A<0,\ B>0,\ |A|>B.$$

Здесь $[A]_д = A+q$, $[B]_д = B$. Тогда $[A]_д + [B]_д = A+B+q$ — результат отрицательный.

$$3)\ A<0,\ B>0,\ |A|<B.$$

Здесь $[A]_д = A+q$, $[B]_д = B+q$. Тогда $[A]_д + [B]_д = A+B+q$. Так как значение этой суммы больше q, появляется единица переноса из знакового разряда, что равносильно изъятию из суммы q единиц, т. е. результат равен $[A]_д + [B]_д = A+B$.

$$4)\ A<0,\ B<0,\ |A+B|<1.$$

Здесь $[A]_д = A+q$, $[B]_д = B+q$. Тогда $[A]_д + [B]_д = A+B+q+q = [A+B]_д$ — результат отрицательный (здесь появляется единица переноса из знакового разряда).

Таким образом, теорема справедлива для всех случаев, в которых не возникает переполнение разрядной сетки, что позволяет складывать автоматные изображения чисел по правилам двоичной арифметики (см. табл. 4.2), не разделяя знаковую и цифровую части изображений.

Пример 4.3. Найти сумму чисел $A = 0{,}1010$, $B = 0{,}0100$, используя сумматор дополнительного кода

Р е ш е н и е. Складываются машинные изображения этих чисел:

$$\begin{array}{l} [A]_д = 0{,}1010 \\ \underset{+}{[B]_д = 0{,}0100} \\ \hline [C]_д = 0{,}1110 \end{array}$$

Ответ. $C = 0{,}1110$

Пример 4.4. Найти сумму чисел $A = -0{,}1011$, $B = 0{,}0100$ на сумматоре дополнительного кода

Решение.

$$\begin{array}{l}[A]_д = 1{,}0101\\ +\\ [B]_д = 0{,}0100\\ \hline [C]_д = 1{,}1001\end{array}$$

Ответ $C = -0{,}0111$.

Пример 4.5. Найти сумму чисел $A = 0{,}1011$, $B = -0{,}0100$ на сумматоре дополнительного кода.

Решение.

$$\begin{array}{l}[A]_д = 0{,}1011\\ +\\ [B]_д = 1{,}1100\\ \hline [C]_д = 0{,}0111\end{array}$$

Ответ $C = -0{,}0111$.

Двоичный сумматор обратного кода (ДСОК) — сумматор, оперирующий изображениями чисел в обратном коде. Характерная особенность ДСОК — наличие цепи кругового, или циклического, переноса из знакового разряда в младший разряд цифровой части (рис. 4.3, *в*).

Определим правила сложения чисел на ДСОК.

Теорема. *Сумма обратных кодов чисел есть обратный код результата.*

Доказательство. Рассмотрим следующие основные случаи:

1) $A > 0$, $B > 0$, $A + B < 1$. Тогда $[A]_{об} + [B]_{об} = A + B = [A + B]_{об}$.

2) $A < 0$, $B > 0$, $|A| > B$. Здесь $[A]_{об} = q - q^{-n} + A$, $[B]_{об} = B$.

Тогда $[A]_{об} + [B]_{об} = q - q^{-n} + A + B = [A + B]_{об}$, так как результат отрицательный.

3) $A < 0$, $B > 0$, $|A| < B$. Здесь $[A]_{об} = q - q^{-n} + A$.

Тогда $[A]_{об} + [B]_{об} = q - q^{-n} + A + B$. Так как $[A]_{об} + [B]_{об}$ положительна, правая часть этого выражения становится больше q, что вызывает появление единицы переноса из знакового разряда. Поскольку в ДСОК существует цепь переноса из знакового разряда в младший разряд (величина переноса из знакового разряда равна $q - q^{-n}$), то $[A]_{об} + [B]_{об} = [A + B]_{об}$, результат положительный.

4) $A < 0$, $B < 0$, $|A + B| < 1$. Здесь $[A]_{об} = q - q^{-n} + A$,

$$[B]_{об} = q - q^{-n} + B.$$

Следовательно, $[A]_{об} + [B]_{об} = q - q^{-n} + q - q^{-n} + A + B$. Здесь появляется единица переноса из знакового разряда, что равносильно изъятию из суммы величины $q - q^{-n}$, т. е. $[A]_{об} + [B]_{об} = [A + B]_{об}$.

5) $|A| = B$, $A < 0$, $B > 0$. Тогда $[A]_{об} = q - q^{-n} + A$.

Следовательно, $[A]_{об} + [B]_{об} = q - q^{-n} + A + B = q - q^{-n}$ — результат указывает на то, что сумма равна нулю (получим одно из изображений нуля в обратном коде).

Таким образом доказано, что на ДСОК машинные изображения чисел также складываются по правилам, приведенным в таблице 4.2.

Пример 4.6. Найти сумму чисел $A = 0{,}0101$ и $B = 0{,}0111$, используя сумматор обратного кода.

Решение.

$$\begin{array}{l} [A]_{об} = 0{,}0101 \\ + \\ [B]_{об} = 0{,}0111 \\ \hline [C]_{об} = 0{,}1100 \end{array}$$

Ответ $C = 0{,}1100$

Пример 4.7. Найти сумму чисел $A = -0{,}0101$ и $B = 0{,}0111$, используя ДСОК.

Решение.

$$\begin{array}{l} [A]_{об} = 1{,}1010 \\ + \\ [B]_{об} = 0{,}0111 \\ \hline + \quad 0{,}0001 \\ \quad\quad\quad 1 \\ \hline [C]_{об} = 0{,}0010 \end{array}$$

Ответ $C = 0{,}0010$.

Пример 4.8. Найти сумму чисел $A = 0{,}0101$ и $B = -0{,}0111$, используя ДСОК.

Решение.

$$\begin{array}{l} [A]_{об} = 0{,}0101 \\ + \\ [B]_{об} = 1{,}1000 \\ \hline [C]_{об} = 1{,}1101 \end{array}$$

Ответ $C = -0{,}0010$.

Пример 4.9. Найти сумму чисел $A = -0{,}0101$ и $B = -0{,}1000$, используя ДСОК.

Решение.

$$[A]_{об} = 1{,}1010$$
$$+\ [B]_{об} = 1{,}0111$$
$$+\ 1{,}0001$$
$$[C]_{об} = 1{,}0010$$

Ответ $C = -0{,}1101$.

В дальнейшем для упрощения записи передача циклического переноса будет осуществляться сразу при получении результата и отдельно фиксироваться не будет.

4.3. Переполнение разрядной сетки

При сложении чисел одинакового знака, представленных в форме с фиксированной запятой, может возникнуть переполнение разрядной сетки.

I. Признак переполнения разрядной сетки сумматора прямого кода — появление единицы переноса из старшего разряда цифровой части числа.

Пример 4.10.

1*а*. $A = 0{,}1010$ и $B = 0{,}1101$.

$$[A]_{пр} = 0{,}1010$$
$$+\ [B]_{пр} = 0{,}1101$$
$$[C]_{пр} \neq 0{,}0111$$

1 ← единица переноса

2*а*. $A = -0{,}1100$ и $B = -0{,}1010$

$$[A]_{пр} = 1{,}1100$$
$$+\ [B]_{пр} = 1{,}1010$$
$$[C]_{пр} \neq 1{,}0110$$

1 ← единица переноса

II. Признак переполнения разрядной сетки сумматора дополнительного кода при сложении положительных чисел — отрицательный знак результата, а при сложении отрицательных чисел — положительный знак результата.

3*а*. $A = 0{,}1011$ и $B = 0{,}1010$

$$[A]_{д} = 0{,}1011$$
$$+\ [B]_{д} = 0{,}1010$$
$$[C]_{д} \neq 1{,}0101$$

4*а*. $A = -0{,}1011$ и $B = -0{,}1001$

$$[A]_{д} = 1{,}0101$$
$$+\ [B]_{д} = 1{,}0111$$
$$[C]_{д} \neq 0{,}1100$$

III. Признак переполнения разрядной сетки сумматора обратного кода — знак результата, противоположный знакам операндов.

5*а*. $A = 0{,}0111$ и $B = 0{,}1101$.

$$\begin{array}{l} [A]_{об} = 0{,}0111 \\ \underset{+}{[B]_{об}} = 0{,}1101 \\ \hline [C]_{об} \neq 1{,}0100 \end{array}$$

6*а*. $A = -0{,}0110$ и $B = -0{,}1101$.

$$\begin{array}{l} [A]_{об} = 1{,}1001 \\ \underset{+}{[B]_{об}} = 1{,}0010 \\ \hline [C]_{об} \neq 0{,}1011 \end{array}$$

Для обнаружения переполнения разрядной сетки в составе цифрового автомата должны быть предусмотрены аппаратные средства, автоматически вырабатывающие признак переполнения — сигнал φ.

Чтобы обнаружить переполнение разрядной сетки ДСОК и ДСДК, вводится вспомогательный разряд в знаковую часть изображения числа (рис. 4.4, *а*), который называют разрядом переполнения. На рис. 4.4, *б*, *в* соответственно представлены изображения положительного и отрицательного чисел. Такое представление числа называется модифицированным. Тогда в случае появления переполнения сигнал $\varphi = 1$:

$$Sg_1 \wedge \overline{S}g_2 = 1,\ \overline{S}g_1 \wedge Sg_2 = 1; \qquad (4.6)^*$$

в остальных случаях $\varphi = 0$.

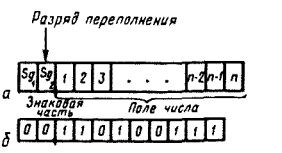


Рис. 4.4. Представление чисел в модифицированном коде

Это подтверждается следующими примерами.

3*б* $[A]^{м}_{п} = 00{,}1011$ — модифицированное изображение операнда A;
$\underset{+}{[B]^{м}_{п}} = 00{,}1010$ — модифицированное изображение операнда B;
$[C]^{м}_{п} = 01{,}0101$ — 01 — признак переполнения в знаковых разрядах, $\varphi = 1$;

4*б* $[A]^{м}_{п} = 11{,}0101$ — модифицированное изображение операнда A;
$\underset{+}{[B]^{м}_{п}} = 11{,}0111$ — модифицированное изображение операнда B;
$[C]^{м}_{п} = 10{,}1100$ — 10 — признак переполнения в знаковых разрядах, $\varphi = 1$;

5*б* $[A]^{м}_{об} = 00{,}0111$ — модифицированное изображение операнда A;
$\underset{+}{[B]^{м}_{об}} = 00{,}1101$ — модифицированное изображение операнда B;
$[C]^{м}_{об} = 01{,}0100$ — 01 — признак переполнения в знаковых разрядах, $\varphi = 1$;

6*б* $[A]^{м}_{об} = 11{,}1001$ — модифицированное изображение операнда A;
$\underset{+}{[B]^{м}_{об}} = 11{,}0010$ — модифицированное изображение операнда B;
$[C]^{м}_{об} = 10{,}1100$ — 10 — признак переполнения в знаковых разрядах, $\varphi = 1$.

---

* Здесь и в дальнейшем тексте символ $\wedge$ означает логическую функцию И, а черта над символом $\overline{Sg}$ — функцию НЕ (см § 10.1).

4.4. Особенности сложения чисел, представленных в форме с плавающей запятой

Числа, представленные в форме с плавающей запятой, изображаются двумя частями — мантиссой и порядком. При операции алгебраического сложения действия, выполняемые над мантиссами и порядками, различны. Следовательно, в цифровом автомате должны быть два раздельных устройства для обработки мантисс и для обработки порядков.

Так как для чисел с плавающей запятой справедливо условие (3.21), то всякий результат, не удовлетворяющий этому условию, должен быть приведен в соответствие с формулой (3.21). Такую операцию называют *нормализацией числа*. Операция нормализации числа состоит из проверки выполнимости условия (3.21) и сдвига изображения мантиссы в ту или иную сторону. Сдвиги могут осуществляться на один разряд и более в левую или правую сторону в пределах разрядной сетки машины.

Простой сдвиг — операция, выполняемая по следующим правилам:

Исходная комбинация	Сдвинутая влево на один разряд	Сдвинутая вправо на один разряд
$0, a_1a_2 \ldots a_n$	$a_1, a_2 \ldots a_n 0$	$0, 0a_1 \ldots a_{n-1}$
$1, a_1a_2 \ldots a_n$	$a_1, a_2 \ldots a_n \alpha$	$0, 1a_1 \ldots a_{n-1}$

Модифицированный сдвиг — операция над модифицированными изображениями, выполняемая следующим образом*:

Исходная комбинация	Сдвинутая влево на один ряд	Сдвинутая вправо на один разряд
$00, a_1a_2 \ldots a_n$	$0a_1, a_2 \ldots a_n 0$	$00, 0a_1 \ldots a_{n-1}$
$01, a_1a_2 \ldots a_n$	$1a_1, a_2 \ldots a_n 0$	$00, 1a_1 \ldots a_{n-1}$
$10, a_1a_2 \ldots a_n$	$0a_1, a_2 \ldots a_n \alpha$	$1, 0a_1 \ldots a_{n-1}$
$11, a_1a_2 \ldots a_n$	$1a_1, a_2 \ldots a_n \alpha$	$1, 1a_1 \ldots a_{n-1}$

Нарушение нормализации числа — невыполнение условия (3.21). Так как условие (3.21) содержит два неравенства, то может быть нарушение справа и слева. Признак нарушения нормализации числа справа γ (когда величина результата равна или превышает единицу) — наличие разноименных комбинаций в знаковых разрядах сумматора, т. е.

$$\gamma = 1, \text{ если } Sg_1 \wedge \overline{Sg_2} = 1 \,;\; \overline{Sg_1} \wedge Sg_2 = 1 \qquad (4.7)$$

* Величина α зависит от кода: для дополнительного кода $\alpha = 0$, для обратного кода $\alpha = 1$.

(в остальных случаях $\gamma = 0$), где γ — признак нарушения нормализации числа справа, указывающий на необходимость сдвига числа вправо на один разряд.

Признак нарушения нормализации числа слева δ (когда результат по собственной величине оказывается меньше $1/q$) — наличие одинаковых комбинаций в разряде переполнения и старшем разряде цифровой части сумматора (p_1):

$$\delta = 1, \text{ если } Sg_2 \wedge p_1 = 1;\ Sg_2 \wedge \overline{p}_1 = 1 \qquad (4.8)$$

(в остальных случаях $\delta = 0$), где δ — признак нарушения нормализации, указывающий на необходимость сдвига числа влево на один разряд.

Таким образом, операция нормализации числа состоит из совокупности сдвигов и проверки наличия признаков нарушения γ и δ.

Рассмотрим сложение чисел $A = m_A p_A$ и $B = m_B p_B$, имеющих одинаковый порядок $p_A = p_B$. Обе мантиссы удовлетворяют условию нормализации.

Сложение мантисс осуществляется на соответствующем сумматоре по правилам, изложенным ранее для чисел, представленных в форме с фиксированной запятой. Если после сложения мантисса результата удовлетворяет условию нормализации (т. е. $\delta = 0$, $\gamma = 0$), то к этому результату приписывается порядок любого из операндов. В противном случае происходит нормализация числа.

Пример 4.11. Найти сумму чисел $A = 0{,}1000 \cdot 2^{-1}$ и $B = -0{,}1011 \cdot 2^{-3}$, если мантиссы и порядок обрабатываются на сумматорах дополнительного кода (шесть разрядов для мантиссы и четыре разряда для порядка)

Решение. Сначала записываются машинные изображения операндов:

$$[m_A]^{м}_{д} = 00{,}1000;\ [p_A]_{д} = 1{,}101;$$

$$[m_B]^{м}_{д} = 11{,}0101;\ [p_B]_{д} = 1{,}101,$$

а затем мантиссы складываются:

$$\begin{array}{r} 00{,}1000 \\ +\ 11{,}0101 \\ \hline [m_C]^{м}_{д} = 11{,}1101 \end{array}$$

Здесь $Sg_2 \wedge p_1 = 1$, т. е. $\delta = 1$, $\gamma = 0$, значит необходим сдвиг мантиссы влево на разряд:

$$[m'_C]^{м}_{д} = 11{,}1010, (\delta = 1,\ \gamma = 0).$$

Одновременно со сдвигом влево нужна коррекция порядка, т. е. уменьшение его величины на единицу (что равносильно прибавлению кода 1,111):

$$\begin{array}{r} [p_C]_{д} = 1{,}101 \\ +\ 1{,}111 \\ \hline [p'_C]_{д} = 1{,}100 \end{array}$$

Так как после сдвига снова $\delta = 1$, то осуществляется еще раз сдвиг и коррекция порядка.

$$[m_C^*]_{\text{д}}^{\text{м}} = 11{,}0100,\ (\delta = 0,\ \gamma = 0),\quad [p_C']_{\text{д}} = 1{,}100$$
$$+\ 1{,}111$$
$$[p_C']_{\text{д}} = \overline{1{,}011}$$

Так будет продолжаться до тех пор, пока величина δ не станет равной нулю. Следовательно, $[m_C^*]_{\text{д}}$ удовлетворяет условию нормализации и результат равен $[m_C^*]_{\text{д}}^{\text{м}} = 11{,}0100$, $[p_C^*]_{\text{д}} = 1{,}011$.

Ответ $C = -0{,}1100\ 2^{-5}$.

Пример 4.12. Найти сумму чисел $A = -0{,}1100 \cdot 2^4$ и $B = -0{,}1000 \cdot 2^4$, если числа складываются на сумматоре обратного кода (шесть разрядов для мантиссы и четыре разряда для порядка).

Р е ш е н и е. Машинные изображения операндов записываются в следующем виде

$$[m_A]_{\text{об}}^{\text{м}} = 11{,}0011;\ [p_A]_{\text{об}} = [p_B]_{\text{об}} = 1{,}100;\ [m_B]_{\text{об}}^{\text{м}} = 11{,}01111.$$

Затем складываются мантиссы:

$$\begin{array}{r} 11{,}0011 \\ +\ \underline{11{,}0111} \\ [m_C']_{\text{об}}^{\text{м}} = 10{,}1011\ \ (\delta = 0,\ \gamma = 1). \end{array}$$

Здесь произошло нарушение нормализации справа и требуется модифицированный сдвиг мантиссы результата вправо на один разряд:

$$[m_C']_{\text{об}}^{\text{м}} = 10{,}0101\ (\delta = 0,\ \gamma = 0).$$

Одновременно со сдвигом проводится коррекция порядка результата на величину +0,001, или $[p_C']_{\text{об}} = 0{,}100 + 0{,}001 = 0{,}101$, в результате получается окончательный результат

Ответ $C = -0{,}1010 \cdot 2^{-5}$.

Рассмотрим наиболее общий случай сложения чисел, представленных в форме с плавающей запятой, когда их порядки не равны друг другу, т. е. $p_A \neq p_B$. Для операции сложения чисел необходимым условием является соответствие разрядов операндов друг другу. Значит, прежде всего нужно уравнять порядки, что, естественно, повлечет за собой временное нарушение нормализации одного из слагаемых. Выравнивание порядков означает, что порядок меньшего числа надо увеличить на величину $\Delta p = |p_A - p_B|$, что означает сдвиг мантиссы меньшего числа вправо на количество разрядов, равное Δp.

Следовательно, цифровой автомат должен самостоятельно определять, какой из двух операндов меньший. На это укажет знак разности $p_A - p_B$: положительный знак будет при $p_A \geq p_B$, а отрицательный — при $p_A < p_B$.

Операции сложения и вычитания чисел в форме с плавающей запятой осуществляются во всех современных машинах по изложенным выше правилам.

Пример 4.13. Сложить числа $A = 0{,}1011 \cdot 2^{-2}$ и $B = -0{,}1001 \cdot 2^{-3}$ на сумматорах обратного кода (шесть двоичных разрядов для мантиссы и четыре двоичных разряда для порядка)

Решение Прежде всего записываются машинные изображения чисел и определяется, какой из двух порядков больше:

$$[m_A]^{м}_{об} = 00{,}1011;\ [p_A]_{об} = 1{,}101;$$
$$[m_B]^{м}_{об} = 11{,}0110;\ [p_B]_{об} = 1{,}100,$$
$$[\Delta p]_{об} = [p_A]_{об} - [p_B]_{об}.$$

Величину $-[p_B]_{об}$ обозначим $[p_B]_{об}$, что означает изменение знака числа p_B на обратный, т. е. $[p_B]_{об} = 0{,}011$. Тогда $[\Delta p]_{об} = [p_A]_{об} + [p_B]_{об} = 0{,}001$.

Так как величина Δp положительна, то $p_A > p_B$. Следовательно, надо сдвинуть мантиссу числа B вправо на количество разрядов, равное Δp, т. е. на один разряд: $[m_B]^{м}_{об} = 11{,}1011$ (сдвиг модифицированный, стрелка над символом m_B показывает сдвиг в соответствующую сторону). Теперь порядки операндов равны и дальнейшие действия проводятся в последовательности, аналогичной последовательности, рассмотренной в примере 4.12.

Складываются изображения мантисс:

$$\begin{array}{l} [m_A]^{м}_{об} = 00{,}1011 \\ + \\ \underline{[m_B]^{м}_{об} = 11{,}1011} \\ [m_C]^{м}_{об} = 00{,}0111 \quad (\delta = 1,\ \gamma = 0). \end{array}$$

Осуществляется нормализация мантиссы ($\delta = 1$) и соответствующая коррекция порядка

$$[m'_C]_{об} = 00{,}1110 \quad (\delta = 0,\ \gamma = 0),$$
$$[p'_C]_{об} = 1{,}101 + 1{,}110 = 1{,}100.$$

Так как нарушений нормализации нет, то получен окончательный результат.

Ответ $C = 0{,}1110 \cdot 2^{-3}$.

Пример 4.14. Сложить числа A и B, заданные в форме с фиксированной точкой: $m_A = 100110$, $X_A = 101$; $X_B = -111001$; $X_B = 011$. Для выполнения операции сложения использовать сумматор дополнительного кода, имеющий семь битов для мантиссы со знаком, четыре бита для характеристики со знаком.

Решение Запишем машинные изображения мантисс: $[m_A]^{м}_{д} = 00{,}100110$; $[m_B]^{м}_{д} = 11{,}000111$

Исходные числа в памяти машины можно хранить либо в прямом, либо в обратном (дополнительном) кодах. Если числа хранятся в памяти машины в прямом коде, то при выпол-

нении операции сложения (вычитания) на сумматорах обратного (дополнительного) кода необходимо провести преобразование из прямого кода в обратный (дополнительный) код. По окончании операции должно проводиться преобразование результата из обратного (дополнительного) кода в прямой

При выполнении данного примера предполагается, что числа в памяти машины хранятся в дополнительном коде.

Прежде всего необходимо сравнить характеристики:

$$\Delta X = [X_A]_д - [X_B]_д = 0{,}101 + 1{,}101 = 0{,}010 .$$

Разность характеристик — положительная: второй порядок меньше первого на 2. Следовательно, мантисса второго числа сдвигается на два разряда (сдвиг модифицированный) и после этого мантиссы складываются:

$$\begin{array}{l} [\tilde{m}_B]_д = 11{,}110001 \\ + \\ [m_A]_д = 00{,}100110 \\ \hline [m_C]_д = 00{,}010110 \quad (\delta = 1,\ \gamma = 0). \end{array}$$

Так как $\delta = 1$, то проводится сдвиг влево на один разряд с коррекцией характеристики:

$$[m'_C]_д = 00{,}101110 \ (\delta = 0,\ \gamma = 0)\ [X_C]_д = 0{,}101 + 1{,}111 = 0{,}100 .$$

Таким образом, окончательный результат получен в нормализованном виде.

Ответ $C = +101110$, $X_C = 100$.

Пример 4.14 приведен для случая, когда мантисса — целое число и представляется в форме с фиксированной точкой перед старшим разрядом. Сформулированные выше правила выполнения алгоритма алгебраического сложения действуют в данном случае без существенных изменений.

При реализации операций сложения (вычитания) чисел, представленных в форме с плавающей запятой, может возникнуть переполнение разрядной сетки сумматора порядков (характеристик): мантисса получается нормализованной и правильной, а порядок (характеристика) не соответствует. Следовательно, необходимо вырабатывать сигнал переполнения сумматора порядков.

Нормализация результата операции сложения (вычитания) приводит и к исчезновению порядка (т. е. характеристика становится отрицательной), несмотря на то, что мантисса отлична от нуля. В ЕС ЭВМ вводится специальный разряд, в котором записывается нуль или единица: при нулевом значении этого разряда в результате операции записывается истинный нуль, т. е. число с нулевой мантиссой, положительным знаком и нулевой характеристикой; при единичном значении этого разряда к характеристике прибавляется $+X_{max}$.

4.5. Методы ускорения операции сложения

Уменьшить время на выполнение операции сложения можно разными путями. Идея, лежащая в основе многих методов ускорения операции сложения, заключается в том, что осуществляется поэтапное получение условных сумм и переносов с последующей их раздельной обработкой. Наибольший эффект методы раздельной обработки условных сумм и переносов дают при параллельной или последовательно-параллельной обработке разрядов. Покажем эту идею на примере сложения чисел A и B:

	$A = 0011110100111010$	
	+	
	$B = 1001001101001101$	
1-й такт	$c' = 1010111001110111$	условная сумма
	+	
	$п' = 0010001000010000$	переносы
2-й такт	$c'' = 1000110001100111$	
	+	
	$п'' = 0100010000100000$	
3-й такт	$c''' = 1100100001000111$	
	+	
	$п''' = 0000100001000000$	
4-й такт	$c^{IV} = 1100000000000111$	
	+	
	$п^{IV} = 0001000010000000$	
	$c = 1101000010000111$	результат
	0000000000000000	перенос

Весь процесс сложения прошел за пять тактов. Признак получения результата — нулевые значения поразрядных переносов. Метод раздельного сложения условных сумм и переносов в свое время послужил толчком для создания специальных сумматоров со сквозным переносом, в которых достигается существенное сокращение времени за счет удаления этапов передачи переносов через разряды, в которых условная сумма равна единице. Этот метод был впервые широко использован в ЭВМ БЭСМ-6.

Метод может быть усовершенствован, если использовать следующий прием*. Для каждого разряда складываемых операндов вычисляются две пары сумм и переносов: одна — в предположении, что перенос, вносимый в данный разряд, равен нулю (условие α), другая пара — в предположении, что перенос, вносимый в данный разряд, равен единице (условие β). Исклю-

* *Sklansky J.* Conditional sum addition logic (RE Trans. on Electronic Comput. 1960, N 2, p. 226–231).

чение представляет нулевой разряд, в котором перенос равен только нулю. Таким образом, правила получения этих пар выглядят следующим образом: для нулевого разряда

$$c_0^0 = a_0 \oplus b_0 = 1, \text{ условная сумма}$$

$$п_1^0 = a_0 \cdot b_0 = 0, \text{ перенос;}$$

для первого разряда

$$c_1^0 = a_1 \oplus b_1 = 1,\ п_2^0 = a_1 \cdot b_1 = 0,$$

$$c_1^1 = c_1^0 \oplus 1 = 0,\ п_2^1 = a_1 \vee b_1 = 1 \text{ и т. д.}$$

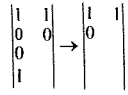
Для первого такта условные суммы и переносы вычисляются для всех разрядов.

Во втором такте условные суммы и переносы определяются для пар соседних разрядов (0 и 1, 2 и 3 и т. д.) при условиях α и β.

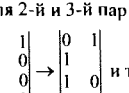
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Номер разряда
	0	0	1	1	1	1	0	1	0	0	1	1	1	0	1	0	
	1	0	0	1	0	0	1	1	0	1	0	0	1	1	0	1	
	1	0	1	0	1	1	1	0	0	1	1	1	0	1	1	1	c^0 сумма и перенос
1-й такт	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	$п^0$ при условии α
	0	1	0	1	0	0	0	1	1	0	0	0	1	0	0		c' сумма и перенос
	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1		$п'$ при условии β
	1	0	0	1	1	1	0	1	0	1	1	1	0	1	1	1	
	0		1		0		1		0		0		1		0		
2-й такт	0	1	1	0	0	0	1	0	1	0	0	0	1	0			
	1		0		1		0		0		1		1				
	1	0	0	1	0	0	0	1	1	0	0	0	0	1	1	1	
	0				0				0				1				
3-й такт	1	1	0	1	0	0	0	0	1	0	0	0					
	1				1				1								
	1	1	0	1	0	0	0	0	1	0	0	0	0	1	1	1	
	0								1								
4-й такт	1	1	0	1	0	0	0	0									
	0																
5-й такт	1	1	0	1	0	0	0	0	1	0	0	0	0	1	1	1	результат

Рассмотрим пару соседних разрядов 0 и 1. Во втором такте входы в верхнюю половину первого разряда находим следующим образом: если второй вход нулевого разряд равен 0, т. е. $п_1^0 = 0$, то верхняя половина первого разряда идентична верхней половине первого такта; если же $п_1^0 = 1$, то

верхняя половина первого разряда идентична нижней половине первого такта. В итоге преобразование

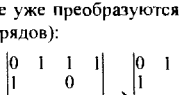


Аналогичным образом для 2-й и 3-й пар разрядов:

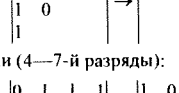


и т. д.

В следующем такте уже преобразуются четверки разрядов (например, для первых четырех разрядов):



или для другой четверки (4—7-й разряды):



В результате появляется возможность весь процесс суммирования реализовать на схемах типа И, ИЛИ, НЕ.

4.6. Оценка точности выполнения арифметических операций

Выбор системы счисления и длины разрядной сетки машины, а также формы представления числа в машине тесно связан с обеспечением заданной точности вычислений. Важное значение имеет также оценка точности арифметических вычислений при использовании в машинах чисел, представленных в форме с фиксированной и плавающей запятой. При операциях сложения и вычитания с фиксированной запятой (при условии отсутствия переполнения в естественной форме) можно считать, что они выполняются точно.

Для чисел, представленных в форме с плавающей запятой, при операциях сложения и вычитания необходимо выравнивать порядки, что ведет к

потере некоторых разрядов мантиссы при сдвиге. Поэтому при нормализованной форме представления чисел сама операция алгебраического сложения также является источником погрешностей.

Таким образом, причинами погрешностей вычисления на ЭВМ могут быть:

неточное задание исходных данных, участвующих в выполняемой операции (либо из-за ограниченности разрядной сетки машины, либо из-за погрешностей перевода информации из одной системы счисления в другую);

использование приближенных методов вычислений, что само по себе дает методическую погрешность (например, использование рядов Ньютона и Тейлора при интегрировании);

округление результатов элементарных операций, что, в свою очередь, может привести к появлению накопленных погрешностей;

сбои в работе ЭВМ (эта причина может быть устранена введением системы контроля выполнения любых операций).

Погрешности выполнения арифметических операций. Для определения этих погрешностей в цифровых автоматах будем рассматривать арифметические операции как элементарные операции над операндами.

Проведем арифметические действия над числами $A=[A]+\Delta A$ и $B=[B]+\Delta B$, заданными с абсолютными погрешностями:

$$A+B=[A]+[B]+(\Delta A+\Delta B);$$
$$A-B=[A]-[B]+(\Delta A-\Delta B);$$
$$AB=[A][B]+[A]\Delta B+[B]\Delta A+\Delta A\Delta B,$$

где абсолютная погрешность суммы $\Delta(A+B)=\Delta A+\Delta B$, а абсолютная погрешность разности $\Delta(A-B)=\Delta A-\Delta B$.

Так как произведение $\Delta A\Delta B$ на два порядка меньше чисел A и B, этим произведением можно пренебречь. Следовательно, $AB\approx[A][B]+[A]\Delta B+$ $+[B]\Delta A$, т. е. абсолютная погрешность произведения $(AB)=[A]\Delta B+[B]\Delta A$.

При выполнении операции деления получаем

$$\frac{A}{B}=\frac{[A]+\Delta A}{[B]+\Delta B}=\frac{[A]+\Delta A}{[B]}\left(\frac{1}{1+\Delta B/[B]}\right).$$

Второй сомножитель в правой части уравнения разложим в ряд. После преобразований получим

$$A/B=[A]/[B]-[A]\Delta B/[B]^2+[A](\Delta B)^2/[B]^3+\Delta A/[B]-\Delta A\Delta B/[B]^2-\ldots \quad (4.9)$$

Пренебрегая членами второго порядка малости, (4.9) можно упростить: $A/B \approx [A]/[B] + \Delta A/[B] - [A]\Delta B/[B]^2$. Отсюда абсолютная погрешность частного $\Delta(A/B) = \Delta A/[B] - [A]\Delta B/[B]^2$.

Аналогичным образом можно вывести выражения для относительных погрешностей при сложении-вычитании, умножении и делении соответственно:

$$\delta_{A\pm B} = \frac{[A]}{[A]+[B]}\frac{\Delta A}{[A]} \pm \frac{[B]}{[A]+[B]}\frac{\Delta B}{[B]}; \tag{4.10}$$

$$\delta_{AB} = \Delta A/[A] + \Delta B/[B]; \tag{4.11}$$

$$\delta_{A/B} = \Delta A/[A] - \Delta B/[B]. \tag{4.12}$$

Погрешности округления. Если предположить, что исходная информация не содержит никаких ошибок и все вычислительные процессы конечны и не приводят к ошибкам, то все равно присутствует третий тип ошибок — ошибки округления. Предположим, что вычисления проводят на некоторой гипотетической машине, в которой каждое число представляется пятью значащими цифрами, и что необходимо сложить числа 9,2654 и 7,1625, причем эти числа точные. Сумма чисел равна 16,4279, она содержит шесть значащих цифр и не помещается в разрядной сетке машины. Поэтому шестизначный результат будет округлен до значения 16,428. В результате возникает погрешность округления.

Так как вычислительные машины всегда работают с конечным количеством значащих цифр, то погрешность в округлении возникает довольно часто. Погрешность округления имеет смысл только для действительных чисел; это объясняется тем, что ЭВМ автоматически выравнивает порядки действительных чисел при сложении и вычитании.

Для чисел, представленных в форме с плавающей запятой, справедливы выражения $A_q = m_A q^k$, $1/q \le |m_A| < 1$.

Если для представления мантиссы используется только n разрядов, то изображение числа разбивается на две части:

$$A_q = [m_A]q^n + [A_0]q^{k-n},$$

где $[A_0]q^{k-n} = A_0$ — «хвост» числа, не попавший в разрядную сетку.

В зависимости от того, как учитывается величина A_0, в машинном изображении существует несколько способов округления.

1. Отбрасывание A_0. При этом возникает относительная погрешность, равная

$$\delta_{\text{окр}} = |A_0|q^{k-n} / (|m_A|q^k).$$

Так как $q^{-1} \le |m_A| < 1$; $0 \le |A_0| < 1$, то

$$\delta_{\text{окр}} \le \frac{1q^{k-n}}{q^{-1}q^k} = q^{-(n-1)}, \tag{4.13}$$

т. е. математическое ожидание погрешности округления не зависит от величины самого числа, а зависит только от количества разрядов в машине для любой системы счисления.

Дисперсия погрешности округления примерно равна $q^{2n}/12$.

2. Симметричное округление. При этом проводится анализ величины A_0. Принимается, что

$$[A] = \begin{cases} [m_A]q^n, \text{ если } |A_0| < q^{-1}; \\ [m_A]q^n + q^{k-n}, \text{ если } |A_0| \ge q^{-1}. \end{cases} \tag{4.14}$$

При условии $|A_0| \ge q^{-1}$ проводится прибавление единицы к младшему разряду мантиссы. Абсолютная погрешность округления при этом

$$\Delta_{\text{окр}} = \begin{cases} |A_0|q^{k-n}; \\ |1 - A_0|q^{k-n}. \end{cases} \tag{4.15}$$

Максимально возможное значение модуля абсолютной погрешности равно $0{,}5q^{k-n}$. Математическое ожидание относительной погрешности округления

$$\delta_{\text{окр}} \le 0{,}5q^{k-n}/(m_A q^k) = 0{,}5q^{-(n-1)}, \tag{4.16}$$

т. е. ошибка не превышает половины единицы младшего разряда.

Способ симметричного округления наиболее часто используют на практике.

3. Округление по дополнению. В этом случае для округления берется информация, содержащаяся в $(n+1)$-м разряде.

При $q = 2$, если в $(n+1)$-м разряде содержится единица, то в n-й разряд добавляется единица; если в $(n+1)$-м разряде находится нуль, то содержимое разрядов правее n-го отбрасывается.

4. Случайное округление. Для такого округления необходимо иметь датчик случайных величин (1 или 0), который выдает единицу в самый младший разряд машинного изображения числа. Погрешность округления — случайная величина с нулевым математическим ожиданием.

Оценка накопленной погрешности при вычислениях на машине особенно затруднительна при использовании чисел в форме с плавающей запя-

той. При таком представлении возможно перемещение ошибки из младших разрядов мантиссы в старшие разряды. Это происходит, например, при вычитании друг из друга близких по значению мантисс. В результирующей мантиссе первые нулевые разряды оказываются сдвинутыми в правую часть разрядной сетки машины. При нормализации они перемещаются в левую часть разрядной сетки, давая большую погрешность результата.

Для автоматической оценки накопленной ошибки при вычислении чисел в форме с плавающей запятой в разрядной сетке машины кроме числовой информации записывается также информация об ошибке, содержащейся в числовой информации. При этом предполагается, что ошибки всех чисел — независимые величины, и их распределение подчинено нормальному закону. Эти допущения весьма существенны, так как на практике ошибки при вычислениях, конечно, являются зависимыми величинами и их распределение может быть далеким от нормального. Кроме того, принимается, что все числа, записанные в разрядной сетке машины, имеют погрешность $+0{,}5$ последней значащей цифры. Значение этой вероятной ошибки записывается в исходных данных в разрядах, находящихся правее самого младшего разряда мантиссы. После арифметических операций нормализация осуществляется не всегда, а лишь в случаях, когда срабатывает критерий сдвига, оценивающий величину погрешности, вносимой в число в процессе нормализации.

Оценка точности вычислений на машинах зависит не только от состава выполняемых операций, но и от их следования друг за другом.

Задание для самоконтроля

1. Написать изображения чисел $A = -0{,}101010$ и $B = 0{,}100010$ в прямом, обратном и дополнительном кодах.

2. Возможно ли переполнение разрядной сетки, если числа с плавающей запятой складываются, умножаются, делятся?

3. Сложить на сумматоре прямого кода числа $A = -0{,}11101$ и $B = 0{,}10100$.

4. Сложить на сумматоре обратного кода числа $A = 0{,}10110$ и $B = -0{,}10110$.

5. Сложить на сумматоре дополнительного кода числа $A = 0{,}11001$ и $B = 0{,}10111$.

6. Указать признак переполнения разрядной сетки на сумматоре обратного кода при сложении отрицательных чисел и положительных чисел.

7. Применимы ли понятия обратного, дополнительного и прямого кодов для представления чисел в минус-двоичной системе счисления?

5. ВЫПОЛНЕНИЕ ОПЕРАЦИЙ УМНОЖЕНИЯ ЧИСЕЛ НА ДВОИЧНЫХ СУММАТОРАХ

5.1. Методы умножения двоичных чисел

Применительно к двоичной системе счисления наиболее известны следующие основные способы выполнения операции умножения:

1) умножение начиная с младших разрядов множителя:

```
    1101   — множимое,
  × 1101   — множитель,
    1101
   0000
  1101     — частные произведения,
 1101
1010 1001  — произведение;
```

2) умножение начиная со старших разрядов множителя:

```
    1101     — множимое,
  × 1101     — множитель,
    1101
  + 1101     — частные произведения,
    0000
      1101
1010 1001    — произведение.
```

В обоих случаях операция умножения состоит из ряда последовательных операций сложения частных произведений. Операциями сложения управляют разряды множителя: если в каком-то разряде множителя находится единица, то к сумме частных произведений добавляется множимое с соответствующим сдвигом; если в разряде множителя — нуль, то множимое не прибавляется.

Таким образом, кроме операции сложения чисел для получения произведения необходима операция сдвига чисел. При этом появляется возможность сдвигать множимое или сумму частных произведений, что дает основание для разных методов реализации операции умножения.

Метод 1. Пусть A — множимое ($A > 0$), B — множитель ($B > 0$), C — произведение. Тогда в случае представления чисел в форме с фиксированной запятой получаем: $A = 0, a_1 a_2 \ldots a_n$; $B = 0, b_1 b_2 \ldots b_n = b_1 \cdot 2^{-1} + b_2 \cdot 2^{-2} + \ldots + b_n \cdot 2^{-n}$.

Отсюда

$$C = AB = 0, a_1 a_2 \ldots a_n (b_1 \cdot 2^{-1} + b_2 \cdot 2^{-2} + \ldots + b_n \cdot 2^{-n}) = (2^{-1} \cdot 0, a_1 a_2 \ldots a_n) b_1 + (2^{-2} \cdot 0, a_1 a_2 \ldots a_n) b_2 + \ldots + (2^{-n} \cdot 0, a_1 a_2 \ldots a_n) b_n. \quad (5.1)$$

Умножение на 2^{-n} означает сдвиг на n разрядов вправо числа, которое заключено в скобки, т. е. в данном случае сдвигается вправо множимое, и умножение начинается со старших разрядов.

Структурная схема рассмотренного множительного устройства представлена на рис. 5.1, *а*.

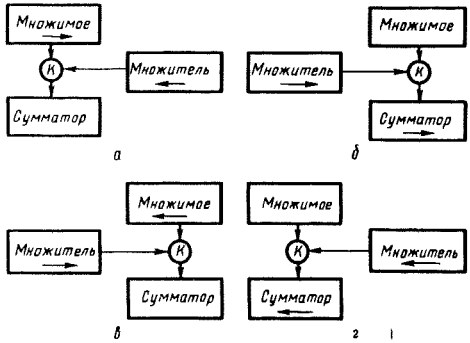


Рис. 5.1. Структурные схемы множительных устройств

Метод 2. Пусть $A = 0, a_1 a_2 \ldots a_n$ — множимое и $B = 0, b_1 b_2 \ldots b_n$ — множитель.

Множитель можно легко преобразовать, используя метод Горнера: $B = (\ldots((b_n \cdot 2^{-1} + b_{n-1})2^{-1} + \ldots + b_2)2^{-1} + b_1)2^{-1}$. Тогда

$$C = AB = (\ldots((b_n \cdot 0, a_1 a_2 \ldots a_n) \cdot 2^{-1} + b_{n-1} \cdot 0, a_1 a_2 \ldots a_n)2^{-1} + \ldots + b_1 \cdot 0, a_1 a_2 \ldots a_n) \cdot 2^{-1}. \quad (5.2)$$

Здесь умножение начинается с младших разрядов, и сдвигается вправо сумма частных произведений. Структурная схема множительного устройства, реализующего этот метод, представлена на рис. 5.1, *б*.

Метод 3. Пусть $A = 0, a_1a_2 \ldots a_n$ — множимое и $B = 0, b_1b_2 \ldots b_n$ — множитель.

Множитель, используя метод Горнера, можно записать так:

$$B = 2^{-n}(b_1 \cdot 2^{n-1} + b_2 \cdot 2^{n-2} + \ldots + b_{n-1} \cdot 2^1 + b_n \cdot 2^0) =$$
$$= 2^{-n}(\ldots(b_1 \cdot 2^1 + b_2)2^1 + \ldots + b_{n-1}2^1 + b_n).$$

В этом случае

$$C = AB = 2^{-n}(b_n \cdot 0, a_1a_2 \ldots a_n + (2^1 \cdot 0, a_1a_2 \ldots a_n)b_{n-1} + \\ + \ldots + (2^{n-1} \cdot 0, a_1a_2 \ldots a_n) \cdot b_1), \tag{5.3}$$

что означает: умножение начинается с младших разрядов, и множимое сдвигается влево на один разряд в каждом такте. Схема множительного устройства представлена на рис. 5.1, *в*.

Метод 4. Пусть $A = 0, a_1a_2 \ldots a_n$ — множимое и $B = 0, b_1b_2 \ldots b_n$ — множитель.

Если множитель B записать по методу Горнера:

$$C = AB = 2^{-n}((\ldots(2^1(b_1 \cdot 0, a_1a_2 \ldots a_n) + b_2 \cdot 0, a_1a_2 \ldots a_n)2^1 + \\ + \ldots + b_{n-1} \cdot 0, a_1a_2 \ldots a_n)2^1 + b_n \cdot 0, a_1a_2 \ldots a_n), \tag{5.4}$$

то умножение начинается со старшего разряда и в каждом такте сдвигается влево сумма частных произведений. Схема множительного устройства представлена на рис. 5.1, *г*.

Таким образом, для реализации операции умножения необходимо иметь сумматор, регистры для хранения множимого и множителя и схему анализа разрядов множителя. Сумматор и регистры должны иметь цепи сдвига содержимого в ту или иную сторону в соответствии с принятым методом умножения.

Анализ формул (5.1)–(5.4) показывает, что с формальной точки зрения процесс умножения двух чисел может быть представлен:

при последовательном выполнении — в виде многократно повторяющегося по количеству разрядов цикла

$$S_i = S_{i-1} + Ab_i, \tag{5.5}$$

где S_i, S_{i-1} — суммы частных произведений на $(i-1)$-м и i-м шагах соответственно;

при параллельном выполнении — суммой членов диагональной матрицы, для которой заданы по строкам $A \cdot 2^i$, а по столбцам — b_i (i — текущий номер разряда).

В дальнейшем основное внимание будет уделено последовательному принципу выполнения операции умножения.

При точном умножении двух чисел количество цифр в произведении превышает количество цифр сомножителей не более чем в два раза. При умножении нескольких чисел количество цифр произведения может оказаться еще больше. Конечное число разрядов в устройствах цифрового автомата вынуждает ограничиваться максимально удвоенным количеством разрядов сумматоров.

При ограничении количества разрядов сумматора в произведение вносится погрешность. В случае большого объема вычислений погрешности одного знака накладываются друг на друга, в результате чего общая погрешность сильно возрастает. Поэтому существенное значение имеет округление результатов умножения, что дает возможность сделать погрешность произведения знакопеременной, а математическое ожидание погрешности округления при условии, что отброшенные младшие разряды могут с одинаковой вероятностью иметь любое из возможных значений, — равным нулю. При этом предельное по абсолютной величине значение погрешности будет наименьшим из возможных при заданном количестве значащих цифр, т. е. равным половине значения младшего разряда.

При выполнении операции умножения чисел возможен выход за пределы разрядной сетки только со стороны младших разрядов в силу ограничения, которое было наложено на числа, представленные в форме с фиксированной запятой. Точное произведение получается во всех четырех методах умножения, однако при этом требуется разное количество оборудования.

5.2. Умножение чисел, представленных в форме с фиксированной запятой, на двоичном сумматоре прямого кода

Пусть заданы машинные изображения двух чисел:

$$[A]_{\text{пр}} = Sg_A, a_1 a_2 \ldots a_n, \; [B]_{\text{пр}} = Sg_B, b_1 b_2 \ldots b_n .$$

Тогда их произведение

$$C_{\text{пр}} = Sg_C, c_1 c_2 \ldots c_n ,$$

где — $Sg_C = Sg_A \oplus Sg_B$; $\oplus$ — знак сложения по модулю 2.

При выполнении этой операции должны быть заданы структурная схема устройства, на котором проводится операция, и метод умножения.

Пример 5.1. Умножить числа $[A]_{пр} = 1{,}11010$ и $[B]_{пр} = 0{,}11001$.

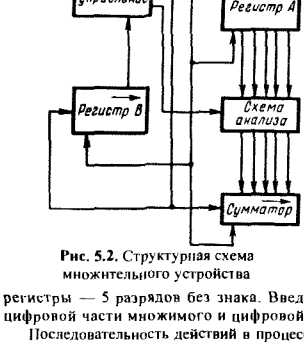


Рис. 5.2. Структурная схема множительного устройства

При умножении будут использованы метод 2 и устройство, показанное на рис. 5.2.

Запись всех действий, выполняемых устройством, осуществляется с помощью условных обозначений, т. е.: = — оператор присваивания означает, что блоку, который указан слева от оператора, присваивается значение, указанное справа от оператора; $[\overrightarrow{\text{Рг}A}]$ — сдвиг содержимого регистра Рг вправо на один разряд; [СМ] — содержимое сумматора СМ; И.П. — исходное положение.

Р е ш е н и е. Знак произведения определяем отдельно от цифровой части в соответствии с уравнением

$$Sg_C = Sg_A \oplus Sg_B = 1 \oplus 0 = 1$$

Получение цифровой части можно показать в виде следующей записи. Пусть сумматор имеет 10 разрядов без учета знака, а регистры — 5 разрядов без знака. Введем обозначения соответственно изображения цифровой части множимого и цифровой части множителя.

Последовательность действий в процессе выполнения операции умножения представлена в виде таблицы 5.1.

Ответ. $[C]_{пр} = 1{,}1010001010$.

Таблица 5.1

Сумматор	Регистр B	Примечание
0000000000	11001	И.П. [СМ] := 0 ; $[\text{Рг}A] := [A']$; $[\text{Рг}B] := [B]$.
+ 11010		$b_5 = 1$; [СМ] := [СМ] + $[\text{Рг}A]$;
1101000000		$[\overrightarrow{\text{Рг}B}]$; $[\overrightarrow{\text{СМ}}]$;
0110100000	—1100	$b_4 = 0$; $[\overrightarrow{\text{Рг}B}]$; $[\overrightarrow{\text{СМ}}]$;
0011010000	—110	$b_3 = 0$; $[\overrightarrow{\text{Рг}B}]$; $[\overrightarrow{\text{СМ}}]$;
0001101000	—11	$b_2 = 1$; [СМ] := [СМ] + $[\text{Рг}A]$
+11010		$[\overrightarrow{\text{СМ}}]$ $[\overrightarrow{\text{Рг}B}]$
1110101000		
0111010100	—1	$b_1 = 1$; [СМ] := [СМ] + $[\text{Рг}A]$
+11010		
10100010100*		$[\overrightarrow{\text{Рг}B}]$; $[\overrightarrow{\text{СМ}}]$;
1010001010		Конец

* Если в процессе выполнения умножения возникает единица переноса из старшего разряда, то ее надо сохранять.

Чтобы процесс умножения проходил правильно, необходимо предусмотреть блокировку выработки сигнала переполнения, так как возможно временное переполнение на каком-то шаге умножения (см. пример 5.1). Пример показывает, что в данном случае не обязательно иметь сумматор длиной $2n$ разрядов. Хранение «хвостов» произведения можно осуществлять в освобождающихся разрядах регистра множителя. Для этого достаточно обеспечить цепь передачи информации из младшего разряда сумматора в старший разряд регистра множителя.

Во всех приведенных ниже примерах будет применяться рассмотренный способ.

5.3. Особенности умножения чисел, представленных в форме с плавающей запятой

Для чисел, представленных в форме с плавающей запятой, обязательным является представление в виде мантиссы и порядка (характеристики). При операции умножения действия, выполняемые над мантиссами и порядками, различны: мантиссы перемножаются, порядки складываются. Очевидно, что результат умножения может получиться ненормализованным, тогда потребуется нормализация с соответствующей коррекцией порядка результата. Следовательно, структурная схема множительного устройства должна измениться (рис. 5.3).

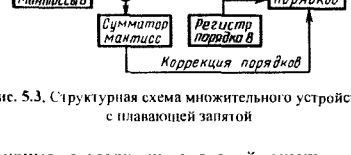


Рис. 5.3. Структурная схема множительного устройства с плавающей запятой

Рассмотрим пример выполнения операций умножения чисел, заданных в прямом коде.

Пример 5.2. Перемножить числа $A = -0{,}11001 \cdot 2^{-1}$ и $B = 0{,}10011 \cdot 2^{+1}$.

В качестве множительного устройства используется схема, показанная на рис. 5.3, где РгА и РгB — соответственно регистры для порядков p_A и p_B.

Р е ш е н и е . Мантиссы перемножаются по правилам, рассмотренным для чисел, представленных в форме с фиксированной запятой. Для перемножения мантисс используется сумматор прямого кода, а для сложения порядков — сумматор обратного кода.

Сначала записываются машинные изображения чисел:

$$[m_A]_{пр} = 1{,}11001;\ [p_A]_{об} = 1{,}100\ .$$

$$[m_B]_{пр} = 0{,}10011;\ [p_B]_{об} = 0{,}001\ .$$

Последовательность действий в процессе выполнения операции умножения мантисс представим в таблице 5.2

После выполнения указанных действий находится мантисса произведения $[m_t]_{пр} = 1{,}0111011011$

Одновременно с этим над порядками проводится операция сложения $[p_t]_{об} = [p_A]_{об} + [p_B]_{об} = 1{,}100 + 0{,}001 = 1{,}101$.

Так как мантисса результата не удовлетворяет условию нормализации (нарушена левая граница, $\delta = 1$, $\gamma = 0$), то проводится сдвиг мантиссы влево на один разряд $[m_t']_{пр} = 1{,}1110110110$, и коррекция порядка $[p_t']_{об} = [p_t]_{об} + 1{,}110 = 1{,}101 + 1{,}110 = 1{,}110$

Таблица 5.2

Знак результата	Сумматор	Регистр B	Примечание
$Sg_t = Sg_A \oplus \oplus Sg_B = 1 \oplus 0 = 1$	00000	10011	И.П. $Pr m_B := [m_B]$; $Pr m_A := [m_A]$; $Pr p_A := [p_A]$; $Pr p_B := [p_B]$; $CMm := 0$.
	+ 11001		$b_5 = 1$; $CMm := [CMm] + [Pr m_A]$;
	11001	01001	$[\overrightarrow{CMm}]; [\overrightarrow{Pr m_B}]$;
	011001		
	+ 11001		$b_4 = 1$; $CMm := [CMm] + [Pr m_A]$;
	1001011		
	1001011	00100	$[\overrightarrow{CMm}]; [\overrightarrow{Pr m_B}]$;
	01001011	00010	$b_3 = 0$; $[\overrightarrow{CMm}]$; $[\overrightarrow{Pr m_B}]$;
	001001011	00001	$b_2 = 0$; $[\overrightarrow{CMm}]$; $[\overrightarrow{Pr m_B}]$;
	+ 11001		$b_1 = 1$; $CMm := [CMm] + [Pr m_A]$;
	111011011		
	0111010111011	00000	$[\overrightarrow{CMm}]$; $[\overrightarrow{Pr m_B}]$;
			Конец

Если сумматор мантисс содержит только n разрядов, то после округления получается исходный результат

Ответ: $C = -0{,}11110\ 2^{\backslash}$

При выполнении операции умножения может иметь место ряд особых случаев. Например:

если один из сомножителей равен нулю, то произведение также равно нулю. Следовательно, необходимо предусмотреть блокировку выполнения алгоритма умножения и формировать результат, равный нулю;

если порядок результата равен наибольшей отрицательной величине, то необходимо формировать машинный нуль.

Эти особые случаи можно предусмотреть в алгоритме операции введением анализатора сомножителей на нуль в начале операции (первый случай) или коррекцией произведения на основании признаков результата (второй случай).

5.4. Умножение чисел, представленных в форме с фиксированной запятой, на двоичном сумматоре дополнительного кода

В случаях, когда числа в машине хранятся в дополнительных кодах, целесообразно все операции над числами проводить на сумматоре дополнительного кода. Однако при этом возникает ряд особенностей, которые необходимо учитывать.

Произведение дополнительных кодов сомножителей равно дополнительному коду результата только в случае положительного множителя.

Пусть множимое A — любое число, т. е. $A = [A]_д$, а множитель $B > 0$. Тогда

$$AB = [A]_д \cdot 0, b_1 b_2 \ldots b_n = [A]_д b_1 \cdot 2^{-1} + [A]_д b_2 \cdot 2^{-2} + \ldots + [A]_д b_n \cdot 2^{-n}. \quad (5.6)$$

На основании теоремы о сложении дополнительных кодов можно утверждать, что в правой части уравнения (5.6) стоит дополнительный код результата.

Таким образом, умножение на сумматоре дополнительного кода заключается в анализе разрядов множителя и при $b_i = 1$ в прибавлении дополнительного кода множимого к содержимому сумматора. При этом должны осуществляться модифицированные сдвиги.

Пример 5.3. Умножить на сумматоре дополнительного кода (используется метод 2) числа $A = -0{,}10101$ и $B = 0{,}10011$.

Решение. Сначала записываются машинные изображения чисел: $[A]_д^м = 11{,}01011$; $[B]_д^м = 00{,}10011$.

Последовательность действий, проводимых над числами, представлена в таблице 5.3.

Ответ $C = AB = -0{,}0110001111$.

Теперь рассмотрим случай, когда множимое A — любое число, а множитель $B < 0$. Тогда $[B]_д = 1, \bar{b}_1\bar{b}_2 \ldots \bar{b}_n$.

На основании (3.24) можно записать, что $B = [B]_д - 2$, или $B = 0, \bar{b}_1\bar{b}_2 \ldots \bar{b}_n - 1$. Следовательно, произведение чисел

$$AB = A(0, \bar{b}_1\bar{b}_2 \ldots \bar{b}_n - 1) = A \cdot 0, \bar{b}_1\bar{b}_2 \ldots \bar{b}_n - A. \qquad (5.7)$$

Таблица 5.3

Сумматор	Регистр B	Примечание
00,00000	10011	И.П. СМ := 0; $Pr A := [A]^м_д$; $Pr B := [B]$;
+ 11,01011		$b_5 = 1$; СМ := [СМ] + [PrA];
11,01011		
11,10101	→ 11001	$[\overrightarrow{Pr B}]$; $[\overrightarrow{СМ}]$;
+ 11,01011		$b_4 = 1$; СМ := [СМ] + [PrA];
11,00000		$[\overrightarrow{Pr B}]$; $[\overrightarrow{СМ}]$;
11,10000	→ 01100	$b_3 = 0$; $[\overrightarrow{Pr B}]$; $[\overrightarrow{СМ}]$;
11,11000	→ 00110	$b_2 = 0$; $[\overrightarrow{Pr B}]$; $[\overrightarrow{СМ}]$;
11,11100	→ 00011	$b_1 = 0$; СМ := [СМ] + [PrA]
+ 11,01011		
11,00111	→ 10001	$[\overrightarrow{Pr B}]$; $[\overrightarrow{СМ}]$
11,10011		Конец

Формула (5.7) показывает, что при отрицательном множителе произведение дополнительных кодов операндов не равно дополнительному коду результата. Если ввести замену $-A$ на A, то можно вывести следующее правило:

Если множитель отрицательный, то произведение чисел на сумматоре дополнительного кода получается прибавлением поправки $[A]$ к произведению дополнительных кодов сомножителей.

Пример 5.4. Умножить на сумматоре дополнительного кода по методу 2 с использованием структурной схемы примера 5.1 числа $A = -0{,}10111$ и $B = -0{,}11001$.

Решение. Сначала запишем машинные изображения чисел: $[A]^м_д = 11{,}01001$. $[B]^м_д = 11{,}00111$; $[\bar{A}]^м_д = 00{,}10111$.

Последовательность действий, проводимых над числами, показана в таблице 5.4.

Ответ $C = AB = 0{,}1000111111$.

Таблица 5.4

Сумматор	Регистр B	Примечания
00,00000	00111	И.П. СМ := 0 ; РгB := $[B'_м]_д$; [РгA] := $[A]^м_д$;
+ 11,01001		$\bar{b}_5 = 1$; СМ := [СМ] + $[A]^м_д$;
11,01001		
11,10100	→ 10011	$[\overrightarrow{\text{СМ}}]$; $[\overrightarrow{\text{Рг}B}]$;
+ 11,01001		$\bar{b}_4 = 1$. СМ := [СМ] + $[A]^м_д$;
10,11101		
11,01110	→ 11001	$[\overrightarrow{\text{СМ}}]$; $[\overrightarrow{\text{Рг}B}]$;
+ 11,01001		$\bar{b}_3 = 1$; СМ := [СМ] + $[A]^м_д$;
10,10111		$[\overrightarrow{\text{СМ}}]$; $[\overrightarrow{\text{Рг}B}]$;
1101011	→ 11100	$\bar{b}_2 = 0$; $[\overrightarrow{\text{СМ}}]$; $[\overrightarrow{\text{Рг}B}]$;
11,10101	→ 11110	$\bar{b}_1 = 0$; $[\overrightarrow{\text{СМ}}]$; $[\overrightarrow{\text{Рг}B}]$;
11,11010	→ 11111	
+ 00,10111		СМ := [ССМ+ $[\bar{A}]^м_д$
00,10001	11111	Конец

Таким образом, на сумматоре дополнительного кода в процессе перемножения машинных изображений операндов получаем одновременно знаковую и цифровую части произведения.

5.5. Умножение чисел на двоичном сумматоре обратного кода

Рассмотрим правила умножения операндов, заданных в обратном коде.

Произведение обратных кодов сомножителей равно обратному коду результата только в случае положительного множителя.

Пусть множимое $A = [A]_{об}$, а множитель $B > 0$. Тогда

$$AB = [A]_{об} \cdot 0, b_1 b_2 \ldots b_n = [A]_{об} b_1 \cdot 2^{-1} + [A]_{об} b_2 \cdot 2^{-2} + \ldots + [A]_{об} b_n \cdot 2^{-n}.$$

По теореме о сложении обратных кодов в правой части данного уравнения получается обратный код результата.

Следовательно, умножение на сумматоре обратного кода также заключается в анализе разрядов множителя, и если оказывается, что очередной разряд множителя равен единице, то к содержимому сумматора добавляется обратный код множимого.

Пример 5.5. Умножить на сумматоре обратного кода (структурная схема взята из примера 5.1) числа $A = -0{,}10011$ и $B = 0{,}11001$.

Решение Сначала записываются машинные изображения чисел:

$$[A]^{м}_{об} = 11{,}01100;\ [B]^{м}_{об} = 00{,}11001.$$

Последовательность действий, проводимых над числами, представлена в таблице 5.5.

Ответ: $[C]^{м}_{об} = 11{,}1000100100;\ C = AB = -0{,}0111011011.$

Таблица 5.5

Сумматор	Регистр B	Примечания
11,11111	11001	И.П. СМ := 0; PrA = $[A]^{м}_{об}$; PrB = $[B']^{м}_{об}$;
+ 11,01100		$b_5 = 1$; СМ := [СМ] + $[A]^{м}_{об}$;
11,01100		$[\overrightarrow{СМ}]$; $[\overrightarrow{PrB}]$;
11,10110	→ 01100	$b_4 = 0$; $[\overrightarrow{СМ}]$; $[\overrightarrow{PrB}]$;
11,11011	→ 00110	$b_3 = 0$; $[\overrightarrow{СМ}]$; $[\overrightarrow{PrB}]$;
11,11101	→ 10011	$b_2 = 1$; СМ := [СМ] + $[A]^{м}_{об}$;
+ 11,01100		
11,01010	→ 10011	
11,10101	→ 10011	$[\overrightarrow{СМ}]$; $[\overrightarrow{PrB}]$;
+ 11,01100		$b_1 = 1$; СМ := [СМ] + $[A]^{м}_{об}$;
11,00010	01001	$[\overrightarrow{СМ}]$; $[\overrightarrow{PrB}]$;
11,10001	→ 00100	Конец

Пусть $A = [A]_{об}$ и $B < 0$. Тогда $[B]_{об} = 1, b_1 b_2 \ldots b_n$. В соответствии с (3.28) $[B]_{об} = 2 + B - 2^{-n}$. Следовательно, $B = 0, b_1 b_2 \ldots b_n + 2^{-n} - 1$.

$$AB = [A]_{об} \cdot 0, b_1 b_2 \ldots b_n + [A]_{об} \cdot 2^{-n} + \overline{A}. \qquad (5.8)$$

На основании (5.8) можно сформулировать правило:

Если множитель отрицательный, то произведение чисел на сумматоре обратного кода получается прибавлением поправок $[A]$ и $[A]_{об} \cdot 2^{-n}$ к произведению обратных кодов сомножителей.

Пример 5.6. Умножить на сумматоре обратного кода (используется метод 2 структурная схема из примера 5.1) числа $A = -0{,}110101$ и $B = -0{,}101000$.

Р е ш е н и е. Сначала записываются машинные изображения чисел:

$$[A]^{м}_{об} = 11{,}001010;$$

$$[B]^{м}_{об} = 11{,}010111;$$

$$[\overline{A}]^{м}_{об} = 00{,}110101.$$

Последовательность действий, проводимых над числами, показана в таблице 5.6.

Ответ $AB = 00{,}100010$

Таблица 5.6

Сумматор	Регистр B	Примечания
11,111111	010111	И П СМ := 0; PrA = $[A]^{м}_{об}$; PrB = $[B']^{м}_{об}$;
+ 11,001010		Добавление $[A]^{м}_{об}$ в СМ, т. е. СМ := [СМ] + $[A]^{м}_{об}$,
11,001010		
+ 11,001010		$b_6 = 1$; СМ := [СМ] + $[A]^{м}_{об}$;
10,010101		
11,001010	→ 101011	$[\overline{СМ}]$; $[\overrightarrow{PrB}]$;
+ 11,001010		$b_5 = 1$; СМ := [СМ] + $[A]^{м}_{об}$;
10,010101		
11,001010	→ 110101	$[\overline{СМ}]$; $[\overrightarrow{PrB}]$;
+ 11,001010		$b_4 = 1$; СМ := [СМ] + $[A]^{м}_{об}$;
10,010101		
11,001010	→ 111010	$[\overline{СМ}]$; $[\overrightarrow{PrB}]$;
11,100101	→ 011101	$b_3 = 0$; $[\overline{СМ}]$; $[\overrightarrow{PrB}]$;
+ 11,001010		$b_2 = 1$; СМ := [СМ] + $[A]^{м}_{об}$;
10,110000		
11,011000	→ 001110	$[\overline{СМ}]$; $[\overrightarrow{PrB}]$;
11,101100	→ 000111	$b_1 = 0$; $[\overline{СМ}]$; $[\overrightarrow{PrB}]$;
+ 00,110101		СМ = [СМ] + $[A]^{м}_{об}$;
00,100010		Конец

Таким образом, в общем случае на сумматоре обратного кода произведение получается сразу со знаком и длиной в n разрядов, так как на последнем шаге умножения прибавляются числа разных знаков, из-за чего нельзя к результату приписать так называемый «хвост», хранящийся в регистре множителя.

5.6. Метод сокращенного умножения

В специализированных машинах иногда используют метод сокращенного умножения, начиная со старших разрядов. Особенность этого метода состоит в том, что в произведении получается только n старших разрядов. Существуют разные пути выполнения сокращенного умножения. Рассмотрим наиболее распространенные.

В некоторых машинах реализован метод, который можно объяснить с помощью следующей схемы.

Полная схема

$$
\begin{array}{rl}
A = & 0\,0,1\,0\,0\,1\,1 \\
\times & \\
B = & 0\,0,1\,1\,0\,0\,1 \\
\hline
 & 0\,0\,1\,0\,0\,1\,1 \\
+ & \\
b_1 = 1 & 0\,0\,1\,0\,0\,1\,1 \\
b_2 = 1 & 0\,1\,1\,1\,0\,0\,1 \\
+ & \\
b_3 = 0 & 0\,0\,0\,0\,0\,0\,0 \\
b_4 = 0 & 0\,1\,1\,0\,0\,1\,0 \\
+ & \\
b_5 = 1 & 0\,0\,0\,0\,0\,0\,0 \\
 & 0\,1\,0\,0\,1\,0\,0 \\
+ & \\
 & 0\,0\,1\,0\,0\,1\,1 \\
\hline
 & 0\,0\,1\,1\,0\,1\,1
\end{array}
$$

Сокращенная схема

$$
\begin{array}{rl}
 & 0\,0 \\
 & 0\,1 \\
+ & 0\,1 \\
 & 0\,1 \\
 & 0\,0 \\
\hline
 & 0,0\,1\,1\,1\,1\,0
\end{array}
$$

Таким образом, $AB = 0{,}01110 \cdot 2^5$.

Из этой схемы видно, что произведение получается, если складывать только старшие два разряда от сумм, полученных на каждом шаге умножения (показано справа). В этом случае достаточно иметь только n-разрядный сумматор. Однако следует учитывать, что для получения n точных знаков в произведении необходимо ввести дополнительные разряды.

В самом деле, пусть $A = 0, a_1 a_2 \ldots a_n$ и $B = 0, b_1 b_2 \ldots b_n$.

Тогда, если все $b_i = 1$, то

$$
\begin{array}{ll|l}
b_1 & 1 & a_1 a_2 \ldots a_n & \\
b_2 & 1 & a_1 a_2 \ldots a_{n-1} & a_n \\
b_3 & 1 & a_1 a_2 \ldots a_{n-2} & a_{n-1} a_n \\
\vdots & & \ddots & \\
b_n = 1 & & \underbrace{a_1}_{n \text{ разрядов}} & \underbrace{a_2 \ldots\ldots a_n}_{n-1 \text{ разрядов}}
\end{array}
$$

Предположим, что все разряды, стоящие справа от черты, отбрасываются. Если суммировать только n разрядов, то вносится погрешность, так как не учитываются переносы из отброшенных разрядов в разряды слева от вертикальной черты. Эти переносы могут распространиться на k разрядов слева от черты. Предполагается, что каждый разряд дает перенос, равный единице в каждом такте суммирования. Пусть все $a_i = 1$. Тогда количество единиц переноса из отброшенной части будет равно

$$\Delta_{max} = \frac{n-1}{2} + \frac{n-2}{2^2} + \frac{n-3}{2^3} + \ldots + \frac{n(n-1)}{2^{n-1}}. \qquad (5.9)$$

При достаточно больших значениях n (5.9) может быть записано так:

$$\Delta_{max} \approx n - 2. \qquad (5.10)$$

Единицы переноса распространятся на x разрядов сумматора, и произведение будет содержать только $n - x$ точных разрядов.

Следовательно, чтобы получить результат с точностью до n разрядов, необходимо выделить $n + x$ разрядов в сумматоре. Количество дополнительных разрядов при этом можно приближенно оценить величиной

$$x = \lg(n-2)/\lg 2. \qquad (5.11)$$

Ниже приведены результаты расчета по (5.11):

n	24	32	40	48	64	72
x	4,5	4,9	5,3	5,5	6,0	6,2

5.7. Ускорение операции умножения

В программах решения различных задач операции умножения встречаются достаточно часто. Поэтому методам выполнения умножения, его ускорению и рациональному построению множительных устройств всегда уделяется большое внимание.

По времени выполнения операцию умножения относят к длинным операциям. Так, затраты времени на умножение двух чисел в прямом коде можно оценить следующей формулой (для случая последовательного анализа разрядов множителя):

$$t_{умн} = \sum_{i=1}^{n} (t_{сдв} + p_i t_{сл}), \qquad (5.12)$$

где $t_{слв}$ – время выполнения сдвига числа на один разряд; $t_{сл}$ — время суммирования на сумматоре; p_i — вероятность появления единицы в разрядах множителя; n – количество разрядов множителя.

Аналогичные формулы можно написать и для других методов умножения.

Анализируя (5.12), можно наметить следующие пути сокращения времени умножения: уменьшение затрат времени на сдвиг и суммирование операндов; уменьшение количества слагаемых в формуле, т. е. уменьшение разрядов множителя n. Этого можно добиться логическими или аппаратными (схемными) средствами. В дальнейшем тексте будет обращено внимание на логические средства.

Рассмотрим возможности изменения величин p_i и $t_{сл}$.

Наиболее простой способ изменения величин p_i и $t_{сл}$ — пропуск тактов суммирования в случаях, когда очередная цифра множителя равна нулю. Этот способ может быть применен для систем счисления, содержащих нуль как одну из цифр, например, для систем (0, 1), (0, 1, $\bar{1}$) и т. п. Исключением в этом отношении являются симметричные системы (1, $\bar{1}$) и им подобные. Однако от системы (1, $\bar{1}$) можно осуществить переход к системе (0, 1) или (0, 1, $\bar{1}$), используя соотношения

$$1\,\bar{1}\,\bar{1}\,\bar{1}\ldots\bar{1} = 000\ldots1. \qquad (5.13)$$

Такая возможность часто имеется независимо от длины последовательностей нулей или единиц. Для ускорения операции умножения можно использовать также наличие последовательностей 0 или 1 (или $\bar{1}$). Например, последовательность вида $\underbrace{0100\ldots0}_{k}$ дает возможность сразу осуществить сдвиг на k разрядов, не проводя операции, а последовательность вида $\ldots\underbrace{011\ldots1}_{k}$ переходом к избыточной системе (0, 1, $\bar{1}$) может быть заменена на последовательность вида $\ldots\underbrace{100\ldots\bar{1}}_{k}$, что также приводит к уменьшению количества операций сложений.

Таким образом, переход от одной разновидности двоичной системы счисления к другой при преобразовании множителя позволяет получить выигрыш во времени выполнения операции в целом.

При этом возникают определенной длины последовательности нулей или единиц, что приводит к необходимости одновременного анализа нескольких разрядов множителя и сдвигу на произвольное число разрядов.

Рассмотрим простую иллюстрацию этого положения. Пусть множитель $B = 0{,}0111100011101111000$ необходимо преобразовать таким образом, что-

бы получить меньшее количество единиц в его изображении. Если представить этот множитель в системе счисления (0, 1, $\overline{1}$), то получим наименьшее число единиц в его изображении: $B = 0{,}1000\,\overline{1}\,00100\,\overline{1}\,100\,\overline{1}\,000$ (содержит только шесть единиц вместо десяти). Проведя замену $\overline{1}\,1$ на $0\,\overline{1}$, получим окончательно $B = 0{,}1000\,\overline{1}\,001000\,\overline{1}\,00\,\overline{1}\,000$.

Чтобы оценить относительную эффективность того или иного логического метода ускорения умножения, можно воспользоваться формулой

$$\sigma = \frac{\sum_{j=1}^{m}(t_{\text{слв}} + p_j t_{\text{сл}})}{\sum_{j=1}^{n}(t_{\text{слв}} + p_i t_{\text{сл}})}, \qquad (5.14)$$

где m — число шагов при выполнении операций умножения ускоренным методом (количество групп для анализа); p_i, p_j — вероятности появления в анализируемой i-й или j-й группе разрядов; σ — коэффициент эффективности.

Если $\sigma < 1$, то применяемый метод дает эффект; если же $\sigma \geq 1$ — эффекта не будет.

Группировка разрядов множителя и анализ этих групп позволяют предложить несколько методов ускорения операции умножения.

Анализ двух разрядов множителя одновременно. На основании вышеизложенного можно составить следующие правила преобразования разрядов множителя: $B = 0, b_1, b_2 \ldots b_n$.

Разбиение множителя на группы длиной k разрядов означает переход к новой системе счисления с основанием $q = 2^k$. Если при этом удается сократить количество элементарных действий, выполняемых при операции умножения (сложений и сдвигов), то операция умножения ускоряется. Рассмотрим метод на примере одновременного анализа двух разрядов множителя, начиная с младших разрядов.

Основой для правил преобразования разрядов множителя служат следующие соображения.

Комбинации вида 00 и 01 не преобразуются. Комбинация вида 10 означает, что необходим предварительно сдвиг множимого влево, сложение и затем сдвиг содержимого сумматора вправо на два разряда. Комбинация вида 11 заменяется на комбинацию вида 101, что означает запоминание единицы для следующей анализируемой пары цифр множимого и вычитание на данном шаге с последующим сдвигом на два разряда.

В таблице 5.7 представлены правила преобразования множителя для системы $(0, 1, \bar{1})$.

Таблица 5.7

Анализируемая пара разрядов $b_{i+1}b_i$	Перенос из предыдущей пары разрядов	Преобразованная пара разрядов $b'_{i+1}b'_i$	Примечание
00	0	00	
01	0	01	
10	0	10	
11	0	$0\bar{1}$	Предварительный сдвиг множимого
00	1	01	Запоминается единица для следующей пары разрядов
01	1	10	
10	1	$0\bar{1}$	Предварительный сдвиг множимого
11	1	00	Запоминается единица для следующей пары разрядов

В результате преобразования множителя меняется характер действий, которые должны выполняться при операции умножения, т. е. имеют место операции сложения и вычитания. Следовательно, такой способ умножения может быть реализован только на сумматорах обратного или дополнительного кода. Основной выигрыш во времени получается за счет того, что на каждом шаге умножения проводится либо сложение, либо вычитание множимого и сдвиг на два разряда одновременно. Особенность — дополнительный сдвиг множимого в противоположную сторону в двух случаях из восьми и запоминание единицы для передачи в следующую пару разрядов множителя. Естественно, это потребует усложнения устройства местного управления умножением.

Пример 5.7. Умножить два числа с одновременным анализом двух разрядов множителя, начиная с младших разрядов на сумматоре дополнительного кода: $A = 0{,}10011001$, $B = 0{,}10110110$.

Решение. При умножении чисел потребуются следующие значения: $[A]^{м}_{д} = 00{,}10011001$; $[\bar{A}]^{м}_{д} = 11{,}01100111$; $[\vec{A}]^{м}_{д} = 01{,}00110010$.

Преобразуем множитель в соответствии с правилами таблицы 5.7.

$$[B]_{п} = 01{,}0\bar{1}0\bar{1}0110.$$

В сумматоре выполняются следующие действия:

00,00000000	анализ 1-й пары — 10:
+ 01,00110010	СМ := [СМ] + $[\overline{A}]^{M}_{д}$;
01,00110010	
00,0100110010	сдвиг на 2 разряда;
+ 00,10011001	анализ 2-й пары — 01:
	СМ := [СМ] + $[A]^{M}_{д}$.
00,1100010110	
00,001100010110	сдвиг на 2 разряда;
+ 11,01100111	анализ 3-й пары — $0\overline{1}$:
	СМ := [СМ] + $[\overline{A}]^{M}_{д}$
11,1010000000110	
11,1110100000000110	сдвиг на 2 разряда;
+ 11,01100111	анализ 4-й пары — $0\overline{1}$:
	СМ := [СМ] + $[\overline{A}]^{M}_{д}$
11,0100111000110	
11,1101001111000110	сдвиг на 2 разряда;
+ 00,10011001	анализ 5-й пары — 01:
	СМ := [СМ] + $[\overline{A}]^{M}_{д}$;
00,0110100111000110	Конец.

Ответ $C = 0{,}0110100111000110$.

Как видно из примера 5.7, количество шагов при умножении на единицу превышает количество пар разрядов в цифровой части множителя, так как предусматривается возможность появления 1 в знаковой части.

При одновременном анализе двух разрядов, начиная со старших, правила преобразования существенно изменяются. Прежде всего на характер действий влияет значение соседнего справа разряда по отношению к анализируемой паре разрядов. Если его содержимое равно нулю, комбинации вида 00 и 01 выполняются, как в предыдущем случае. Комбинация вида 10 заменяется равнозначной ей комбинацией вида $1\overline{1}0$, что означает предварительный сдвиг множимого и вычитание его. Комбинация вида 11 заменяется комбинацией вида $10\overline{1}$, что означает вычитание множимого на данном шаге. В случае, если значение соседнего справа разряда равно единице, происходит изменение преобразуемых состояний и выполнение действий в соответствии с таблицей 5.8.

При этом анализ надо начинать с пустой пары.

Таблица 5.8

Анализируемая пара разрядов $b_j b_{j+1}$	Соседний справа разряд b_{j+2}	Преобразованная пара разрядов $b'_j b'_{j+1}$	Примечание
00	0	00	
01	0	01	
10	0	$\bar{1}0$	Предварительный сдвиг множимого
11	0	$0\bar{1}$	
00	1	01	Предварительный сдвиг множимого
01	1	10	
10	1	$0\bar{1}$	
11	1	00	

Пример 5.8. Умножить два числа на сумматоре обратного кода с одновременным анализом двух разрядов множителя, начиная со старших разрядов: $A = 0{,}10010001$; $B = 0{,}10111101$.

Решение. При умножении потребуются следующие величины:

$[A]^{м}_{об} = 00{,}10010001$, $[\overleftarrow{A}]^{м}_{об} = 01{,}00100010$, $[\overline{A}]^{м}_{об} = 11{,}01101110$, $[\overleftarrow{\overline{A}}]^{м}_{об} = 10{,}11011101$.

В сумматоре выполняются следующие действия:

$$[B]_п = 01,\bar{1}0100\bar{1}01.$$

	00,0000000000000000	Сдвиг 1-й пары — 01;
+	00,0000000010010001	
	00,0000000010010001	
	00,0000001001000100	сдвиг на 2 разряда;
+	11,1111111011011101	анализ 2-й пары — $\bar{1}0$;
	00,0000000100100010	
	00,0000010010001000	сдвиг на 2 разряда;
+	00,0000000100100010	анализ 3-й пары — 10;
	00,0000010110101010	
	00,0001011010101000	сдвиг на 2 разряда;
+	11,1111111101101110	анализ 4-й пары — $0\bar{1}$;
	00,0001011000101111	
	00,0101100001011100	сдвиг на 2 разряда;
+	00,0000000010010001	анализ 5-й пары — 01;
	00,0101100011101101	Конец.

Ответ $C = 00{,}0101100011101101$.

Анализ произвольного количества разрядов множителя. Идея метода состоит в том, что выявляются последовательности нулей или единиц и затем проводится групповая обработка разрядов множителя. В случае, если встречается группа вида $\ldots 01\underbrace{00\ldots0}_{k}$, то проводится сразу сдвиг на k разрядов и прибавление множимого в сумматор. Если анализируется группа $\ldots 11\underbrace{00\ldots0}_{k}$, то проводится сдвиг на k разрядов и вычитается множимое из содержимого сумматора. При анализе группы разрядов вида $\ldots 00\underbrace{11\ldots1}_{k}$ проводится замена ее на новую группу вида $\ldots\underbrace{00\ldots\overline{1}}_{k}$, что означает вычитание множимого на первом шаге, сдвиг на k разрядов и аиализ группы следующих разрядов, образовавшейся после преобразования. Такой метод ускорения операции умножения требует создания сдвигающего устройства, обладающего возможностью одновременного сдвига на произвольное количество разрядов.

Конечно, если бы числа состояли из достаточно длинного ряда последовательностей 0 или 1, то такой метод дал бы высокий эффект в смысле сокращения времени на обработку. Однако, как правило, в разрядах чаще чередуются 0 или 1. Это значит, что в подобных случаях рассмотренный выше метод не дает никакого эффекта. Поэтому он может быть применен только в комбинации с обычными методами последовательного умножения.

Умножение в системе счисления с основанием $q = 2^k$. В некоторых машинах ЕС ЭВМ использовался переход на новое основание системы счисления вида $q = 2^k$, за счет чего удается существеиио сократить время выполнения операции умножения.

Пусть $k = 4$; это означает, что числа представляются в шестнадцатеричной системе счисления. При этом с помощью приема, аналогичного приему, использованному в случае одновремеиного анализа двух разрядов множителя, можно анализировать сразу тетраду (четыре двоичных разряда): рассматриваются текущая цифра (тетрада) множителя и его предыдущая цифра (тетрада). В зависимости от значений цифры миожителя в предыдущем разряде (равна или больше восьми, т. е. равен или нет единице старший разряд тетрады) проводятся разные действия (табл. 5.9). Для реализации этого приема требуется также предварительио готовить множимое A, увеличивая его в 1, 2, 3 и 6 раз.

Анализ четырех двоичных разрядов одновременно дает возможность сразу осуществить сдвиг на четыре двоичных разряда.

Таблица 5.9

Анализируемая цифра	Анализируемая тетрада	Действия на сумматоре при значении предыдущей цифры	
		≥ 8	< 8
0	0000	$+1A$	0
1	0001	$+2A$	$+1A$
2	0010	$+3A$	$+2A$
3	0011	$+(2A+2A)$	$+3A$
4	0100	$+(3A+2A)$	$+(2A+2A)$
5	0101	$+6A$	$+(2A+3A)$
6	0110	$+(6A+1A)$	$+6A$
7	0111	$+(6A+2A)$	$+(6A+1A)$
8	1000	$-(1A+6A)$	$+(6A+2A)$
9	1001	$-6A$	$-(6A+1A)$
(A) 10	1010	$-(3A+2A)$	$-6A$
(B) 11	1011	$-(2A+2A)$	$-(3A+2A)$
(C) 12	1100	$-3A$	$-(2A+2A)$
(D) 13	1101	$-2A$	$-3A$
(E) 14	1110	$-1A$	$-2A$
(F) 15	1111	0	$-1A$

Так как при операции умножения необходимо складывать и вычитать коды, то потребуется сумматор дополнительного или обратного кода.

Пример 5.9. Умножить два числа $A = 0{,}0001101$ и $B = 0{,}00001110$ с одновременным анализом четырех разрядов множителя. Использовать сумматор дополнительного кода.

Решение. Для умножения потребуются числа: $A = 0{,}00001101$, $2A = 0{,}00011010$, $3A = 0{,}00100111$, $6A = 0{,}01001110$.

Множитель представляется в дополнительном коде $[B]_д = 1,\underbrace{1110}_{b_1}\underbrace{0010}_{b_2}$.

Шаг 1. Анализ первой цифры множителя для $q = 2^4$ (предыдущая цифра равна нулю): $b_2 = 0010$ — на основании таблицы 5.9 надо вызвать множимое $+2A = 0{,}00011010$ и направить его в сумматор. Содержимое сумматора станет 0,00011010.

Шаг 2. Анализ второй цифры множителя (предыдущая цифра меньше восьми): $b_1 = \overline{1\ 1\ 1\ 1}$ – на основании таблицы 5.9 надо вызвать множимое $-A_1 = 1{,}11110011$, сдвинуть его на четыре разряда влево и направить в сумматор, где проводится следующее действие:

$$\begin{array}{r} 0{,}00011010 \\ +\,1{,}00110000 \\ \hline 1{,}01001010 \end{array}$$

Ответ $AB = -0{,}10110110$.

5.8. Матричные методы умножения

Существует ряд методов умножения, основанных на суммировании групп частных произведений с последующим объединением сумм вместе с переносами для получения произведения. Например, частные произведения группируются по три и подаются на входы цепочки сумматоров. Выходы цепочки сумматоров подключаются к регистрам, запоминающим отдельно получившиеся сумму и переносы в другие группы. Выходы этих регистров объединяются в группы по три и подключаются уже к другим сумматорам. В конце цепочки складываются только сумма и переносы для слагаемых. Такая раздельная обработка промежуточных сумм и переносов требует так называемого «дерева сумматоров». Подобный метод умножения использован в вычислительных машинах IBM-360.

Существуют также ускоренные методы умножения, основанные на использовании матриц промежуточных результатов. Рассмотрим схему умножения на примере двух пятиразрядных чисел:

$$\begin{array}{r} \times\ A = a_5a_4a_3a_2a_1 \\ B = b_5b_4b_3b_2b_1 \\ \hline a_5b_1a_4b_1a_3b_1a_2b_1a_1b_1 \\ +\ \cdots\cdots\cdots\cdots \\ a_5b_5a_4b_5a_3b_5a_2b_5a_1b_5 \\ \hline C = c_{10}\ \ c_9 \cdots\cdots\cdots\cdots c_2\ \ c_1 \end{array}$$

Эту схему умножения можно представить также в виде матрицы (табл. 5.10).

Каждый элемент этой матрицы равен 0 или 1. Произведение двух чисел можно получить, если суммировать элементы матрицы в следующем порядке:

$$\begin{array}{c|c|c|c} \cdots & a_3b_1 & a_2b_1 & a_1b_1 \\ & + & + & \\ \cdots & a_2b_2 & a_1b_2 & \\ & + & & \\ \cdots & a_1b_3 & & \end{array}$$

Таблица 5.10

b_i	a_i				
	a_5	a_4	a_3	a_2	a_1
b_1	a_5b_1	a_4b_1	a_3b_1	a_2b_1	a_1b_1
b_2	a_5b_2	a_4b_2	a_3b_2	a_2b_2	a_1b_2
b_3	a_5b_3	a_4b_3	a_3b_3	a_2b_3	a_1b_3
b_4	a_5b_4	a_4b_4	a_3b_4	a_2b_4	a_1b_4
b_5	a_5b_5	a_4b_5	a_3b_5	a_2b_5	a_1b_5

Так как при суммировании по столбцам складываются только 0 и 1, операцию сложения можно выполнить с помощью счетчиков. Однако при значительном числе разрядов сомножителей потребуются счетчики с большим количеством входов и выходов, что существенно увеличивает время суммирования. Но этот принцип умножения можно реализовать таким образом, что количество входов счетчиков на каждом этапе будет не больше трех. Значит, для этих целей можно использовать одноразрядные двоичные полусумматоры и сумматоры.

Наиболее известны среди матричных алгоритмов умножения алгоритмы Дадда, Уоллеса, матричный и алгоритм с сохранением переносов. На рис. 5.4 представлена структурная схема множительного устройства для реализации матричного алгоритма, а на рис. 5.5 — схема, с помощью которой может быть реализован алгоритм Дадда. Как видно из рисунков, алгоритмы отличаются друг от друга не только группировкой частных произведений, но и количеством ступеней преобразования: для матричного алгоритма количество ступеней преобразования равно четырем, т. е. на единицу меньше числа разрядов сомножителей, а для алгоритма Дадда — трем. Но с другой стороны, группировка разрядов по методу Дадда требует более сложного устройства управления операцией умножения.

Реализация матричных методов выполнения операции умножения требует большего количества оборудования, чем методов последовательного анализа разрядов или групп разрядов множителя и дает больший выигрыш во времени. Однако в связи с широким развитием микроэлектронных и особенно больших интегральных схем (БИС) ограничения по количеству оборудования становятся все менее строгими, поэтому рассмотренные выше методы применяют на практике.

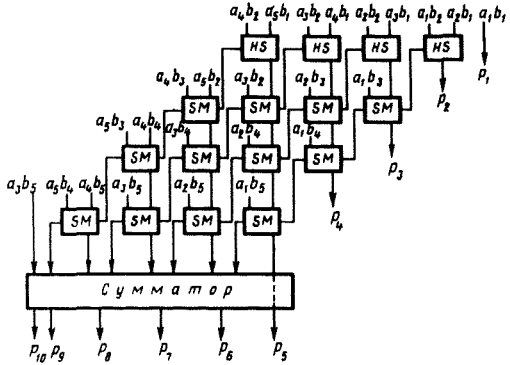


Рис. 5.4. Структурная схема матричного умножителя

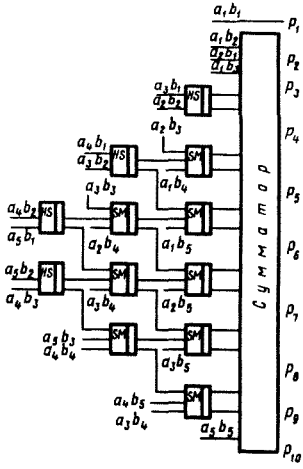


Рис. 5.5. Структурная схема множительного устройства с запоминанием переносов

5.9. Методы параллельного умножения с использованием итеративных структур

Время, затрачиваемое на выполнение операции умножения, можно существенно уменьшить, воспользовавшись методами параллельного умножения. Конечно, методы последовательного выполнения операции обеспечиваются более простыми схемами. Однако в конкретной практике существует множество задач, для решения которых отводится весьма мало времени. Это задачи, выполняемые с использованием метода быстрого преобразования Фурье, матричные задачи и другие задачи, решаемые в реальном масштабе времени. Поэтому, несмотря на то, что параллельные умножители гораздо сложнее и дороже устройств, построенных на традиционных методах умножения, разработчики все чаще обращаются к созданию именно параллельных структур*. При этом очень широко используются так называемые ячеистые (cellular) или итеративные элементы. По существу, с помощью итеративных элементов параллельно образуются несколько частных произведений с соответствующими весовыми коэффициентами, которые тут же суммируются, и определяется полное произведение.

Типовой итеративный элемент (ТИЭ) представлен на рис. 5.6*. Функционально ТИЭ может состоять из схемы U и полного сумматора SM с соответствующими входами для множимого A, множителя B, предыдущего частного произведения S_{i-1} и переноса из младшего разряда C_1 (C_2 — перенос в старший разряд).

Таким образом, на ТИЭ реализуются следующие операции:

$$\begin{aligned} S_i &= S_{i-1} + C_1 + [A \cdot B]; \\ C_2 &= S_{i-1} \cdot C + [S_{i-1} + C_1] \cdot A \cdot B. \end{aligned} \qquad (5.15)$$

Параллельный умножитель для четырехразрядных двоичных чисел представлен на рис. 5.7. Каждый ТИЭ, представленный в структуре на рис. 5.7, служит для получения одного частного произведения, к которому прибавляются частные произведения, имеющие одинаковый с ним весовой коэффициент. Полученная сумма S_i подается на следующий элемент того же веса, а перенос C_2 — на соседний слева элемент.

---

* В первой советской ЭВМ «Стрела» было реализовано параллельное множительное устройство, основанное на комбинационных схемах, что по тем временам являлось передовым решением.

* Наибольший вклад в создание и разработку итеративных структур внесли американские специалисты Дж. Спрингер, Н. Алфке, Д. Аграваль и др.

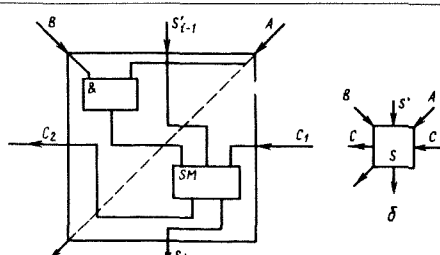


Рис. 5.6. Схема типового итеративного элемента

Рис. 5.7. Параллельный умножитель

Достоинство итеративных логических структур состоит и в том, что они позволяют одновременно с умножением выполнять операцию сложения.

Основная проблема, решаемая при проектировании итеративных структур с помощью технологии интегральных схем, состоит в том, сколько функциональных элементов можно разместить на одном кристалле. Поэтому первые итеративные структуры были реализованы на кристаллах, вмещавших 4×2 двоичных разряда (бита). Развитие технологии производства самих интегральных схем позволяет уже сегодня создавать на одном кристалле структуры на 16×16 битов и более.

5.10. Систолический метод вычислений

Сущность систолического метода вычислений состоит в том, что в так называемых систолических процессорах информация «прогоняется через параллельные магистрали подобно тому, как сердце перекачивает кровь по системе кровообращения»*. В традиционных процессорах и итеративных структурах информация продвигается посимвольно через единственный процессорный элемент. Систолические процессоры содержат динамические перестраиваемые матрицы элементов, которые принимают потоки данных и пропускают каждый поток через определенную магистраль, содержащую отдельные процессоры. Изменять информационные магистрали в систолических процессорах значительно проще, чем в других параллельных структурах.

Рис. 5.8. Схема систолических вычислений

Рассмотрим работу систолического метода на примере перемножения двух двумерных матриц 3×2 и 2×2. Необходимая для этого примера систолическая структура представлена на рис. 5.8, *а*, *б*. Элементы матрицы A поступают слева сверху, а матрицы B — справа сверху. Стрелками показаны направления движения информации. Обработка матриц осуществляется несколькими последовательными шагами. Все ячейки этой структуры работают одновременно. На рис. 5.8, *а* показан третий шаг процесса перемножения матриц. На этом шаге получено произведение $a_{11}\cdot b_{11}$ (нижняя часть верхнего элемента), оно будет сложено с произведением $a_{12}\cdot b_{21}$, элементы

* Название метода дано его создателем проф. Х. Т. Куном из Университета Корнеги-Меллона (США).

которого поступили в верхнюю часть процессора. Пустая ячейка еще ие получила никакой информации. На рис. 5.8, *б* показан четвертый шаг, в результате которого появляется первый элемеит полного произведения матриц $c_{11} = a_{12} \cdot b_{21} + a_{11} \cdot b_{11}$. Весь процесс заиимает шесть шагов. Сложение и перемножение символов матриц продолжается до тех пор, пока все символы не пройдут через систолический процессор, формируя в его ячейках символы матрицы-результата.

Задание для самоконтроля

1. Перемножить на сумматоре прямого кода числа $A = -0{,}1100011$ и $B = -0{,}1011101$.

2. Перемножить в обратном и дополнительном кодах числа $A = -0{,}11$ и $B = -0{,}11$.

3. Преобразовать множитель $B = 0{,}110011101011$ в избыточную двоичную систему $(0, 1, \overline{1})$.

4. Перемножить по способу одновременного анализа двух разрядов множителя числа $A = 0{,}1100011101$ и $B = -0{,}1100100011$: а) начиная с младших разрядов; б) начиная со старших разрядов.

5. Перемножить комбинированным методом с одновременным анализом произвольного количества разрядов множителя числа $A = -0{,}100000001$ и $B = 0{,}011001110$.

6. Перемножить числа $A = 0{,}11001101$ и $B = 0{,}10001100$, использовав промежуточную систему счисления с $q = 2^4$ с одновременным анализом четырех разрядов множителя.

7. Перемножить числа $A = 0{,}1101101$ и $B = 0{,}1000111$, используя формулу $AB = 2AB/2$, если B — четное; $AB = 2A(B-1)/2 + A$, если B — нечетное.

8. Умножить числа $A = 0{,}100101 \cdot 2^5$ и $B = 0{,}110001 \cdot 2^{-3}$, если задана разрядность: семь двоичных разрядов со знаком для мантиссы и четыре двоичных разряда со знаком для порядка

6. ВЫПОЛНЕНИЕ ОПЕРАЦИЙ ДЕЛЕНИЯ ЧИСЕЛ НА ДВОИЧНЫХ СУММАТОРАХ

6.1. Методы деления двоичных чисел

Деление двоичных чисел во многом аналогично делению десятичных чисел. Процесс деления состоит в том, что последовательно разряд за разрядом отыскиваются цифры частного путем подбора с последующим умножением этой цифры на делитель и вычитанием этого произведения из делимого.

Из множества разных методов выполнения операции деления рассмотрим наиболее распространенные.

Прежде всего это — «школьный» алгоритм деления, заключающийся в том, что делитель на каждом шаге вычитается столько раз из делимого (начиная со старших разрядов), сколько это возможно для получения наименьшего положительного остатка. Тогда в очередной разряд частного записывается цифра, равная числу делителей, содержащихся в делимом на данном шаге. Таким образом, весь процесс деления сводится к операциям вычитания и сдвига.

Другой метод выполнения операции деления заключается в умножении делимого на обратную величину делителя. Здесь возникает новая операция — вычисление обратной величины, осуществляемое по известным приближенным формулам (например, разложением в биномиальный ряд Ньютона и т. п.). В этом случае в состав команд машины должна входить специальная операция нахождения обратной величины.

К наиболее распространенным методам выполнения операции деления относится также метод, заключающийся в использовании приближенной формулы для нахождений частного от деления двух величин. От метода умножения делимого на обратную величину он отличается только тем, что частное определяется по некоторой формуле, которая сводится к выполнению операций сложения, вычитания и умножения.

Фактически два последних метода пригодны для использования в специализированных машинах, в которых операция деления встречается не часто и ее целесообразно реализовать программным путем.

В универсальных вычислительных машинах, как правило, реализуется разновидность «школьного» алгоритма деления.

В общем случае «школьный» алгоритм деления на примере двоичных чисел выглядит следующим образом:

```
делимое —      1100100 | 1010   — делитель
делитель —     1010    | 1010   — частное
остаток —    - 0101
               1010
             - 1011
             +
восстановление остатка —  1010
                        01010
                      - 1010
                        0000
```

Здесь цифры частного получаются последовательно, начиная со старшего разряда путем вычитания делителя из полученного остатка. Если получен положительный остаток, то цифра частного равна единице; если остаток отрицательный, то цифра частного равна нулю, при этом восстанавливается предыдущий положительный остаток.

В случае положительного остатка для получения следующей цифры частного последний остаток сдвигается влево на один разряд (либо делитель вправо на один разряд) и из него вычитается делитель и т. д.

В случае отрицательного остатка восстанавливается предыдущий положительный остаток прибавлением к отрицательному остатку делителя, и восстановленный остаток сдвигается на один разряд влево (либо сдвигается делитель вправо на один разряд) и из него вычитается делитель. Такой алгоритм деления получил название алгоритма деления с восстановлением остатка. Формально все действия можно описать следующим образом.

Пусть A - делимое, B — делитель и C — частное, при этом $A = 0, a_1 a_2 \ldots a_n$; $B = 0, b_1 b_2 \ldots b_n$; $C = 0, c_1 c_2 \ldots c_n$.

Для реализации алгоритма деления двоичных чисел, представленных в форме с фиксированной запятой, необходимо, чтобы выполнялось условие $|A| < |B|$. Если это условие не выполняется, то в первом шаге возникает переполнение разрядной сетки и операция не выполняется. Если $|A| < |B|$, то на первом шаге операции проводится сдвиг делителя и определяется остаток $A_1 = A - B \cdot 2^{-1}$.

Пусть $A_1 > 0$, тогда $C_1 = 1$. Процесс деления продолжается дальше: $A_2 = A_1 - B \cdot 2^{-2}$.

Пусть $A<0$, тогда $C_2=0$ и проводится восстановление остатка A_1: $A_2=A_2+B\cdot 2^{-2}$.

Этот остаток принимается за A_2' и деление продолжается дальше следующим образом: $A_3=A_2'-B\cdot 2^{-3}$.

Таким образом, алгоритм деления можно описать в общем виде на i-м шаге:

$$A_i=A_{i-1}-B\cdot 2^{-i}. \qquad (6.1)$$

Если $A_i\ge 0$, то $C_i=1$ и переход к следующему шагу; если $A_i<0$, то $C_i=0$ и восстановление остатка

$$A_{i-1}=A_i+B\cdot 2^{-i}, \qquad (6.2)$$

который принимается за остаток A_i', и процесс продолжается по формуле (6.1). Следовательно, операция деления сводится к последовательному выполнению вычитании (сложений) в сумматоре и сдвигам делителя. Сдвиг делителя может быть заменен сдвигом содержимого сумматора в противоположную сторону.

Алгоритм деления, основанный на реализации формул (6.1) и (6.2), называется *делением с восстановлением остатка.*

Рассмотрим процесс восстановления остатка. $A_i<0$ и $C_i=0$, то на следующем шаге выполняются действия по формуле $A_{i+1}=A_i'-B\cdot 2^{-(i+1)}=$ $=A_i+B\cdot 2^{-i}-B\cdot 2^{-(i+1)}$. После преобразования получим

$$A_{i+1}=A_i+B\cdot 2^{-(i+1)}. \qquad (6.3)$$

Таким образом, появляется возможность построить алгоритм деления по следующей схеме: $A_i=A_{i-1}-B\cdot 2^{-i}$, если $A_i\ge 0$, то $C_i=1$ и переход к следующему шагу; если $A_i<0$, то $C_i=0$ и продолжение по формуле $A_{i+1}=A_i+B\cdot 2^{-(i+1)}$. При этом, если $A_{i+1}\ge 0$, то $C_{i+1}=1$ и переходим к формуле (6.1), если $A_{i+1}<0$, то $C_{i+1}=0$ и переходим к формуле (6.3). Такой алгоритм называется *делением без восстановления остатков.*

Реализация рассмотренных алгоритмов деления возможна с помощью двоичных сумматоров обратного или дополнительного кода. Для хранения делителя, делимого и накопления очередных разрядов частного выделяются регистры. Минимальный состав блоков, необходимый для выполнения делений, показан на рис. 6.1 в двух вариантах. Варианты отличаются спосо-

бом реализации сдвига. В варианте рис. 6.1, *а* осуществлен сдвиг содержимого регистра делителя, а в варианте рис. 6.1, *б* — сдвиг содержимого сумматора. Стрелки на рисунке указывают направление сдвига. Индекс (n) указывает ориентировочное количество разрядов в цифровой части сумматора и регистра делителя. В дальнейших разделах при выполнении примеров будет использована структура, показанная на рис. 6.1, *а*.

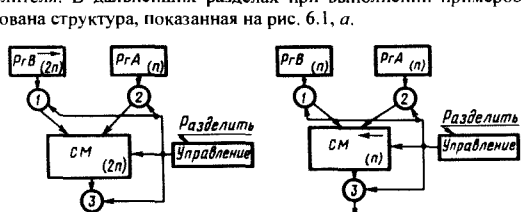


Рис. 6.1. Структурные схемы для реализации деления

6.2. Деление чисел, представленных в форме с фиксированной запятой, на сумматорах обратного и дополнительного кода

При делении знаковая и цифровая части частного получаются раздельно. Знак частного образуется по формуле

$$Sg_C = Sg_A \oplus Sg_B .$$

Для образования цифр частного воспользуемся следующим соответствием, в котором приведены логические действия над остатками и делителем:

Знак делимого A	+	+	–	–
Знак делителя B	+	–	+	–
Что делать в сумматоре. . . .	$A_i + \overline{B}\cdot 2^{-1}$	$A_i + B\cdot 2^{-1}$	$A_i + B\cdot 2^{-1}$	$A_i + \overline{B}\cdot 2^{-1}$

(Символ «-» указывает на изменение знака на противоположный на очередном шаге деления.)

На сумматорах обратного кода возможны алгоритмы получения частного в прямом или обратном коде. Это определяется правилами анализа остатка.

Пример 6.1. Разделить числа $A = -0{,}10011$ и $B = -0{,}11001$ на двоичном сумматоре обратного кода с получением частного в прямом коде.

Решение. Для реализации примера воспользуемся структурной схемой рис. 6.1, *а* и методом деления с восстановлением остатка.

Представляем числа в обратных кодах $[A]^{м}_{об} = 11{,}01100$; $[B]^{м}_{об} = 11{,}00110$; $[\overline{B}]^{м}_{об} = 00{,}11001$.

Определим знак частного: $Sg_C = Sg_A \oplus Sg_B = 1 \oplus 1 = 0$.

При этом руководствуемся следующими правилами:

Вариант	1	2	3	4
Знак делимого A	+	+	–	–
Знак делителя B	+	–	+	–
Что делать в CM	$A_{i-1} + \overline{B}\ 2^{-(i-1)}$	$A_{i-1} + B \cdot 2^{-(i-1)}$	$A_{i-1} + B \cdot 2^{-(i-1)}$	$A_{i-1} + \overline{B}\ 2^{-(i-1)}$
Знак остатка A_i	$A_i \geq 0\ A_i < 0$	$A_i \geq 0\ A_i < 0$	$A_i \geq 0\ A_i < 0$	$A_i \geq 0\ A_i < 0$
Цифра частного c_i	$c_i = 1\ c_i = 0$	$c_i = 1\ c_i = 0$	$c_i = 0\ c_i = 1$	$c_i = 0\ c_i = 1$

Последовательность действий для получения цифр частного показана в таблице 6.1

Ответ $[C_{пр}] = 0.11000$.

Таблица 6.1

Сумматор ←	Регистр C ←	Примечание
1101100	00000	И. П. СМ := $[A]^{м}_{об}$; PrB := $[B]^{м}_{об}$; PrC := 0;
1011001	0000–	$[\overleftarrow{CM}]$; $[\overleftarrow{PrC}]$;
+ 0011001		$[CM] := [CM] + [\overline{B}]^{м}_{об}$;
1110010	00001	$A_1 < 0$; $c_1 = 1$;
1100101	0001–	$[\overleftarrow{CM}]$; $[\overleftarrow{PrC}]$;
+ 0011001		$[CM] := [CM] + [\overline{B}]^{м}_{об}$;
1111110	00011	$A_2 < 0$; $c_2 = 1$;
1111101	0011–	$[\overleftarrow{CM}]$; $[\overleftarrow{PrC}]$;
+ 0011001		$[CM] := [CM] + [\overline{B}]^{м}_{об}$;
0010111	00110	$A_3 > 0$; $c_3 = 0$;
+ 1100110		Восстановление остатка
1111101		$[CM] := [CM] + [\overline{B}]^{м}_{об}$;
1111011	0110–	$[\overleftarrow{CM}]$; $[\overleftarrow{PrC}]$;
+ 0011001		$[CM] := [CM] + [\overline{B}]^{м}_{об}$;
0010101	11000	$A_4 > 0$; $c_4 = 0$;
+ 1100110		Восстановление остатка
1111011		$[CM] := [CM] + [\overline{B}]^{м}_{об}$;
1110111	1100–	$[\overleftarrow{CM}]$; $[\overleftarrow{PrC}]$;
+ 0011001		$[CM] := [CM] + [\overline{B}]^{м}_{об}$;
0010001	11000	$A_5 > 0$; $c_5 = 0$;
		$\Sigma_{сл} = 5$.
		Конец операции

По своему характеру операция деления относится к операциям, дающим не всегда точный результат, поэтому признаком окончания операции деления может быть достижение заданной точности (по сумме сдвиговых сигналов). Если в процессе деления получили остаток $A_i = 0$, то операция останавливается и в оставшиеся разряды частного записывается нуль. Обычно формальным признаком конца операции деления является количество сдвигов: при достижении числа сдвигов, равного количеству разрядов в частном, вырабатывается сигнал окончания операции деления.

Требуется несколько интерпретировать правила определения цифры частного. В примере 6.1 проводится сравнение знаков делимого и остатка на каждом шаге: при совпадении знаков в частном записывается единица, при несовпадении — нуль. Можно сравнивать знаки делителя и остатка, тогда единица в очередной разряд частного записывается при несовпадении знаков, а нуль — при совпадении.

Для получения частного, представленного в обратном коде, все действия должны осуществляться по следующим правилам:

Вариант	1		2		3		4	
Знак делимого	+		+		–		–	
Знак делителя	+		–		+		–	
Что делать в СМ	$A_{i-1} + \overline{B} \cdot 2^{-(i-1)}$		$A_{i-1} + B \cdot 2^{-(i-1)}$		$A_{i-1} + B \cdot 2^{-(i-1)}$		$A_{i-1} + \overline{B} \cdot 2^{-(i-1)}$	
Знак остатка A_i	$A_i \geq 0$	$A_i < 0$	$A_i > 0$	$A_i \leq 0$	$A_i > 0$	$A_i \leq 0$	$A_i \geq 0$	$A_i < 0$
Цифра частного c_i	$c_i = 1$ $\overline{c}_i = 1$	$c_i = 0$ $\overline{c}_i = 0$	$c_i = 0$ $\overline{c}_i = 1$	$c_i = 1$ $\overline{c}_i = 0$	$c_i = 1$ $\overline{c}_i = 0$	$c_i = 0$ $\overline{c}_i = 1$	$c_i = 0$ $\overline{c}_i = 1$	$c_i = 1$ $\overline{c}_i = 0$
		вос. ост.		вос. ост.	вос. ост.		вос. ост.	

Пример 6.2. Разделить числа $A = -0{,}10101$ и $B = 0{,}11100$ на двоичном сумматоре обратного кода с получением частного в обратном коде.

Решение. Структурная схема и метод деления те же, что в примере 6.1.

Представим числа в обратных кодах:

$$[A]^{м}_{об} = 11{,}01010;\ [B]^{м}_{об} = 11{,}00011;\ [\overline{B}]^{м}_{об} = 00{,}11100.$$

Знак результата $Sg_r = 1 \oplus 0 = 1$.

Последовательность действий при решении задачи показана в таблице 6.2

Ответ $[C]^{м}_{об} = 11{,}00111$.

Таблица 6.2

Сумматор ←	Регистр C ←	Примечание
1101010	00000	И. П. СМ := $[A]^{м}_{об}$; РгВ := $[B]^{м}_{об}$; РгС := 0;
1010101	0000–	$\overleftarrow{[CM]}$; $\overleftarrow{[РгС]}$;
+ 0011100		$[CM] := [CM] + [B]^{м}_{об}$;
1110001	00000	$A_1 < 0$; $c_1 = 0$;
1100011	0000–	$\overleftarrow{[CM]}$; $\overleftarrow{[РгС]}$;
+ 0011100		$[CM] := [CM] + [B]^{м}_{об}$;
1111111	00000	$A_2 < 0$; $c_2 = 0$; ($A_2 = 0$);
1111111	0000–	$\overleftarrow{[CM]}$; $\overleftarrow{[РгС]}$;
+ 0011100		$[CM] := [CM] + [B]^{м}_{об}$;
0011100	00001	$A_3 > 0$; $c_3 = 0$;
+ 1100011		Восстановление остатка
1111111		
1111111	0001–	$\overleftarrow{[CM]}$; $\overleftarrow{[РгС]}$;
+ 0011100		$[CM] := [CM] + [B]^{м}_{об}$;
0011100	00011	$A_4 > 0$; $c_4 = 1$;
+ 1100011		Восстановление остатка
1111111		
1111111	0011–	$\overleftarrow{[CM]}$; $\overleftarrow{[РгС]}$;
+ 0011100		$[CM] := [CM] + [B]^{м}_{об}$;
0011100	00111	$A_5 > 0$; $c_5 = 1$;
		$\Sigma_{сдв} = 5$. Конец

Затраты времени на выполнение операции деления можно определить по формуле

$$t_{дел} = n[t_{сдв} + (1 + p)t_{сл}], \qquad (6.4)$$

где n — длина разрядной сетки; $t_{сдв}$ — время выполнения сдвига на один разряд; $t_{сл}$ — время выполнения операции сложения; p — вероятность появления нулей в разрядах частного.

На двоичных сумматорах дополнительного кода можно получить частное как в прямом, так и в дополнительном кодах. Знак частного образуется отдельно от цифровой части путем сложения знаковых разрядов по модулю 2.

Для получения частного в прямом коде правила остаются теми же, что и для сумматоров обратного кода (см. пример 6.1).

Пример 6.3. Разделить число $A = 0{,}10001$ на $B = -11011$ на двоичном сумматоре дополнительного кода, частное получить в прямом коде. Используется структурная схема, представленная на рис. 6.1, *а*.

Р е ш е н и е . Последовательность действий при решении примера показана в таблице 6.3.

$[A]_{\text{п}}^{\text{м}} = 00{,}10001$; $[B]_{\text{п}}^{\text{м}} = 11{,}00101$; $[\overline{B}]_{\text{п}}^{\text{м}} = 00{,}11011$. Используем алгоритм без восстановления остатка. Знак частного: $Sg_C = 0 \oplus 1 = 1$.

Таблица 6.3

Сумматор ←	Регистр C ←	Примечание
0010001	00000	И. П. СМ := $[A]_{\text{п}}^{\text{м}}$; Рг$C$:= 0; РгB := $[B]_{\text{п}}^{\text{м}}$;
0100010	0000–	$\overleftarrow{[\text{CM}]}$; $\overleftarrow{[\text{Pr}C]}$;
1100101		[СМ] := [СМ] + $[B]_{\text{п}}^{\text{м}}$;
0000111	00001	$A_1 > 0$; $c_1 = 1$;
0001110	0001–	$\overleftarrow{[\text{CM}]}$; $\overleftarrow{[\text{Pr}C]}$;
1100101		[СМ] := [СМ] + $[B]_{\text{п}}^{\text{м}}$;
1110011	00010	$A_2 < 0$; $c_2 = 0$;
1100110	0010–	$\overleftarrow{[\text{CM}]}$; $\overleftarrow{[\text{Pr}C]}$;
0011011		[СМ] := [СМ] + $[B]_{\text{п}}^{\text{м}}$;
0000001	00101	$A_3 > 0$; $c_3 = 1$;
0000010	0101–	$\overleftarrow{[\text{CM}]}$; $\overleftarrow{[\text{Pr}C]}$;
1100101		[СМ] := [СМ] + $[B]_{\text{п}}^{\text{м}}$;
1100111	01010	$A_4 < 0$; $c_4 = 0$;
1001110	1010–	$\overleftarrow{[\text{CM}]}$; $\overleftarrow{[\text{Pr}C]}$;
0011011		[СМ] := [СМ] + $[\overline{B}]_{\text{п}}^{\text{м}}$;
1101001	10100	$A_5 < 0$; $c_5 = 0$;
		$\Sigma_{\text{сдв}} = 5$, Конец

Ответ: $[C]_{\text{пр}} = 1{,}10100$.

Алгоритм выполнения операции деления должен работать так, что если делитель равен 0 или делимое равно максимальному отрицательному числу, то вырабатывается сигнал переполнения.

Сигнал переполнения должен вырабатываться и в тех случаях, когда делимое больше делителя по абсолютной величине для чисел с фиксированной запятой.

6.3. Особенности деления чисел, представленных в форме с плавающей запятой

Для получения частного от деления двух чисел, представленных в форме с плавающей запятой, необходимо определить $m_C = m_A / m_B$, $p_C = p_A - p_B$.

Так как мантиссы делимого и делителя — нормализованные числа, при делении возможны случаи, когда $|m_A| \geq |m_B|$; $|m_A| < |m_B|$.

Если мантисса делимого больше или равна мантиссе делителя, то в конце операции деления потребуется нормализация частного (нарушение правой границы). Таким образом, алгоритм деления начинается с операции вычитания делителя из делимого и записи единицы в целую часть частного. Все остальные действия над мантиссами аналогичны действиям над числами, представленнымн в форме с фиксированной запятой.

Если мантисса делимого меньше мантиссы делителя, то после операции вычитания на первом шаге получится отрицательный остаток, что означает нуль в целой части мантиссы частного и продолжение алгоритма деления по рассмотренным выше правилам для чисел, представленных в форме с фиксированной запятой. Таким образом, частное всегда получается в прямом коде, а действия над мантиссами осуществляются на ДСОК или ДСДК.

Так как при операции деления порядки чисел вычитаются (изображения складываются), возможно переполиение разрядной сетки в сумматоре порядков. При переполнении в сторону отрицательных величин порядка мантисса результата превращается в машинный нуль, а порядку присваивается наибольшее отрицательное значение. Если делитель равен нулю, то также вырабатываются сигналы $\varphi = 1$ и останов машины. Эти частные случаи предусмотрены при реализации алгоритмов в ЕС ЭВМ, где используется алгоритм деления без восстановления остатков. При этом мантисса делимого имеет длину в два раза больше, чем делитель, что иллюстрируется примером 6.4.

Пример 6.4. Разделить число $A = 0{,}10001111 \cdot 2^3$ на $B = 0{,}1111 \cdot 2^2$

Р е ш е н и е. Рассматривается случай, когда $|m_A| < |m_B|$. Прежде всего записываются машинные изображения мантиссы делимого $[m_A]_{\text{д}}^{\text{м}} = 00{,}10001111$; мантиссы делителя $[m_B]_{\text{д}}^{\text{м}} = 00{,}1111$ и $[\overline{m}_B]_{\text{д}}^{\text{м}} = 11{,}0001$.

Все действия выполняются на сумматоре дополнительного кода в последовательности, указанной в таблице 6.4

Таблица 6.4

Сумматор ←	Регистр C ←	Примечание
00,10001111	0000	$И.П.\ СМ := [m_A]^м_д$; $РгС := 0$; $РгВ := [m_B]^м_д$;
01,00011110	000–	сдвиг сумматоров и регистров
+ 11,0001		$[СМ] := [СМ] + [\overline{m}]^м_д$;
00,00101110	0001	$c_1 = 1$;
00,01011100	001	сдвиг сумматоров и регистров
+ 11,0001		$[СМ] := [СМ] + [\overline{m}_B]^м_д$;
11,01101100	0010	$c_2 = 0$;
10,11011000	010–	сдвиг сумматоров и регистров
+ 00,1111		$[СМ] := [СМ] + [m_B]^м_д$;
11,11001000	0100	$c_3 = 0$;
11,10010000	100	сдвиг сумматоров и регистров
+ 00,1111		$[СМ] := [СМ] + [m_B]^м_д$.
00,10000000	1001	$c_4 = 1$;
		Конец

Одновременно вычисляется порядок частного следующим образом: $p_C = p_A - p_B = 0{,}011 - 0{,}010 = 0{,}001$ и определяется знак частного $Sg_C = 0 \oplus 0 = 0$.

Ответ: $C = 0{,}1001 \cdot 2^1$

6.4. Ускорение операции деления

Идея метода ускорения операции деления заключается в том, что в случае образования достаточно малого или достаточно большого по абсолютной величине остатка очередные цифры частного будут группой одинаковых цифр — либо нулей, либо единиц, поэтому продолжение процесса деления обычным способом излишне, так как эту группу цифр можно записать в частное сразу.

Рассмотрим примеры деления десятичных и двоичных чисел:

1)
$$\begin{array}{r|l} 2281825 & 2275 \\ \hline 2275 & 1003 \\ \hline \underbrace{0006825}_{k} & k-1 \end{array}$$

$$\begin{array}{r} 6825 \\ \hline 0000 \end{array}$$

2)
$$\begin{array}{r|l} 1010001 & 1001 \\ \hline 1001 & 1001 \\ \hline \underbrace{000}1001 & k-1 \end{array}$$

$k-$

$$\begin{array}{r} 1001 \\ \hline 0000 \end{array}$$

Здесь на первом шаге получился малый по величине положительный остаток, который содержит нули в трех старших разрядах. В частном после первой цифры записаны нули в два следующих разряда. Можно утверждать, что при малом положительном остатке в частное можно сразу записать $k-1$ нулей в соответствующие разряды, остаток сдвинуть на k разрядов влево, вычесть из него делитель и далее продолжить операцию деления.

Если в результате вычитания на каком-то шаге получается большой положительный остаток, в котором k старших разрядов содержат единицы (для двоичных чисел), то по меньшей мере в $(k-1)$-е разряды частного можно записать единицу:

$$
\begin{array}{l}
1000000000\ldots \\
\underline{-\ 10001} \\
011110 \\
\underbrace{}_{k} \\
\underline{-\ 10001} \\
011010 \\
\underline{-\ 10001} \\
010010 \\
\underline{-\ 10001} \\
000001\ldots
\end{array}
\qquad
\begin{array}{|l}
10001 \\
\hline
11110
\end{array}
$$

Таким образом, в данном случае процесс деления должен осуществляться комбинированным способом, с анализом последовательности нулей (если остаток положительный) или последовательности единиц (если остаток отрицательный) в старших разрядах остатка. При обнаружении такой последовательности длиной k сразу можно записать в $(k-1)$-е разряды частного соответственно нули или единицы. После этого провести сдвиг на $(k-1)$ разряд и продолжить операцию деления.

В других случаях операция выполняется по обычному алгоритму.

Рассмотренный метод ускорения операции деления может дать эффект только тогда, когда встречаются последовательности нулей или единиц в остатках.

Для ускорения операции деления можно воспользоваться также методом одновременного определения двух цифр частного, который является развитием изложенной выше идеи. Этот метод реализован в некоторых ЭВМ третьего поколения.

Пусть A — делимое; B — делитель; A_{i-1} — остаток на $(i-1)$-м шаге деления.

Для остатка справедливо $0 \geq |A_{i-1}| < B$.

Если теперь остаток сдвинуть на два разряда, то $2^2 A_{k-1}$. Тогда следующую пару цифр частного можно найти из условий, представленных в таблице 6.5.

Таблица 6.5

Условие для анализа	Пара цифр частного	A_i
$2^2 A_{i-1} < B$	00	$2^2 A_{i-1}$
$B \le 2^2 A_{i-1} < 2B$	01	$2^2 A_{i-1} - B$
$2B \le 2^2 A_{i-1} < 3B$	10	$2^2 A_{i-1} - 2B$
$3B \le 2^2 A_{i-1}$	11	$2^2 A_{i-1} - 3B$

Изложенный алгоритм деления может быть упрощен, если каждую пару цифр частного определять, исходя из анализа нескольких старших разрядов делителя B и сдвинутого остатка $2^2 A_{i-1}$. При этом если старшие разряды остатка $2^2 A_{i-1}$ совпадают со старшими разрядами групп, то предполагается, что $2^2 A_{i-1}$ принадлежит к области с наибольшим значением пары цифр частного и исходя из этого находится остаток A_i. Если A_i — отрицательный, то получена величина остатка, уменьшенная на B:

$$-B \le A_i' = (A_i - B) < 0. \qquad (6.5)$$

Тогда коррекция остатка осуществляется следующим образом: если раньше проверялось условие $jB \le 2^2 A_{i-1} \le (j+1)B$, где $j = 0, 1, 2, 3$, то теперь проверяется эквивалентное ему условие

$$(j-4)B \le 2^2 (A_{i-1} - B) < (j-3)B. \qquad (6.6)$$

Можно сформулировать правила, представленные в таблице 6.6.

Таблица 6.6

$A_{i-1} \ge 0$		$A_{i-1} = A_{i-1} - B < 0$		Пара цифр частного	
Условие для анализа	A_i	Условие для анализа	A_i	$A_{i-1}' \ge 0$	$A_{i-1}' < 0$
$2^2 A_{i-1} < B$	$2^2 A_{i-1}$	$2^2 A_{i-1} < -3B$	$2^2 A_{i-1} + 4B$	00	—
$B \le 2^2 A_{i-1} < 2B$	$2^2 A_{i-1} - B$	$-3A \le 2^2 A_{i+1} < -2B$	$2^2 A_{i-1} + 3B$	01	00
$2B \le 2^2 A_{i-1} < 3B$	$2^2 A_{i-1} - 2B$	$-2A \le 2^2 A_{i-1} < -B$	$2^2 A_{i-1} + 2B$	10	01
$3B \le 2^2 A_{i-1}$	$2^2 A_{i-1} - 3B$	$-A \le 2^2 A_{i-1} < 0$	$2^2 A_{i-1} + B$	11	10

По этим правилам строятся указанные алгоритмы выполнения операции деления.

6.5. Параллельные методы деления с использованием итеративных структур

Как правило, при выполнении операции деления тратится один тактовый цикл для получения одной двоичной цифры частного. Параллельные делители практически невозможно реализовать, используя классический прием определения цифры частного: вычитанием делителя из текущего остатка делимого. Вместе с тем наличие быстродействующих параллельных умножителей (см. гл. 5) позволяет создавать параллельные алгоритмы деления с использованием этих умножителей. Можно реализовать алгоритм деления, используя обратную величину делителя и замену операции деления операцией умножения. При этом обратная величина делителя для всего диапазона значений хранится в ПЗУ. Емкость ПЗУ зависит от разрядности машинного слова (см. табл. 6.7)*.

Таблица 6.7

Разрядность слова, бит	Емкость ПЗУ, Кбит
8	2
12	50
14	225
16	1040
24	384000

Рассмотрим возможность реализации операции деления с использованием итеративных структур на примере устройства, предложенного в работе М. Канна, В. К. Галахера «Итеративное устройство для ускоренного двоичного деления». Здесь приняты следующие обозначения:

делимое — $A = A_0 \cdot A_1 A_2 \dots A_N$;

делитель — $D = D_0 \cdot D_1 D_2 \dots D_N$;

остаток — $R = R_0 \cdot R_1 R_2 \dots R_N$;

частное — $Q = Q_0 \cdot Q_1 Q_2 \dots Q_N$.

Предполагается, что все операнды положительные, нормализованные дроби.

В итерационной матрице деления (ИМД) используется алгоритм деления без восстановления остатка, описанный ранее, но с двумя существенными изменениями. Основная идея метода — уменьшение времени волны поразрядных переносов, пропорционального числу разрядов N, в каждом ряду матрицы.

* *Мейзер С., Элтман Б.* «Быстродействующее устройство деления для специализированных ИС различного назначения». М.: Электроника, № 14, 1989. С. 29–35.

Первое изменение заключается в том, что промежуточный остаток R не формируется в каждом ряду матрицы, а представляется в виде двух двоичных векторов S и C, которые, будучи сложенными, дают настоящее значение промежуточного остатка в данном ряду матрицы. Второе изменение состоит в применении дополнительной схемы, для того чтобы по виду векторов S и C определять, пронзойдет ли перенос в знаковый разряд промежуточного остатка. Эта схема облегчает определение знака промежуточного остатка, очередного разряда частного и сигнала управления (прибавлять или отнимать делитель) для следующего ряда матрицы.

Два вектора S и C получаются сложением на сумматоре с запоминанием переносов векторов S и C предыдущего ряда матрицы и соответствующего вида делителя (в зависимости от того, прибавляется он илн отнимается).

Вычитание делителя проводится в дополнительном коде, как и в случае базовой матрицы для деления.

На таком сумматоре задержка сигнала определяется только одной логической ячейкой и не зависит от длины операндов. Быстродействие схемы определения переноса в знаковый разряд, выборка делителя илн его дополнения, задержка на одну ячейку при суммировании — основные факторы, определяющие время, необходимое для формирования одного разряда частного.

Определение переноса в знаковый разряд промежуточного остатка проводится по формуле

$$GL = G_1 + P_1G_2 + P_1P_2G_2 + \ldots + P_1P_2 \ldots P_{N-2}G_{N-1},$$

где $G_j = S_jC_j$ — порождающая функция; $P_j = S_j + C_j$ — накапливающая функция.

Для случая N разрядов имеем

$$CL = S_1C_1 + (S_1 + C_1)S_2C_2 + (S_1 + C_1)(S_2 + C_2) \times$$
$$\times S_1C_3 + \ldots + (S_1 + C_1)\ldots(S_{N-2} + C_{N-2})S_{N-1}C_{N-1}.$$

Для описания алгоритма деления ИМД можно использовать обозначения, приведенные в гл. 5. Векторы S и C — это векторы поразрядной суммы и переноса соответственно, получающиеся на сумматоре типа 3 в 2 (полный суммагор).

Алгоритм работы ИМД выглядит следующим образом:

Цикл для i — от 0 до N.

Шаг 1. Формирование двух новых векторов промежуточного остатка:

если $Q_{i-1} = 1$, то $S, C := S + C - D$;

если $Q_{i-1} = 0$, то $S, C := S + C + D$.

Шаг 2. Определение переноса в знаковый разряд промежуточного остатка:

$$GL = G_1 + P_1G_2 + P_1P_2G_2 + \ldots + P_1P_2 \ldots P_{N-2}G_{N-1}.$$

Шаг 3. Определение знака промежуточного остатка:

$$R_0 + S_0 \oplus C_0 \oplus CL.$$

Определение очередного разряда частного: $Q_i = \overline{R}_0$.

Шаг 4. Сдвиг векторов S и C влево: $S := 2(S)$, $C := 2(C)$. Для $i = 0$ *формально принимается* $S = A =$ *деление*, $Q_{-1} = 1$ *и* $C = 0$. *Основным является* шаг 1, *сформированный выше, он может быть проиллюстрирован следующим образом:*

$$\begin{array}{l} S_0 \cdot S_1S_2 \ldots S_{N-1}S_N \\ \pm\ C_0 \cdot C_1C_2 \ldots C_{N-1}C_N \\ \underline{D_0 \cdot D_1D_2 \ldots D_{N-1}D_N} \\ S_0 \cdot S_1S_2 \ldots S_{N-1}S_N \\ C_0 \quad \underbrace{C_1C_2 \ldots C_{N-1}}_{\text{Биты, определяющие } CL}0 \end{array}$$

Комбинация суммирования с запоминанием переносов и ускоренного определения знака промежуточного остатка обеспечивает высокое быстродействие алгоритма ИМД при не очень больших затратах на дополнительное оборудование.

Для ИМД, описанной выше, время деления возрастает линейно с увеличением длины слова. Фактически зависимость выражается в виде $N \cdot \log N$, где составляющая $\log N$ вносится схемой определения переноса в знаковый разряд (основание логарифма определяется максимальным коэффициентом объединения по входу логических схем). Время деления в базовой матрице деления, описанной выше, растет пропорционально N^2 из-за распространения переноса вдоль каждого ее ряда.

Для $N = 16$ ИМД работает в 2,6 раза быстрее базовой матрицы деления, но дороже ее на 22 %. Для $N = 32$ скорость ИМД в пять раз превышает скорость базовой матрицы деления и дороже ее на 25 %.

Пример 6.5. Разделить с помощью ИМД $A = 0{,}10011$ на $B = 0{,}11001$ на сумматоре дополнительного кода.

Решение: $[A]_д = 0{,}10011, [B]_д = 0{,}11001, [\overline{B}]_д = 1{,}00111$;

$CL = S_1C_1 + (S_1 + C_1)S_2C_2 + (S_1 + C_1)(S_2 + C_2)S_3C_3 + (S_1 + C_1)(S_2 + C_2)(S_3 + C_3)S_4C_4$

Ответ: $C_д = 0{,}11000$.

Таблица 6.8

Разряды							Примечание
	0	1	2	3	4	5	
S	0	1	0	0	1	1	$S_0 = [A]_д$,
C	0	0	0	0	0	0	$C_0 = 0$, $Q_{-1} = 1$,
$-B$	1	0	0	1	1	1	если $Q_{-1} = 1$, то S, $C := S + C - B$;
S	1	1	0	1	0	0	$C_1 = 1 \cdot 0 + (1+0) \cdot 0 \cdot 0 + (1+0)(0+0)1 \cdot 1 + (1+$
C	0	0	0	1	1	0	$+(1+0)(0+0)(1+1) \cdot 0 \cdot 1 = 0$,
							$R_1 = 1 \oplus 0 \oplus 0 = 1$; $Q_0 = \overline{R}_0 = 0$,
S	1	0	1	0	0	0	$S_1 = \tilde{S}$,
C	0	0	1	1	0	0	$C_1 = \tilde{C}$,
$+B$	0	1	1	0	0	1	если $Q_0 = 0$, то S, $C := S + C + B$;
S	1	1	1	1	0	1	$C_2 = 1 \cdot 1 + (1+1)1 \cdot 0 + (1+1)(1+0)1 \cdot 0 + (1+$
C	0	1	0	0	0	0	$+(1+0)(1+0)(1+1) \cdot 0 \cdot 0 = 1$,
							$R_2 = 1 \oplus 0 \oplus 1 = 0$; $Q_1 = \overline{R}_0 = 1$,
S	1	1	1	0	1	0	$S_2 = \tilde{S}$,
C	1	0	0	0	0	0	$C_2 = \tilde{C}$,
$-B$	1	0	0	1	1	1	если $Q_1 = 1$, то S, $C := S + C - B$;
S	1	1	1	1	0	1	$C_3 = 1 \cdot 0 + (1+0)1 \cdot 0 + (1+0)(1+0)1 \cdot 1 + (1+$
C	0	0	0	1	0	0	$+(1+0)(1+0)(1+1) \cdot 0 \cdot 0 = 1$,
							$R_3 = 1 \oplus 0 \oplus 1 = 0$; $Q_2 = \overline{R}_0 = 1$,
S	1	1	1	0	1	0	$S_3 = \tilde{S}$,
C	0	0	1	0	0	0	$C_3 = \tilde{C}$,
$-B$	1	0	0	1	1	1	если $Q_2 = 1$, то S, $C := S + C - B$;
S	0	1	0	1	0	1	$C_4 = 1 \cdot 1 + (1+1) \cdot 0 \cdot 0 + (1+1)(0+0)1 \cdot 1 + (1+$
C	0	1	0	1	0	0	$+(0+0)(1+1) \cdot 0 \cdot 0 = 1$,
							$R_4 = 0 \oplus 0 \oplus 1 = 1$; $Q_3 = 0$,
S	1	0	1	0	1	0	$S_4 = \tilde{S}$,
C	1	0	1	0	0	0	$C_4 = \tilde{C}$,
$+B$	0	1	1	0	0	1	если $Q_3 = 0$, то S, $C := S + C + B$;
S	0	1	1	0	1	1	$C_5 = 1 \cdot 1 + (1+1) \cdot 1 \cdot 0 + (1+1)(1+0) \cdot 0 \cdot 0 + (1+$
C	0	1	0	0	0	0	$+(1+0)(0+0)1 \cdot 0 = 1$,
							$R_5 = 0 \oplus 0 \oplus 1 = 1$; $Q_4 = 0$,
S	1	1	0	1	1	0	$S_5 = \tilde{S}$,
C	1	0	0	0	0	0	$C_5 = \tilde{C}$,
$+B$	0	1	1	0	0	1	если $Q_4 = 0$, то S, $C := S + C + B$;
S	0	0	1	1	1	1	$C_6 = 0 \cdot 0 + (0+0)1 \cdot 0 + (0+0)(1+0)1 \cdot 0 + (0+$
C	1	0	0	0	0	0	$+(1+0)(1+0)1 \cdot 0 = 0$,
							$R_6 = 0 \oplus 1 \oplus 0 = 1$; $Q_5 = 0$.

На рис. 6.2 представлена типовая процессорная ячейка, разработанная Агравалем для итеративных структур, используемых для устройств деления. В типовой ячейке сигнал b_i прибавляется к результату суммирования

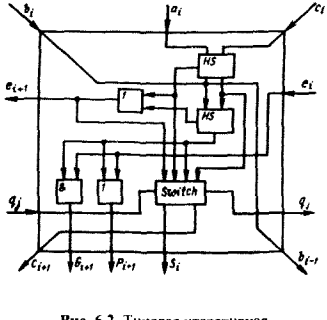


Рис. 6.2. Типовая итеративная структура Атраваля

a_i и c_i только в том случае, когда $q_j = 1$. Кроме того, c_i всегда включается в процесс вычислений ожидаемого переноса e_{i+1} и двух сквозных переносов G_{i+1} и P_{i+1}. Все процессы, происходящие в типовом процессоре, описываются уравнениями:

$$S_i = (a_i \oplus c_i) \oplus (q_j \cdot b_i),$$

$$c_{i+1} = a_i c_i + (a_i + c_i)(q_j \cdot b_i),$$

$$e_{i+1} = a_i c_i + (a_i + c_i) \cdot b_i,$$

$$G_{i+1} = (a_i \oplus b_i \oplus c_i) \cdot e_i,$$

$$P_{i+1} = (a_i \oplus b_i \oplus c_i) + e_i.$$

Общая схема делительного устройства представлена на рис. 6.3. Основными элементами этого устройства являются одноразрядные сумматоры с ускоренным переносом (ОСУП). На рис. 6.3 величина a (делимое, имеет семь разрядов) делится на величину b (делитель, имеет четыре разряда), разряды частного q получаются на соответствующих выходах. При этом разряд частного $q_j = 1$ только тогда, когда прибавление делителя b в дополнительном коде приводит к положительному остатку, что указывается единицей переполнения из старшего значащего разряда. С другой стороны, если $q_j = 0$, то первоначальное значение остатка на этом шаге восстанавливается. В устройстве деления в каждом ряду матричной структуры образуются две частичные суммы. Поэтому восстановление происходит несколько своеобразным способом. Если возникает необходимость восстановления (или любого неарифметического действия), то результат суммирования двух частичных остатков от предыдущего шага передается в следующий ряд как входная величина. Дополнительный код делителя образуется как обратная величина в каждом разряде делителя, что вызывает перенос $e = 1$ и добавление единицы в правый разряд делителя.

Очень важна процедура определения знака остатков. Примем во внимание следующее:

1) если коррекция не требуется, то среди переменных S, C_1, C_2 и $C_{\text{СЛА}}$ только одна величина может быть равна единице (если единице равна не одна величина, то это указывает на положительный знак);

Рис. 6.3. Структурная схема устройства деления на итеративных структурах

2) если коррекция переполнения необходима, то по крайней мере две или три величины из S, C_1, C_2, $C_{\text{СЛА}}$ равны единице (если три величины равны единице, то знак остатков положительный);

3) если $C_{\text{СЛА}} = 1$, то требуется коррекция на следующем шаге.

Сигналы S, C_1, C_2 — это выходы левых ячеек каждого ряда, а сигнал $C_{\text{СЛА}}$ — ожидаемый перенос в данном ряду.

Таким образом, положительный знак получается при условии

$$q_+ = S \oplus C_1 \oplus C_2 \oplus C_{\text{СЛА}}.$$

6.6. Операция извлечения квадратного корня

Операция извлечения квадратного корня как самостоятельная операция в систему команд ЭВМ включается в случае, когда приходится относительно часто прибегать к ее выполнению (не менее 2 % от общего числа операций). Такие ситуации могут встречаться при создании специализированных ЭВМ.

Основным приемом для приближенного вычисления квадратного корня в универсальных ЭВМ является метод итераций, основанный на использовании формулы Ньютона.

Пусть $y = \sqrt{x}$. Тогда $F(x, y) = y^2 - x = 0$, $F'(x, y) = 2y$. Считая, что $y_n \approx y$ есть приближенное значение искомой функции, по теореме Лагранжа можно определить $F(x, y_n) = F(x, y_n) - F(x, y) = (y_n - y)F'_y(x, \bar{y}_n)$, где $\bar{y}_n$ — некоторое промежуточное значение между y_n и y.

Тогда $y_{n+1} = y_n - (y^2_n - x)/2y_n$, или

$$y_{n+1} = 0{,}5(y_n + x/y_n), \qquad (6.7)$$

где $n = 0, 1, 2, \ldots$

Формула (6.7) может быть положена в основу алгоритма вычисления квадратного корня.

На практике используют и несколько иной метод. Пусть задано подкоренное число $a = 0, a_1 a_2 \ldots a_n$. Предположим, что удалось найти $(k-1)$-ю цифру значения корня, равную $0, b_1 b_2 \ldots b_{k-1}$. По условию, очередной остаток $A_{k-1} = A - 0, b_1 b_2 \ldots b^2_{k-1} > 0$. Очередная цифра b_k может быть нуль или единица. Очевидно, что если $A_k > 0$, то $b_k = 1$. При этом

$$A_k = A - (0, b_1 b_2 \ldots b_{k-1}1)^2 = A - (0, b_1 b_2 \ldots b_{k-1})^2 - 2 \cdot 2^{-k} \cdot 0, b_1 b_2 \ldots b_{k-1} - 2^{-2k},$$

или

$$A_k = A_{k-1} - 2^{-(k-1)} 0, b_1 b_2 \ldots b_{k-1} 01. \qquad (6.8)$$

Следовательно, для получения очередного остатка надо к уже найденному числу $0, b_1 b_2 \ldots b_{k-1}$ приписать пару цифр 01 и вычесть полученное число, предварительно сдвинутое на $(k-1)$ разряд, из предыдущего остатка. При этом, если $A_k \geq 0$, то $b_k = 1$; если $A_k < 0$, то $b_k = 0$.

Таким образом, (6.8) показывает, что операция извлечения квадратного корня напоминает операцию деления на переменный делитель, равный $0, b_1 b_2 \ldots b_{k-1} 01$. Первое значение переменного делителя равно 0,01.

Пример 6.6. Извлечь квадратный корень из числа $A = 0{,}100101$ на сумматоре дополнительного кода.

Р е ш е н и е. См. табл. 6.9.

Ответ: $B = A = 0{,}110000$.

При обращении с отрицательными числами следует вырабатывать сигнал переполнения, который покажет нарушение правильного прохождения алгоритма.

Задание для самоконтроля

1. Разделить заданные в прямом коде числа: а) $[A]_{пр} = 1{,}100011$ и $[B]_{пр} = 1{,}110011$, б) $[A]_{пр} = 0{,}100111$ и $[B]_{пр} = 1{,}100011$.

2. Разделить заданные в обратном коде числа $[A]_{об} = 1{,}011001$ и $[B]_{об} = 0{,}11001$ методом с восстановлением остатков.

Таблица 6.9

Сумматор	Регистр B	Примечание
00,100101	000000	И. П. СМ := $[A]^{м}_{п}$; PrB = 0;
+ 11,110000		СМ := [СМ] + $[-0{,}01]^{м}_{п}$;
00,010101	000001	b_6 = 1;
00,101010	00001–	сдвиг
+ 11,011000		СМ := [СМ] + $[-0{,}101]^{м}_{п}$;
00,000010	000011	b_5 = 1;
00,000100	00011	сдвиг
+ 11,001100		СМ := [СМ] + $[-0{,}1101]^{м}_{п}$;
11,010000	000110	b_4 = 0;
+ 00,110100		восстановление остатка
00,000100		
00,001000	00110–	сдвиг
+ 11,001110		СМ := [СМ] + $[-0{,}11001]^{м}_{п}$;
11,010110	001100	b_3 = 0;
+ 00,110010		восстановление остатка
00,010000	01100–	сдвиг
+ 11,001111		СМ := [СМ] + $[-0{,}110001]^{м}_{п}$;
11,011111	011000	b_2 = 0;
+ 00,110001		восстановление остатка
00,010000		
00,100000	11000–	сдвиг
+ 11,001111		СМ := [СМ] + $[-0{,}1100001]^{м}_{п}$;
11,101111	110000	b_1 = 0;
		Конец

3. Разделить заданные в дополнительном коде числа $[A]_{доп} = 0{,}110000$ и $[B]_{доп} = 1{,}000111$ методом без восстановления остатков.

4. Найти частное от деления чисел $[A]_{об} = 0{,}1010001111$ и $[B]_{об} = 1{,}1010110010$ с использованием ускоренного метода.

5. Можно ли применить правила двоичного деления чисел, представленных в минус-двоичной системе?

6. Провести на сумматоре обратного кода деление чисел $A = -0{,}10011 \cdot 2^{-6}$ и $B = 0{,}100101 \cdot 2^{-4}$, представленных в форме с плавающей запятой.

7. Провести деление на сумматоре дополнительного кода чисел $A = 0{,}101101 \cdot 2^{5}$ и $B = -0{,}111001 \cdot 2^{1}$, представленных в форме с плавающей запятой.

8. Возможно ли переполнение разрядной сетки при делении чисел, представленных в форме с плавающей запятой?

9. Извлечь квадратный корень из числа $A = 0{,}111111$ на сумматоре обратного кода.

7. ВЫПОЛНЕНИЕ ОПЕРАЦИЙ НАД ДЕСЯТИЧНЫМИ ЧИСЛАМИ В ЦИФРОВЫХ АВТОМАТАХ

7.1. Представление десятичных чисел в Д-кодах

Операции над десятичными числами (десятичная арифметика) часто включаются в состав основных команд универсальных ЭВМ. Кроме того, десятичная арифметика реализуется широко в электронных калькуляторах и персональных микроЭВМ. Поэтому кроме общей информации о возможности представления десятичных чисел разработчику необходимо знать и алгоритм выполнения арифметических операций.

Д-код[*] *(двоично-кодированное представление) десятичного числа — такое его представление, в котором каждая десятичная цифра изображается тетрадой из двоичных символов:*

$$A_д = \{\alpha_4^1\alpha_3^1\alpha_2^1\alpha_1^1\}_1\{\alpha_4^2\alpha_3^2\alpha_2^2\alpha_1^2\}_2 \ldots \{\alpha_4^n\alpha_3^n\alpha_2^n\alpha_1^n\}_n, \qquad (7.1)$$

где α_i^j *— двоичные разряды тетрады j; n — количество десятичных разрядов.*

Количество различных Д-кодов определяется количеством возможных сочетаний по 10 из 16 комбинаций, которые допускает тетрада.

При образовании Д-кода следует исходить из общих требований, предъявляемых к системам счисления:

различным десятичным цифрам должны соответствовать различные тетрады;

бóльшая десятичная цифра должна изображаться бóльшей тетрадой (если разряды тетрады имеют вес по двоичной системе счисления);

для десятичных цифр $a = \{\alpha_4\alpha_3\alpha_2\alpha_1\}$ и $b = \{\beta_4\beta_3\beta_2\beta_1\}$, связанных со отношением $a + b = 9$, должно удовлетворяться условие

$$\beta_i = \begin{cases} 0, \text{ если } \alpha_i = 1 \\ 1, \text{ если } \alpha_i = 0 \end{cases} (i = 1, 2, 3, 4), \qquad (7.2)$$

[*] Название «Д-код» образовано от полного названия «десятичный код».

Для однозначности перевода чисел в Д-код и обратно желательно, чтобы разряды тетрад имели определенный вес. Тогда значение десятичной цифры α_i соответствует выражению

$$a_i = \alpha_4\sigma_4 + \alpha_3\sigma_3 + \alpha_2\sigma_2 + \alpha_1\sigma_1, \qquad (7.3)$$

где σ_i — вес разряда тетрады.

В таблице 7.1 представлено кодирование десятичных цифр в различных Д-кодах.

Таблица 7.1

Десятичные цифры	Эквиваленты в кодах						
	$Д_1$ (система 8421)	$Д_2$ (система 2421)	$Д_3$ (система 5121)	$Д_4$ (система 8421+3)	$Д_5$ (система 53–21)	$Д_6$ (система 75–31)	$Д_7$ (система 5421)
0	0000	0000	0000	0011	0000	0000	0000
1	0001	0001	0001	0100	0001	0001	0001
2	0010	0010	0010	0101	0111	0110	0010
3	0011	0011	0011	0110	1010	0111	0011
4	0100	0100	0111	0111	0101	1010	0100
5	0101	1011	1000	1000	1000	0100	1000
6	0110	1100	1001	1001	1001	0101	1001
7	0111	1101	1010	1010	1111	1000	1010
8	1000	1110	1011	1011	1100	1001	1011
9	1001	1111	1111	1100	1101	1110	1100

В таблице для каждого Д-кода указаны разрешенные комбинации. Все другие комбинации — запрещенные.

Наличие разрешенных и запрещенных комбинаций — очень важное свойство Д-кодов. Оно отличает их от обычных позиционных систем счисления, в которых все комбинации — разрешенные.

Рассмотрим наиболее распространенные Д-коды.

Код $Д_1$ прямого замещения (система 8421). В коде $Д_1$ разрешенные комбинации соответствуют двоичным эквивалентам десятичных цифр с весами разрядов, равных степеням основания 2. Для этого кода не выполняется условие (7.2), так как цифры, являющиеся дополнением до 9, не получаются простым инвертированием наборов тетрад.

Код $Д_2$ (система 2421). Для кода $Д_2$ веса разрядов тетрады соответственно равны 2, 4, 2, 1; таблица кодирования делится на две части: от 0 до 4 — тетрады повторяют двоичные эквиваленты; от 5 до 9 — по сравне-

нию с двоичной системой каждая тетрада содержит избыток +0110. Это дает возможность любую цифру одной части таблицы превратить в ее дополнение до 9 простым инвертированием.

Код $Д_4$ (система 8421+3). Для этого кода все тетрады имеют значения на три единицы больше, чем тетрады кода $Д_1$ (отсюда название кода), и для него не существует целочисленных значений веса, которые удовлетворяли бы (7.3).

Коды $Д_5$ (система 53–21) и $Д_6$ (система 75–31). Эти коды отличаются от вышеназванных кодов тем, что для них некоторые веса имеют отрицательное значение.

В вычислительных машинах разного назначения чаще всего используются коды $Д_1$ и $Д_4$.

7.2. Формальные правила поразрядного сложения в Д-кодах

Для определения формальных правил поразрядного сложения чисел, представленных в Д-коде, рассмотрим те особенности, которые присущи этим кодам.

1. Наличие разрешенных и запрещенных комбинаций.

Появление запрещенной комбинации при выполнении каких-то действий над числами свидетельствует о возникновении ошибки или же о необходимости ввести корректировку результата.

2. При сложении тетрад возникает потетрадный перенос $п'_i = 16$ вместо поразрядного переноса $п_i = 10$. Это приводит к необходимости коррекции результата.

В самом деле, если складываются числа $A_д = a_n a_{n-1} \ldots a_1 a_0$ и $B_д = b_n b_{n-1} \ldots b_1 b_0$, то сумма $C_д = A_д + B_д$ и

$$C_i = a_i + b_i + п_{i-1} - п_i q, \qquad (7.4)$$

где C_i — i-й разряд суммы; $п_{i-1}$ — перенос из младшей тетрады; $п_i$ — перенос в старшую тетраду ($п_{i-1} = \{0, 1\}$, $п_i = [0, 1]$, $q = 10$).

Далее выведем правила сложения применительно к Д-кодам.

При сложении чисел в коде $Д_1$ могут возникнуть следующие случаи.

1. Пусть $a_i + b_i + п_{i-1} < 10$, где a_i, b_i — тетрады кода $Д_1$. При сложении в данном разряде числа образуется сумма меньше 10. Если действия над

разрядами тетрады проводят по правилам двоичной арифметики, то правильный результат получают без коррекции.

Пример 7.1. Сложить тетрады $a_t = 0100$ и $b_t = 0101$ при значении $п_{t-1} = 0$.

Решение $c_t = a_t + b_t + п_{t-1} = 1001$.

Ответ $c_t = 1001$.

2. Пусть $a_t + b_t + п_{t-1} \geq 10$, т. е. возникает десятичный перенос и сумма должна быть равна $a_t + b_t + п_{t-1} - п_t 10$, где $п_t = 1$.

Свидетельством того, что результат неправильный, является либо появление запрещенной комбинации, если $15 \geq a_t + b_t + п_{t-1} \geq 10$, либо появление потетрадного переноса $п'_t = 16$, что превышает значение десятичного переноса на 6. Следовательно, требуется коррекция результата в данной тетраде введением поправки, равной $+0110$.

Пример 7.2. Сложить тетрады $a_t = 0101$ и $b_t = 1001$ при значении $п_{t-1} = 1$.

Решение $c'_t = a_t + b_t + п_{t-1} = 1111$.

Величина $c_t = 1111$ – запрещенная комбинация. Следовательно, надо ввести поправку:

$$\begin{array}{r} 1111 \\ +\ 0110 \\ \hline (1) = 0101, \\ п_t \end{array}$$

т. е. результат равен 0101 в данной тетраде и образован перенос в старшую тетраду.

Ответ $c_t = 0101, п_t = 1$.

Пример 7.3. Сложить тетрады $a_t = 0111, b_t = 1001$ при значении $п_{t-1} = 1$.

Решение. $c'_t = a_t + b_t + п_{t-1} = (1)0001.$

$п'_t$

Появление потетрадного переноса требует коррекции результата: $c_t = 0001 + 0110 = 0111$.

Ответ $c_t = 0111, p_t = 1$.

Примеры, рассмотренные выше, дают возможность сформулировать следующие правила потетрадного сложения чисел в $Д_1$-коде:

если при потетрадном сложении перенос в соседнюю старшую тетраду не возникает $(п_t = 0)$, *то результат суммирования не требует коррекции;*

коррекция результата потетрадного сложения путем добавления поправки $+0110$ *требуется в случае, если:*

а) *возникает потетрадный перенос в старшую тетраду* $(п_t = 1)$;

б) *возникает запрещенная комбинация.*

Устройство, которое работает по сформулированным выше правилам, называется *однораздрядным десятичным сумматором* для Д-кода (рис. 7.1).

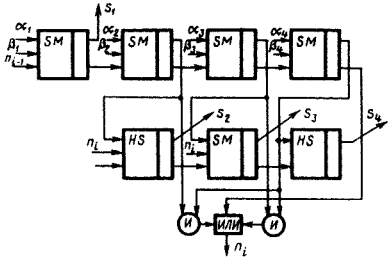


Рис. 7.1. Структурная схема однораздрядного десятичного сумматора для $Д_1$ - кода

Рассмотрим пример суммирования целых чисел.

Пример 7.4. Сложить числа $A = 279 = 0010\,0111\,1001$ и $B = 581 = 0101\,1000\,0001$, записанные в коде $Д_1$.

Решение. Прежде всего проводится потетрадное суммирование, а затем коррекция там, где это необходимо:

$$\begin{array}{rlcccl} A = & & 0010 & 0111 & 1001 & \\ B = & + & 0101 & 1000 & 0001 & \\ \hline & & 0111 & 1111 & 1010 & \\ & + & & 0110 & 0110 & \text{поправки} \\ \hline C = & & 1000 & \leftarrow 0110 & \leftarrow 0000 & \end{array}$$

Здесь стрелка указывает передачу единицы десятичного переноса.

Ответ: $C = 860 = 1000\,0110\,0000_{Д_1}$.

При сложении чисел в коде $Д_2$ могут возникнуть следующие случаи.

1. Пусть $a'_i < 5$ и $b'_i < 5$, где a'_i, b'_i — тетрады для кода $Д_2$. Тогда:

если $a'_i + b'_i + п_{i-1} < 5$, то результат сложения не требует коррекции;

если $a'_i + b'_i + п_{i-1} \geq 5$, то результат попадает во вторую часть таблицы кодирования, где $c'_i = c_i + 6$.

Здесь необходима коррекция результата введения поправки 0110. Признак этого — появление запрещенных комбинаций.

2. Пусть $a_i' \geq 5$ и $b_i' < 5$; $15 > a_i' + b_i' + \pi_{i-1} \geq 5$.

Так как $a_i' = a_i + 6$, при суммировании поправка не требуется.

3. Пусть $a_i' \geq 5$ и $b_i' \geq 5$; $\pi_i = 1$. Тогда:

если $10 \leq a_i' + b_i' + \pi_{i-1} \leq 15$, то результат требует коррекции путем введения поправки -0110, так как появляется запрещенная комбинация*;

если же $a_i' + b_i' + \pi_{i-1} < 15$, то результат не требует коррекции.

При потетрадном суммировании в $Д_2$-коде результат суммирования без коррекции получается во всех случаях, кроме следующих:

если при отсутствии переноса в старшую тетраду ($\pi_i = 1$) возникает запрещенная комбинация, то требуется ввести поправку $+0110$;

если при наличии переноса в старшую тетраду ($\pi_i = 1$) возникает запрещенная комбинация, то требуется ввести поправку -0110 (1001 в обратном коде или 1010 в дополнительном коде).

Так как поправки бывают положительные или отрицательные, их введение сопровождается блокировкой цепей межтетрадного переноса в период коррекции результата.

Пример 7.5. Сложить числа $A = 0010\,01001110$ и $B = 000111001111$, записанные в коде $Д_2$.

Решение. Сначала проводится потетрадное суммирование, а затем осуществляется коррекция

$$\begin{array}{rcccc} A = & & 0010 & 0100 & 1110 \\ B = & + & 0001 & 1100 & 1111 \\ \hline C = & & 0100 \leftarrow & 0001 \leftarrow & 1101 \end{array}$$

Результат получился без коррекции.

Ответ $C_n = 0100\,00011101_{Д_2}$

Пример 7.6. Сложить числа $A = 137 = 000100111101$ и $B = 457 = 010010111101$, записанные в коде $Д_2$.

Решение. Сначала проводится потетрадное суммирование, а затем коррекция:

$$\begin{array}{rcccc} A = & & 0001 & 0011 & 1101 \\ B = & + & 0100 & 1011 & 1101 \\ \hline & & 0101 & 1111 & 1010 \\ & + & 0110 & 0000 + & 1010 \quad \text{поправки} \\ \hline C = & & 1011 & 1111 & 0100 \end{array}$$

* Исключение из этого случая возникает при $a_i + b_i + \pi_{i-1} = 15$, когда не требуется вводить поправку, так как появится разрешенная комбинация.

В младшей тетраде возникает перенос $\pi_i = 1$ и запрещенная комбинация — вводится поправка 1010 в дополнительном коде; в старшей тетраде при $\pi_i = 0$ возникла запрещенная комбинация — поправка вводится 0110.

Ответ $C = 594 = 10111111\,0100_{Д_2}$.

При сложении чисел в коде $Д_4$ возможны следующие случаи.

Пусть $a_i'' = a_i + 3$, $b_i'' = b_i + 3$, где a_i'' и b_i'' — тетрады для кода $Д_4$. Тогда:

если $a_i'' + b_i'' + \pi_{i-1} < 10$, то $c_i'' = a_i + 3 + b_i + 3 + \pi_{i-1} = \underbrace{a_i + b_i + \pi_{i-1}}_{c_i} + 3 + 3$;

результат требует коррекции путем введения поправки -0011;

если $a_i'' + b_i'' + \pi_{i-1} \geq 10$, то $c_i'' = a_i + b_i + \pi_{i-1} + 3 + 3$.

Здесь возникает десятичный перенос, который, по условию, «уносит» с собой шесть избыточных комбинаций. Следовательно, в данном случае требуется коррекция результата путем введения поправки $+0011$.

Пример 7.7. Сложить числа $A = 35 = 01101000$ и $B = 28 = 01011011$, записанные в коде $Д_4$.

Р е ш е н и е. Сначала проводится потетрадное сложение, а затем введение поправок:

$$\begin{array}{rll} A = & 0110 & 1000 \\ B = + & 0101 & 1011 \\ \hline & 1100 \leftarrow & 0011 \\ - & 0011 + & 0011 \quad \text{поправки} \\ \hline C = & 1001 & 0110 \end{array}$$

Ответ $C = 63 = 1001\,0110_{Д_4}$.

Здесь поправки вводятся при блокировке цепей потетрадного переноса.

Правила введения попразок можно сформулировать так:

если при сложении тетрад не возникает переноса ($\pi_i = 0$), то поправка равна -0011 (или дополнение $+1101$); если же возникает потетрадный перенос ($\pi_i = 1$), то поправка равна $+0011$.

Аналогичным образом можно рассмотреть правила суммирования для других Д-кодов.

7.3. Представление отрицательных чисел в Д-кодах

Представление Д-кода в разрядной сетке машины может осуществляться в форме либо с фиксированной, либо с плавающей запятой. При этом отрицательные числа должны представляться в прямом, обратном или дополнительном кодах. Поэтому, если $A = -0, a_1 a_2 \ldots a_n$, где a_i — тетрады, то

$$[A]_{\text{пр}} = 1, a_1 a_2 \dots a_n;$$
$$[A]_{\text{об}} = 1, \overline{\overline{a}}_1 \overline{\overline{a}}_2 \dots \overline{\overline{a}}_n; \qquad (7.5)$$
$$[A]_{\text{д}} = 1, \overline{a}_1 \overline{a}_2 \dots \overline{a}_n,$$

где $\overline{\overline{a}}_i$ — дополнение до $q-1$ во всех тетрадах; $\overline{a}_i$ — дополнение до $q-1$ во всех тетрадах, за исключением младшей, для которой это дополнение до $q=10$.

Из правил преобразования (7.5) следует, что

$$a_i + a_i = q - 1. \qquad (7.6)$$

Это означает, что для Д-кодов, для которых выполняется условие (7.2), обратный код получается простым инвертированием набора тетрад.

Пример 7.8. Найти обратный и дополнительный коды в коде $Д_2$ для числа $A = -0{,}127 = -0{,}0001\,0010\,1101_{Д_2}$.

Р е ш е н и е. На основании (7.5) $[A]_{\text{об}} = 1{,}1110\,1101\,0010$.

Используя соотношение $[A]_{\text{об}} + q^{-n} = [A]_{\text{д}}$, находим дополнительный код: $[A]_{\text{д}} = 1{,}1110\,1101\,0011$.

Прибавление единицы в младшую тетраду при образовании дополнительного кода в коде $Д_2$ осуществляется по правилам сложения для этого кода.

Ответ $[A]_{\text{об}} = 1{,}1110\,1101\,0010_{Д_2}$; $[A]_{\text{д}} = 1{,}1110\,1101\,0011_{Д_2}$.

Пример 7.9. Найти обратный и дополнительный коды в коде $Д_4$ для числа $A = -0{,}4591 = 0{,}0111\,0001\,1000\,0100_{Д_4}$.

Р е ш е н и е. На основании (7.5)

$$[A]_{\text{об}} = 1{,}1000\ \ 0111\ \ 0011\ \ 1011;$$
$$[A]_{\text{д}} = 1{,}1000\ \ 0111\ \ 0011\ \ 1100. \qquad (а)$$

Прибавление единицы в младшую тетраду при образовании дополнительного кода в коде $Д_4$ не требует коррекции.

Ответ см. формулу (а) данного примера.

Код $Д_1$ отличается тем, что для него не выполняется условие (7.2). Эта особенность кода влияет на образование обратного или дополнительного кода, так как инвертирование набора тетрад означает получение дополнения до $2^4 - 1 = 15$. Следовательно, необходимо убрать разницу. Один из используемых при этом приемов состоит в том, что во все цифровые тетрады числа в коде $Д_1$ добавляется $+0110$ и после этого проводится инвертирование набора. Полученное изображение представляет собой обратный код числа.

Пример 7.10. Получить обратный код в коде $Д_1$ для числа $A = -0{,}256 = -0{,}0010\,0101\,0110_{Д_1}$.

Р е ш е н и е . Сначала во все тетрады добавляется 0110:

$$\begin{array}{rccc} & 0.0010 & 0101 & 0110 \\ + & & + & + \\ & 0110 & 0110 & 0110 \\ \hline & 0,1000 & 1011 & 1100 \end{array}$$

После инвертирования этого набора получаем $[A]_{об} = 1{,}0111\,0100\,0011_{Д_1}$.

Ответ. $[A]_{об} = 1{,}0111\,0100\,0011_{Д_1}$.

7.4. Выполнение операций сложения и вычитания чисел в Д-кодах

Все арифметические действия в Д-кодах выполняются над операндами по формальным правилам десятичной арифметики, сформулированным выше.

Возникающие при этом специфические особенности целесообразнее рассмотреть на конкретных примерах.

Пример 7.11. Сложить в коде $Д_1$ на сумматоре дополнительного кода числа $A = -0{,}1000\,0010\,0101$ и $B = 0{,}1001\,0100\,0110$.

Р е ш е н и е . Исходные числа представляются в дополнительном коде и их изображения складываются:

$$\begin{array}{rccccl} [A]_д = & 1. & 0001 & 0111 & 0101 & \\ [B]_д = + & 0. & 1001 & 0100 & 0110 & \\ \hline & 1. & 1010 & 1011 & 1011 & \\ + & & 0110 & 0110 & 0110 & \text{поправки} \\ \hline [C]_д = & 0,\leftarrow & 0001 & \leftarrow 0010 & \leftarrow 0001. & \end{array}$$

Ответ: $C = 0{,}0001\,0010\,0001_{Д_1}$.

Пример 7.12. Сложить в коде $Д_1$ на сумматоре обратного кода числа $A = -0{,}0100\,1000\,0111$ и $B = -0{,}0010\,0011\,0110$.

Р е ш е н и е . Исходные числа представляются в обратном коде и их изображения складываются:

$$\begin{array}{rccccl} [A]_{об} = & 1, & 0101 & 0001 & 0010 & \\ [B]_{об} = + & 1, & 0111 & 0110 & 0011 & \\ \hline & 0, & 1100 & 0111 & 0110 & \\ + & & 0110 & 0000 & 0000 & \text{поправки} \\ \hline [C]_{об} = & 1,\leftarrow & 0010 & 0111 & 0110. & \end{array}$$

При коррекции в коде $Д_1$ цепи межтетрадного переноса в сумматорах не блокируются.

Ответ получаем после преобразования результата из обратного кода.

Ответ. $C = -0{,}0111\,0010\,0011_{Д_1}$.

Пример 7.13. Сложить в коде $Д_2$ на сумматоре дополнительного кода числа $A = -0{,}0000\,1101\,0011\,1011$ и $B = -0{,}0000\,1111\,0010\,1011$.

Р е ш е н и е. Если все делать по правилам, то $[A]_д = 1{,}1111\,0010\,1100\,0101$. В последней тетраде получена запрещенная комбинация, что свидетельствует о необходимости коррекции путем введения поправки 0110:

$$\begin{array}{r} 0101 \\ +\,0110 \\ \hline 1011. \end{array}$$

Аналогично для второго числа получаем в последней тетраде 1011. Следовательно,

$$\begin{array}{lcccccl} [A]_д = & 1, & 1111 & 0010 & 1100 & 1011 & \\ [B]_д = + & 1, & 1111 & 0000 & 1101 & 1011 & \\ \hline & 1,\leftarrow & 1110 \leftarrow & 0011 \leftarrow & 1010 \leftarrow & 0110 & \\ & & + 0000 & + 0000 & + 1010 & + 1010 & \text{поправки} \\ \hline [C]_д = & 1, & 1110 & 0011 & 0100 & 0000. & \end{array}$$

Проводится обратное преобразование дополнительного кода.

Ответ: $C = -0{,}0001\,1100\,1100\,0000_{Д_2}$.

Пример 7.14. Сложить на сумматоре обратного кода (система $Д_2$) числа $A = 0{,}1110\,1011\,0011_{Д_2}$ и $B = -0{,}1100\,0100\,1011_{Д_2}$.

Р е ш е н и е. Сначала числа записываем в обратном коде и их изображения складываем:

$$\begin{array}{lcccccl} [A]_{об} = & 1, & 0011 & 1011 & 0100 & & \\ [B]_{об} = + & 0, & 1110 & 1011 & 0011 & & \\ \hline & 0,\leftarrow & 0010 \leftarrow & 0110 \leftarrow & 1000 & & \\ + & & 0000 & + 1010 & + 0110 & & \text{поправки} \\ \hline [C]_{об} = & 0, & 0010 & 0000 & 1110. & & \end{array}$$

Ответ $[C]_{об} = 0{,}0010\,0000\,1110_{Д_2}$.

Пример 7.15. Сложить в коде $Д_4$ на сумматоре обратного кода числа $A = -0{,}1011\,1100\,0111\,1001$ и $B = 0{,}0011\,1000\,1001\,1010$.

Р е ш е н и е. Сначала записываем в обратном коде и затем складываем изображения:

$$\begin{array}{lcccccl} [A]_{об} = & 1, & 0100 & 0011 & 1000 & 0110 & \\ [B]_{об} = + & 0. & 0011 & 1000 & 1001 & 1010 & \\ \hline & 1, & 0111 & 1100 \leftarrow & 0010 \leftarrow & 0000 & \\ & + & 1101 & + 1101 & + 0011 & + 0011 & \text{поправки} \\ \hline [C]_{об} = & 1, & 0100 & 1001 & 0100 & 0011. & \end{array}$$

При введении поправок цепи межтетрадного переноса блокируются и отрицательные поправки вносятся в виде дополнения (1101).

Ответ $C = -0{,}1011\,0110\,1011\,1100_{Д_4}$.

Рассмотренные выше примеры выполнения операций сложения в Д-кодах позволяют сделать ряд общих замечаний; коррекция результата может

осуществляться автоматически либо программным путем, либо с помощью аппаратных средств. Первый метод потребует разработки специального блока управления, а второй — усложнения схемы собственно сумматора. Эту задачу разработчик решает в конкретной постановке в зависимости от требований. На практике чаще используется схемный метод коррекции.

По изложенным выше правилам реализуются алгоритмы сложения (вычитания) чисел, представленных в форме с фиксированной запятой на специальных десятичных сумматорах.

Для десятичных чисел с плавающей запятой используют ту же методику, что и для двоичных чисел: порядки чисел перед сложением выравниваются (меньшему числу присваивается порядок большего числа) и по окончании операции проводится нормализация результата. При этом со стороны младших разрядов отводится дополнительная тетрада, используемая при сдвигах вправо.

7.5. Умножение чисел в Д-кодах

Выполнение операций умножения в Д-кодах принципиально проводится по классической схеме:

умножение чисел сводится к последовательному суммированию частных произведений, полученных при умножении множимого на очередную цифру множителя. Так как каждая цифра множителя представляется в виде $(\beta_4\beta_3\beta_2\beta_1)_i$, где i — номер разряда, умножение сопровождается расшифровкой значения очередной тетрады множителя и сдвигом на четыре разряда сразу. Расшифровку можно осуществить разными способами. Простейшим приемом является последовательное вычитание единицы из значения тетрады до получения нуля и соответственно прибавление множимого в сумматор на каждом такте. При умножении на сумматоре прямого кода надо предусмотреть дополнительную тетраду на случай местного переполнения.

Пример 7.16. Умножить на сумматоре прямого кода (код $Д_2$) числа

$$[A]_{пр} = 1{,}00111101;$$
$$[B]_{пр} = 0{,}00100100.$$

Р е ш е н и е. При умножении используются сумматор прямого кода для кода $Д_2$ на три тетрады и регистры на две тетрады.

Последовательность выполнения операции показана в таблице 7.2.

При умножении в коде $Д_1$ анализ тетрад можно осуществлять с помощью реверсивного счетчика.

Ответ: $[AB]_{пр} = 1{,}0000111011101110$.

Таблица 7.2

Сумматор	Регистр B	Примечание
0000 0000 0000	0010 0100	И. П. СМ := 0; РгB := $[B]_{пр}$; РгA := $[A]_{пр}$
+ 0011 1101	− 0001	
0011 1101	0011	
+ 0011 1101	− 0001	
1101 0100	0010	Анализ тетрады b_1
+ 0011 1101	− 0001	
0001 0001 0001	0001	
+ 0011 1101	− 0001	
	0000	Конец анализа
0001 0100 1110		Сдвиг на четыре разряда
0000 0001 0100	1110 0010	
+ 0011 1101	− 0001	Анализ тетрады b_2
1011 0001	0001	
+ 0011 1101	− 0001	
0000 0000 1110	0000	Конец анализа
	1110 1110	Сдвиг на четыре разряда
		Конец

В некоторых машинах единой системы при умножении десятичных чисел в коде $Д_1$ используется метод ускорения операции, при котором вся операция сводится к последовательному выполнению сложения или вычитания на сумматоре дополнительного кода в зависимости от величины очередной цифры множителя (табл. 7.3)

Таблица 7.3

Очередная цифра множителя	Вид операции	Очередная цифра множителя	Вид операции
0	0	5	$+2A+2A+1A$
1	$+1A$	6	$-2A-2A$
2	$+2A$	7	$-2A-1A$
3	$+2A+1A$	8	$-2A$
4	$+2A+2A$	9	$-1A$

Таким образом, предварительно должны быть подготовлены множимое и удвоенное множимое. При этом если предыдущая цифра была больше пяти, то действие на очередном шаге надо увеличить на $+1A$.

При умножении десятичных чисел часто используют другие приемы, позволяющие ускорить эту операцию, например, по формулам

$$\left.\begin{aligned} AB &= 2A\frac{B}{2}, \text{ если } B \text{ – четное число,} \\ AB &= 2A\frac{B-1}{2} + A, \text{ если } B \text{ – нечетное число.} \end{aligned}\right\} \qquad (7.7)$$

Действия в соответствии с (7.7) можно рассмотреть на примере десятичных чисел: $35 \cdot 42 = 2 \cdot 35 \cdot (42/2) = 70 \cdot 21 = 140 \cdot 10 + 70 = 280 \cdot 5 + 70 = 560 \cdot 2 + 280 + 70 = 1120 \cdot 1 + 280 + 70 = 1470$.

При двоично-десятичном представлении чисел прием (7.7) может быть упрощен, так как удвоение числа означает сдвиг влево, а деление на 2 — сдвиг вправо. Так как сдвиги проводятся над двоичными кодами, требуется коррекция тетрад на каждом шаге. Корректирующие поправки определяются для каждого Д-кода. Коррекция выполняется тогда, когда происходит сдвиг единицы из крайнего разряда данной тетрады в соседнюю тетраду. Например, для кода $Д_1$ корректирующая поправка равна $+0110$ для тетрад множимого и -0011 (или $+1101$) для тетрад множителя.

Пример 7.17 (см. [19]). Умножить числа $A = 0010\,0100$ и $B = 0100\,0011$ методом (7.7) в коде $Д_1$.

Р е ш е н и е. Схема умножения выглядит следующим образом:

	B		A				C				
	0100	0011	0000	0010	0100		+0000	0000	0010	0100	
(→)	0010	0001	0000	0100	1000	(←)	0000	0000	0100	1000	сдвиг
(→)	0001	0000	0000	1001	+0000	(←)	0000	0000	0110	+1100	сдвиг
					0110					0110	поправки
				1010	0110		0000	0000	0111	0010	
(→)	0000	+1000	0001	+0010	+1100	(←)+					сдвиг
		1101		0110	0110						поправки
	0000	0101	0001	1001	0010		0000	0001	1001	0010	
							0000	0010	+0000	0100	
									0110		поправки
(→)	0000	0010	0011	+0010	0100	(←)	0000	0010	0110	0100	
				0110							сдвиг
			0011	1000	0100						поправки
(→)	0000	0001	0111	+0000	1000	(←)+					
				0110							сдвиг
			0111	0110	1000						поправки
							0000	0111	0110	1000	
(→)	0000	0000	1110	1101	0000	(←)	0000	1001	+1100	+1100	сдвиг
							0110		0110	0110	поправки
							0000	+1010	0011	0010	
								0110			поправки
							0001	0000	0011	0010	

Конец.

Ответ: $AB = 0001\,0000\,0011\,0010$.

7.6. Деление чисел в Д-кодах

Деление десятичных чисел в Д-кодах выполняется методом последовательного вычитания делителя из делимого на первом шаге и из остатков — на последующих шагах. Вычитание на каждом шаге проводится до тех пор, пока не получится отрицательный остаток. Каждый раз при получении положительного остатка добавляется единица в специальный счетчик, где накапливается очередная цифра частного. Затем осуществляется сдвиг на четыре двоичных разряда и прибавление делителя до тех пор, пока не получится положительный остаток. Количество сложений (без последнего) является дополнением соответствующей цифры частного до 9, что заносится в счетчик очередной цифры частного.

Таким образом, процесс деления состоит из ряда последовательно чередующихся циклов сложения и вычитания со сдвигами. Знак частного получается как логическая сумма по модулю 2 знаков чисел.

Все действия при выполнении операции деления должны осуществляться на сумматоре дополнительного (обратного) кода, работающего по правилам сложения — вычитания соответствующего Д-кода.

Для простоты рассмотрим пример деления в десятичной системе счисления, не прибегая к представлению чисел в виде тетрад.

Пример 7.18 Разделить число $A = 0{,}154675$ на $B = 0{,}550$.

Решение Установка исходного положения.

Шаг 1 Осуществляется пробное вычитание: если результат положительный или равен нулю, то вырабатывается сигнал прерывания, если отрицательный, то проводится сдвиг на одну тетраду

В данном случае получается отрицательный остаток (рассматривается только цифровая часть чисел)

Шаг 2	154 675	СЧ := 0
	− 55 000	
	99 675	СЧ := 0 + 1
	− 55 000	
	44 675	СЧ := 1 + 1
	− 55 000	
Остаток < 0	89 675	$C_1 = 2$
Шаг 3:	89 675	СЧ := 9
	+ 5 500	
	95 175	СЧ := 9 − 1
	+ 5 500	
Остаток > 0	00 675	$C_2 = 8$

Шаг 4	00675	СЧ := 0
	− 550	
	00125	СЧ := 0 + 1
	− 550	
Остаток < 0	99575	$C_3 = 1$ и т. д.

Ответ $C = 0{,}281$

Алгоритм десятичного деления, подобный рассмотренному, используют в машинах ЕС ЭВМ.

Для ускорения операции деления применяют приемы, аналогичные приемам, употребляемым для ускорения выполнения операции умножения.

Пусть A — делимое, B — делитель, C —частное. Предположим, что удалось отыскать частное в виде $C = 0, c_1c_2 \ldots c_n$. Тогда

$$A = B \cdot (c_1 \cdot 2^{-1} + c_2 \cdot 2^{-1} + \ldots + c_n \cdot 2^n) + R_n, \qquad (7.8)$$

где R_n — остаток от деления.

Пусть $R_n = 0$. Положим, что $c_1 = 1$, $c_2 = c_3 = \ldots = 0$. Тогда остаток на первом шаге $R_1 = A - 2^{-1}B$.

Если $R_1 \geq 0$, то $c_1 = 1$; если $R_1 < 0$, то $c_1 = 0$. В последнем случае восстанавливается предыдущий положительный остаток. Затем принимаем, что $c_2 = 1$, а остальные $c_i = 0$ и т. д.

Остаток на любом шаге

$$R_i = A - B\sum_{i=1}^{n} c_i \cdot 2^{-i}. \qquad (7.9)$$

Таким образом, при делении десятичных чисел, заданных в Д-коде, результат получается в двоичной системе счисления.

Пример 7.19. Найти частное от деления $A = 0{,}2425$ на $B = 0{,}5200$.

Решение. Для наглядности выполним этот пример в десятичной системе счисления с записью результата в двоичном коде, предполагая, что при переходе к определенному Д-коду все операции над десятичными числами будут выполняться по правилам этого Д-кода.

Последовательность выполнения операции деления дана в таблице 7.4.

Ответ. C=0,011101.

Так как при сдвиге чисел, представленных в Д-коды, приходится вводить поправки, то для хранения делителя целесообразно иметь самостоятельный сумматор, в котором при переносе единицы из одной тетрады в другую автоматически вносится поправка.

Таблица 7.4

Делитель на i-м шаге	Сумматор	Примечания
0,5200	0,2425	И. П.
0,2600	−0,2600	$B\cdot 2^{-1}$; СМ := [СМ] − $B\cdot 2^{-1}$;
	9,9825	остаток отрицательный $c_1 = 0$;
	+0,2600	восстановление R_1;
0,1300	0,2425	$B\cdot 2^{-2}$; СМ := [СМ] − $B\cdot 2^{-2}$;
	−0,1300	
0,0650	0,1125	остаток положительный $c_2 = 1$;
	−0,0650	$B\cdot 2^{-3}$; СМ := [СМ] − $B\cdot 2^{-3}$;
0,325	0,0475	остаток положительный $c_3 = 1$;
0,0162	−0,0325	$B\cdot 2^{-4}$; СМ := [СМ] − $B\cdot 2^{-4}$;
	0,0150	остаток положительный $c_4 = 1$;
	−0,0162	$B\cdot 2^{-5}$; СМ := [СМ] − $B\cdot 2^{-5}$;
0,0081	9,9938	остаток отрицательный $c_5 = 0$;
	+0,0162	восстановление R_5;
	0,0150	$B\cdot 2^{-6}$; СМ := [СМ] − $B\cdot 2^{-6}$;
	−0,0081	
	0,0069	остаток положительный $c_6 = 1$;
	и т. д.	Конец

7.7. Извлечение квадратного корня в Д-кодах

В основу алгоритмов извлечения квадратного корня в десятичной системе может быть с некоторыми уточнениями положена формула (6.8). Если для двоичной системы счисления в качестве нулевого приближения берут величину $y_0 = 2^{E(m/2)}$, где $E(m/2)$ — целая часть числа $m/2$, а m — наибольший показатель степени основания системы, присутствующего в заданном числе, то для десятичной системы правильный выбор нулевого приближения определяет число итераций, необходимых для получения заданной точности. Как известно, точность результата ограничивается числом разрядов сумматора, на котором проводится действие. Так, например, если имеется шестиразрядный сумматор и на шаге i промежуточное значение результата y_i имеет одну верную цифру, то можно предположить, что y_{i+1} будет иметь по крайней мере две верных и одну сомнительную цифры, для получения которых используется пять разрядов сумматора. Для проверки и выявления правильных цифр необходимо получить их на $(i+2)$-м шаге. Но из-за ограниченности разрядной сетки на этом шаге можно полу-

чить только три цифры. Таким образом, на шестиразрядном сумматоре в результате можно получить две верных и одну сомнительную цифры. Из-за этого обстоятельства итерационный метод нахождения корня является нецелесообразным для двоично-десятичного представления чисел. Рассмотрим несколько иной подход.

В десятичной системе счисления целое число можно представить в виде степенного полинома по основанию 10:

$$A = a_n \cdot 10^n + \ldots + a_1 \cdot 10^1 + a_0 \cdot 10^0.$$

Квадрат числа A можно записать так:

$$A^2 = a_n^2 \cdot 10^{2n} + 2a_n \cdot 10^n (a_{n-1} \cdot 10^{n-1} + \ldots + a_1 \cdot 10 + a_0) + (a_{n-1} \cdot 10^{n-1} + \ldots$$
$$\ldots + a_1 \cdot 10 + a_0)^2 = a_n^2 \cdot 10^{2n} + 2a_n (a_{n-1} \cdot 10^{n-1} + \ldots + a_1 \cdot 10 + a_0) + a_{n-1}^2 \cdot 10^{2(n-1)} +$$
$$+ 2a_{n-1} + 10^{n-1} 10^n (a_{n-2} \cdot 10^{n-2} + \ldots + a_1 \cdot 10 + a_0) + \ldots + a_3^2 \cdot 10^6 +$$
$$+ 2a_3 \cdot 10^3 (a_2 \cdot 10^2 + a_1 \cdot 10 + a_0) + a_2^2 \cdot 10^4 + 2a_2 \cdot 10^2 (a_1 \cdot 10 +$$
$$+ a_0) + a_1^2 \cdot 10^2 \cdot 10^2 + 2a_1 \cdot 10 \cdot a_0 + a_0^2,$$

где n — любое целое число.

Раскроем в этом выражении скобки и сгруппирует члены по убывающим степеням:

$$A^2 = a_n^2 \cdot 10^{2n} + a_{n-1} \cdot 10^{n-1} (2a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1}) + a_{n-2} \cdot 10^{n-2} (2a_n \cdot 10^n +$$
$$+ 2a_{n-1} \cdot 10^{n-1} + a_{n-2} \cdot 10^{n-2}) + a_{n-3} \cdot 10^{n-3} (2a_n \cdot 10^n + \ldots + a_{n-3} \cdot 10^{n-3}) + \ldots$$
$$\ldots + a_0 (2a_n \cdot 10^n + 2a_{n-1} \cdot 10^{n-1} + \ldots + 2a_1 \cdot 10 + a_0).$$

Здесь на каждом последующем этапе содержимое в i-й скобке отличается от содержимого скобки на предыдущем этапе тем, что имеет дополнительные слагаемые $a_{n-i+1} \cdot 10^{n-i-1}$ и $a_{n-i} \cdot 10^{n-i}$.

Обозначим скобку на каждом этапе через b_i и примем $b_{n+1} = 0$ и $a_{n+1} = 0$, тогда

$$A^2 = (b_{n+1} + a_{n+1} \cdot 10^{n+1} + a_n \cdot 10^n) a_n \cdot 10^n + (b_n + a_n \cdot 10^n +$$
$$+ a_{n-1} \cdot 10^{n-1}) a_{n-1} \cdot 10^{n-1} + (b_{n-1} + a_{n-1} \cdot 10^{n-1} + a_{n-2} \cdot 10^{n-2}) a_{n-2} \cdot 10^{n-2} + \ldots$$
$$\ldots + (b_2 + a_2 \cdot 10^2 + a_1 \cdot 10) a_1 \cdot 10 + (b_1 + a_1 \cdot 10 + a_0) a_0.$$

Данное допущение возможно, так как первоначальная запись числа A в виде степенного полинома предполагает, что число A не имеет степеней, больших чем n.

Вынося из скобок b_i общие для скобки множители, получим

$$A^2 = (b'_{n+1} \cdot 10 + a_{n+1} \cdot 10 + a_n)a_n \cdot 10^{2n} + (b_n \cdot 10 + a_n \cdot 10 + a_{n-1})a_{n-1} \cdot 10^{2(n-1)} + \ldots + (b'_1 \cdot 10 + a_1 \cdot 10 + a_0)a_0.$$

Окончательно получим

$$A^2 = \sum_i (b'_{i+1} \cdot 10 + a_{i+1} \cdot 10 + a_i)a_i \cdot 10^{2i}, \qquad (7.10)$$

где b'_{i+1} — коэффициент при $a_{i+1} \cdot 10^{2(i+1)}$, т. е. полученный на предыдущем этапе.

Алгоритм ручного вычисления квадратного корня по данной формуле можно представить следующим образом:

1. Провести анализ двух старших разрядов числа A^2, найти число a_n, квадрат которого наиболее близко подходит к двум старшим разрядам числа A^2, оставаясь меньше последнего.

2. Провести вычитание из старших разрядов A^2 квадрата числа a_n.

3. Удвоить число a_n.

4. Сдвинуть остаток от вычитания на два разряда влево, а величину $2a_n$ — на один разряд влево.

5. Приписать справа от остатка вычитания два следующих старших разряда числа A^2.

6. Провести анализ полученного числа на равенство нулю.

7. Если полученное число не равно нулю, то, анализируя его, найти такое a_{n-1}, которое, будучи умноженным на $(2a_n \cdot 10 + a_{n-1})$, даст в результате число, меньшее полученного на пятом шаге, но наиболее близкое к нему по значению. Перейти к п. 3.

8. Если при анализе в п. 6 получено равенство, то перейти к п. 4, предварительно приписав справа от a_n нуль.

9. После получения количества цифр, равного $n/2$, прекратить вычисление.

При анализе в пп. 1, 7 можно использовать следующее соображение: если последовательно рассматривать квадраты чисел от 0 до 9, то переход от квадрата одного числа можно представить как прибавление к уже известному квадрату определенного числа, которое можно определить из формулы $c = a^2 - b^2 = (a-b)(a+b)$; если $a = b-1$, то $a^2 = b^2 + (a-b)(a+b) = b^2 + (a+b)$, где a — предыдущее число, возведенное в квадрат.

Данный метод позволяет повысить точность результата. На шестиразрядном сумматоре можно получить в результате пять точных цифр, так как не все число во время выполнения над ним действий располагается на сумматоре.

К недостаткам метода относят довольно длительное время, необходимое для получения результата.

7.8. Перевод чисел в Д-код

Рассмотрим некоторые вопросы перевода десятичных чисел, представленных в Д-коде, в двоичную систему счисления.

Пусть задано десятичное число $A = a_4a_3a_2a_1$, десятичная цифра, которая должна быть представлена в Д-коде в виде $a_i = \{\alpha'_4\alpha'_3\alpha'_2\alpha'_1\}_i$.

Используя равенство $10 = 8 + 2 = 2^3 + 2^1$, любое десятичное целое число можно записать как

$$A_д = (\ldots((\alpha''_4\alpha''_3\alpha''_2\alpha''_1)(2^3 + 2^1) + \alpha'''_4\alpha'''_3\alpha'''_2\alpha'''_1)(2^3 + 2^1) + \alpha''_4\alpha''_3\alpha''_2\alpha''_1)(2^3 + 2^1) + \alpha'_4\alpha'_3\alpha'_2\alpha'_1).$$

Умножение на 2^k означает сдвиг двоичного кода на k разрядов влево. Следовательно, перевод сводится к сдвигу соответствующих тетрад и их последующему суммированию. Это суммирование может быть выполнено по следующей схеме (для четырехразрядного числа):

α''_4	α''_3	α'''_2	α'''_1	α''_2	α'''_1	α''_1	α'''_1	α''_1	α'_1	α''_1	α'_1	α'_1
		α'''_4	α''_3	α''_3	α''_2	α''_2	α'''_2	α''_2	α'_1	α''_2	α'_2	
		α'''_4	α''_4	α''_4	α''_3	α''_3	α''_3	α''_3	α'_4	α'_3		
		α''_4	α''_3	α''_2	α''_4	α''_4	α''_4	α''_4		α'_4		
				α''_4	α''_1	α''_3	α''_2	α''_3				
				α''_4	α''_3	α''_2	α''_1	α''_4				
					α''_4	α''_4	α''_3					
					α''_3		α''_4					
12	11	10	9	8	7	6	5	4	3	2	1	0

В схеме некоторые символы встречаются многократно. Так как при переводе осуществляется суммирование по столбцам, то пары одинаковых символов дадут единицы переноса в соседние разряды.

Таким образом, перевод числа в Д-коде осуществляется путем суммирования элементов тетрад по столбцам с передачей соответствующих переносов.

Подобные способы перевода реализованы в машинах ЕС ЭВМ, машинах фирмы «IBM» и т. д. При разработке схем перевода приходится решать вопросы создания суммирующего устройства на много входов. При переводе, например, восьмиразрядного десятичного числа количество слагаемых в столбце оказывается равным 13. Значит, надо иметь сумматор на 13 входов. Реализовать такую схему можно с помощью многоступенчатых схем. При этом возникают дополнительные задержки сигнала, что снижает скорость перевода чисел.

Перевод числа из двоичной системы счисления в Д-код может осуществляться разными способами. В некоторых случаях для ряда последовательных операций над двоичным изображением числа может быть использована сама вычислительная машина (например, деление на число 1010 целых двоичных чисел; десятичные цифры получаются последовательно одна за другой. При дробных числах эта операция видоизмеияется таким образом, чтобы при умножении на число 1010 можно было получить соответствующие цифры десятичных дробей).

Алгоритмы перевода чисел из двоичной системы счисления в Д-код могут быть реализованы схемными или программными способами. Схемные способы перевода десятичных чисел в Д-код или из Д-кода в двоичную систему счисления и обратно весьма перспективны.

Задание для самоконтроля

1. Какие комбинации являются запрещенными для кодов $Д_1$, $Д_2$, $Д_5$?

2. Преобразовать число $A = -0{,}6315$ в дополнительный код в кодах $Д_1$ и $Д_2$.

3. Преобразовать число $B = -0{,}1234$ в обратный код в кодах $Д_2$ и $Д_4$.

4. Сложить числа $A = -0{,}6315$ и $B = 0{,}1234$ на сумматоре дополнительного кода в коде $Д_1$.

5. Сложить числа $A = 0{,}6315$ и $B = -0{,}1234$ на сумматоре обратного кода в коде $Д_2$.

6. Сложить числа $A = 0{,}1245$ и $B = -0{,}1246$ на сумматоре дополнительного кода в коде $Д_4$.

7. Перемножить числа $A = 0{,}12$ и $B = 0{,}13$ на сумматоре прямого кода в коде $Д_1$.

8. Разделить число $A = 0{,}1246$ на $B = 0{,}13$ на сумматоре дополнительного кода в коде $Д_2$.

9. Перемножить числа $A = 0{,}146$ и $B = 0{,}178$ ускоренным методом по (7.7) в коде $Д_1$ (сумматор обратного кода).

10. Извлечь квадратный корень из числа $A = 0{,}14412$ на сумматоре обратного кода в коде $Д_1$

8. КОНТРОЛЬ РАБОТЫ ЦИФРОВОГО АВТОМАТА

8.1. Кодирование информации как средство обеспечения контроля работы автомата

Коды как средство тайнописи появились в глубокой древности. Известно, что еще древнегреческий историк Геродот в V в. до н. э. приводил примеры писем, понятных лишь адресату. Секретная азбука использовалась Юлием Цезарем. Над созданием различных секретных шифров работали такие известные ученые средневековья, как Ф. Бэкон, Д. Кардано и др. Появлялись очень хитрые шифры и коды, которые, однако, с течением времени расшифровывались и переставали быть секретом. Первым кодом, предназначенным для передачи сообщений по каналам связи, был код С. Морзе, содержащий разное количество символов для кодирования букв и цифр. Затем появился код Ж. Бодо, используемый в телеграфии, в котором все буквы или цифры содержат одинаковое количество символов. В качестве символов может выступать наличие или отсутствие (пробел) импульса в электрической цепи в данный момент.

Коды, использующие два различных элементарных сигнала, называются *двоичными*. Эти сигналы удобно обозначать символами 0 и 1. Тогда кодовое слово будет состоять из последовательностей нулей и единиц.

Двоичное кодирование тесно связано с принципом *дихотомии*, который реализуется в графическом методе представления двоичной информации в виде графов.

В гл. 1 и 3 были рассмотрены общие и конкретные вопросы кодирования информации в цифровом автомате. Однако эти методы сами по себе еще не обеспечивают правильность выполнения того или иного алгоритма. Рассмотренные ранее 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 меры, сигнализирующие о появлении ошибки. Следовательно, с одной стороны, разработчиками машины

должны быть предусмотрены меры 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-либо детали, либо отклонением от нормы параметров, например изменением 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 строиться с таким расчетом, чтобы она позволяла обнаружить и по возможности исправить любые нарушения. При этом надо различать следующие виды ошибок результата:

1) возникающие из-за погрешностей в исходных данных;

2) обусловленные методическими погрешностями;

3) появляющиеся из-за возникновения неисправностей в работе машины.

Первые два вида ошибок не являются объектом для работы системы контроля. Погрешности перевода или представления числовой информации в разрядной сетке автомата приведут к возникновению погрешности в результате решения задачи. Эту погрешность можно заранее рассчитать и, зная ее максимальную величину, правильно выбрать длину разрядной сетки машины. Методические погрешности также учитываются предварительно.

Проверка правильности функционирования отдельных устройств машины и выявление неисправностей может осуществляться по двум направлениям:

профилактический контроль, задача которого — предупреждение появления возможных ошибок в работе;

оперативный контроль, задача которого — проверка правильности выполнения машиной всех операций.

Решение всех задач контроля становится возможным только при наличии определенной *избыточности* информации. Избыточность может быть создана либо аппаратными (схемными) средствами*, либо логическими или информационными средствами.

* Схемная избыточность будет рассматриваться в дисциплинах, в которых изучаются устройства ЭВМ. В данной главе описаны методы логического контроля, использующие информационную избыточность.

К методам логического контроля, например, можно отнести следующие приемы. В ЭВМ первого и второго поколения 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.

Если в процессе решения какой-то задачи вычисляются тригонометрические функции, то для контроля можно использовать известные соотношения между этими функциями, например $\sin^2\alpha + \cos^2\alpha = 1$. Если это соотношение выполняется с заданной точностью на каждом шаге вычислений, то можно с уверенностью считать, 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.

8.2. Основные понятия теории кодирования

Задача кодирования информации представляется как некоторое преобразование числовых данных в заданной системе счисления. В частном случае эта операция может быть сведена к группированию символов (представление в виде триад или тетрад) или представлению в виде символов (цифр) позиционной системы счисления. Так как любая позиционная система не несет в себе избыточности информации и все кодовые комбинации являются разрешенными, 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 обнаружения и исправления ошибки. Количественно корректирующая способность кода определяется вероятностью обнаружения или исправления ошибки. Если имеем n-разрядный код и вероятность искажения одного символа p, то вероятность того, что искажены k символов, а остальные $n-k$ символов не искажены, по теореме умножения вероятностей будет

$$w = p^k (1-p)^{n-k}. \tag{8.1}$$

Число кодовых комбинаций, каждая из которых содержит k искаженных элементов, равна числу сочетаний из n по k:

$$C_n^k = \frac{n!}{k!(n-k)!}. \tag{8.2}$$

Тогда полная вероятность искажения информации

$$P_\Sigma = \sum_{i=1}^{k} \frac{n!}{i!(n-i)!} p^i (1-p)^{n-i}. \tag{8.3}$$

Так как на практике $p = 10^{-3} \ldots 10^{-4}$, 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** $d(A, B)$ для кодовых комбинаций A и B определяется как вес третьей кодовой комбинации, которая получается поразрядным сложением исходных комбинаций по модулю 2.

Вес кодовой комбинации $V(A)$ — количество единиц, содержащихся в кодовой комбинации.

Пример 8.1. Найти вес и кодовое расстояние для комбинаций $A = 011011100$, $B = 100111001$

Р е ш е н и е Вес для кодовых комбинаций $V(A) = \sum_{i=1}^{9} a_i = 5$; $V(B) = \sum_{i=1}^{9} b_i = 5$.

Находим кодовую комбинацию $C = A \oplus B = 111100101$, для которой определяется вес, равный кодовому расстоянию для A и B: $V(C) = d(A, B) = \sum_{i=1}^{9} c_i = 6$.

Ответ $d(A, B) = 6$.

* Это определение совпадает с понятием кодового расстояния по Хэмингу. Поэтому в теории кодирования оно называется хэминговым расстоянием.

Коды можно рассматривать и как некоторые геометрические (пространственные) фигуры. Например, триаду можно представить в виде единичного куба, имеющего координаты вершин, которые отвечают двоичным символам (рис. 8.1). 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 (рис. 8.2, *а*).

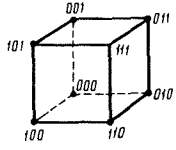


Рис. 8.1. Геометрическое представление кодов

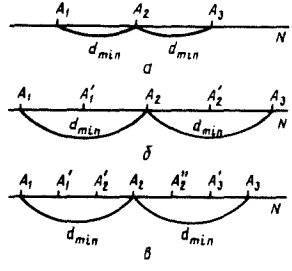


Рис. 8.2. Кодовые расстояния

В теории кодирования [10] показано, что систематический код способен обнаружить ошибки только тогда, когда минимальное кодовое расстояние для него больше или равно $2t$, т. е.

$$d_{\min} \geq 2t, \tag{8.4}$$

где t — кратность обнаруживаемых ошибок (в случае одиночных ошибок $t = 1$ и т. д.).

Это означает, что между соседними разрешенными кодовыми словами должно существовать по крайней мере одно кодовое слово (рис. 8.2, *б*, *в*).

В тех случаях, когда необходимо не только обнаружить ошибку, но и исправить ее (т. е. указать место ошибки), минимальное кодовое расстояние должно быть

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

Существуют коды, в которых невозможно выделить абсолютную избыточность. Пример таких кодов — Д-коды, где количество разрешенных комбинаций меньше количества возможных комбинаций. Неявная избы-

точность характерна также для кодов типа «k из n». Примером является код «2 из 5», который часто используется для представления информации. Суть его в том, что в слове из пяти разрядов только два разряда имеют единичное значение.

В общем случае количество n-разрядных слов, имеющих k единичных разрядов, можно оценить по формуле (8.2).

8.3. Методы эффективного кодирования информации

Информационную избыточность можно ввести разными путями. Рассмотрим один из путей эффективного кодирования.

В ряде случаев буквы сообщений преобразуются в последовательности двоичных символов. Учитывая статистические свойства источника сообщения, можно минимизировать среднее число двоичных символов, требующихся для выражения одной буквы сообщения, что при отсутствии шума позволит уменьшить время передачи.

Такое эффективное кодирование базируется на *основной теореме Шеннона* для каналов без шума, в которой доказано, что сообщения, составленные из букв некоторого алфавита, можно закодировать так, что среднее число двоичных символов на букву будет сколь угодно близко к энтропии источника этих сообщений, но не меньше этой величины.

Теорема не указывает конкретного способа кодирования, но из нее следует, что при выборе каждого символа кодовой комбинации необходимо стараться, чтобы он нес максимальную информацию. Следовательно, каждый символ должен принимать значения 0 и 1 по возможности с равными вероятностями и каждый выбор должен быть независим от значений предыдущих символов.

При отсутствии статистической взаимосвязи между буквами конструктивные методы построения эффективных кодов были даны впервые К. Шенноном и Н. Фано. Их методики существенно не различаются, поэтому соответствующий код получил название кода Шеннона–Фано.

Код строится следующим образом: буквы алфавита сообщений выписываются в таблицу в порядке убывания вероятностей. Затем они разделяются на две группы так, чтобы суммы вероятностей в каждой из групп были по возможности одинаковы. Всем буквам верхней половины в качестве первого символа приписывается 1, а всем нижним — 0. Каждая из полученных групп, в свою очередь, разбивается на две подгруппы с одинаковыми суммарными вероятностями и т. д. Процесс повторяется до тех пор, пока в каждой подгруппе останется по одной букве.

Рассмотрим алфавит из восьми букв (табл. 8.1). Ясно, что при обычном (не учитывающем статистических характеристик) кодировании для представления каждой буквы требуется три символа.

Таблица 8.1

Буквы	Вероятности	Кодовые комбинации
z_1	0,22	11
z_2	0,20	101
z_3	0,16	100
z_4	0,16	01
z_5	0,10	001
z_6	0,10	0001
z_7	0,04	00001
z_8	0,02	00000

Таблица 8.2

Буквы	Вероятности	Кодовые комбинации
z_1	0,22	11
z_2	0,20	10
z_3	0,16	011
z_4	0,16	010
z_5	0,10	001
z_6	0,10	0001
z_7	0,04	00001
z_8	0,02	00000

Используя формулу (1.6), вычислим энтропию набора букв:

$$(z) = -\sum_{i=1}^{8} p(z_i) \log p(z_i) \approx 2{,}76$$

и среднее число символов на букву

$$l_{\text{ср}} = \sum_{i=1}^{8} p(z_i)\, n(z_i) \approx 2{,}84,$$

где $n(z_i)$ — число символов в кодовой комбинации, соответствующей букве z_i.

Значения z и $l_{\text{ср}}$ не очень различаются по величине.

Рассмотренная методика Шеннона–Фано не всегда приводит к однозначному построению кода. Ведь при разбиении на подгруппы можно сделать большей по вероятности как верхнюю, так и нижнюю подгруппу.

Множество вероятностей в предыдущей таблице можно было разбить иначе (табл. 8.2).

При этом среднее число символов на букву оказывается равным 2,80. Таким образом, построенный код может оказаться не самым лучшим. При построении эффективных кодов с основанием $q > 2$ неопределенность становится еще больше.

От указанного недостатка свободна методика Д. Хаффмена. Она гарантирует однозначное построение кода с наименьшим для данного распределения вероятностей средним числом символов на букву.

Таблица 8.3

Буквы	Вероятности	Вспомогательные столбцы						
		1	2	3	4	5	6	7
z_1	0,22	0,22	0,22	→0,26	→0,32	→0,42	→0,58	→1
z_2	0,20	0,20	0,20	0,22	0,26	0,32	0,42	
z_3	0,16	0,16	0,16	0,20	0,22	0,26		
z_4	0,16	0,16	0,16	0,16	0,20			
z_5	0,10	0,10	→0,16	0,16				
z_6	0,10	0,10	0,10					
z_7	0,04	→0,06						
z_8	0,02							

Для двоичного кода методика сводится к следующему. Буквы алфавита сообщений выписываются в основной столбец в порядке убывания вероятностей. Две последние буквы объединяются в одну вспомогательную букву, которой приписывается суммарная вероятность. Вероятности букв, не участвовавших в объединении, и полученная суммарная вероятность снова располагаются в порядке убывания вероятностей в дополнительном столбце, а две последние объединяются. Процесс продолжается до тех пор, пока не получим единственную вспомогательную букву с вероятностью, равной единице.

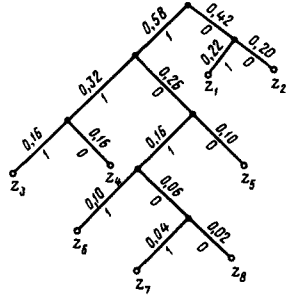


Рис. 8.3. Кодовое дерево

Для составления кодовой комбинации, соответствующей данному сообщению, необходимо проследить путь перехода сообщений по строкам и столбцам таблицы. Для наглядности строится кодовое дерево. Из точки, соответствующей вероятности 1, направляются две ветви, причем ветви с большей вероятностью присваивается символ 1, а с меньшей — 0. Такое последовательное ветвление продолжаем до тех пор, пока не дойдем до каждой буквы (рис. 8.3).

Теперь, двигаясь по кодовому дереву сверху вниз, можно записать для каждой буквы соответствующую ей кодовую комбинацию:

z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
01	00	111	110	100	1011	10101	10100

8.4. Кодирование по методу четности-нечетности

Если в математическом коде выделен один контрольный разряд ($k = 1$), то к каждому двоичному числу добавляется один избыточный разряд и в него записывается 1 или 0 с таким условием, чтобы сумма цифр в каждом числе была по модулю 2 равна 0 для случая четности или 1 для случая нечетности. Появление ошибки в кодировании обнаружится по нарушению четности (нечетности). При таком кодировании допускается, что может возникнуть только одна ошибка. В самом деле, для случая четности правильной будет только половина возможных комбинаций. Чтобы одна допустимая комбинация превратилась в другую, должно возникнуть по крайней мере два нарушения или четное число нарушений. Пример реализации метода четности представлен в таблице 8.4.

Таблица 8.4

Число	Контрольный разряд	Проверка
10101011	1	0
11001010	0	0
10010001	1	0
11001011	0	1 — нарушение

Такое кодирование имеет минимальное кодовое расстояние, равное 2.

Можно представить и несколько видоизмененный способ контроля по методу четности — нечетности. Длинное число разбивается на группы, каждая из которых содержит l 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}	

Увеличение избыточности информации приводит к тому, что появляется возможность не только обнаружить ошибку, но и исправить ее. Пусть произошла неисправность в каком-то из разрядов этого числа (представим, что разряд a_{18} изменил состояние, т. е. $a_{18} = 1$). Это приведет к тому, что

при проверке на четность сумма $\sum_i (a_i + k_i)$ по соответствующим строкам и столбцам изменится для значений, которые содержат элемент a_{18}, т. е. это будет четвертая сверху строка и третий слева столбец. Следовательно, нарушение четности по этой строке и столбцу можно зафиксировать, что в конечном счете означает обнаружение не только самой ошибки, но и места, где возникла ошибка. Изменив содержимое отмеченного разряда (в данном случае a_{18}) на противоположное, можно исправить ошибку.

Пример 8.1. Определить и исправить ошибку в передаваемой информации вида

1001110	0
1110101	0
0101101	0
1010110	0
1101011	1
0001011	

Для контроля использовать метод четности по строкам и столбцам (контрольный столбец 8, контрольная строка 6).

Решение. Прежде всего осуществим проверку на четность по каждой строке: $k_1 = 0$, $k_2 = 1$; $k_3 = 0$, $k_4 = 0$; $k_5 = 0$.

Затем проверим на четность информацию по столбцам: $k_6 = 0$; $k_7 = 1$; $k_8 = 0$; $k_9 = 0$; $k_{10} = 0$, $k_{11} = 0$; $k_{12} = 0$

Проверка показывает, что ошибка возникла в разряде второй строки и второго слева столбца. Следовательно, разряд, содержащий ошибочную информацию, находится на пересечении второй строки и второго столбца.

Ответ

1001110	0
1010101	0
0101101	0
1010110	0
1101011	1
0001011	

Контроль по методу четности–нечетности широко используют в ЭВМ для контроля записи, считывания информации в запоминающих устройствах на магнитных носителях, а также при выполнении арифметических операций.

8.5. Коды Хэмминга

Коды, предложенные американским ученым Р. Хэммингом, способны не только обнаружить, но и исправить одиночные ошибки. Эти коды — *систематические*.

Предположим, что имеется код, содержащий m информационных разрядов и k контрольных разрядов. Запись на k позиций определяется при проверке на четность каждой из проверяемых k групп информационных символов. Пусть было проведено k проверок. Если результат проверки свидетельствует об отсутствии ошибки, то запишем 0, если есть ошибка, то запишем 1. Запись полученной последовательности символов образует двоичное, контрольное число, указывающее номер позиции, где произошла ошибка. При отсутствии ошибки в данной позиции последовательность будет содержать только нули. Полученное таким образом число описывает $n = (m + k + 1)$ событий. Следовательно, справедливо неравенство

$$2^k \geq (m + k + 1). \qquad (8.5)$$

Определить максимальное значение m для данного k можно из следующего:

n	1	2	3	4...	8...15	16...31	32...63	64
m	0	0	1	1...	4...11	11...26	26...57	57
k	1	2	2	3...	4...4	5...5	6...6	7

Определим теперь позиции, которые надлежит проверить в каждой из k проверок. Если в кодовой комбинации ошибок нет, то контрольное число содержит только нули. Если в первом разряде контрольного числа стоит 1, то, значит, в результате первой проверки обнаружена ошибка. Имея таблицу двоичных эквивалентов для десятичных чисел, можно сказать, что, например, первая проверка охватывает позиции 1, 3, 5, 7, 9 и т. д., вторая проверка — позиции 2, 3, 6, 7, 10.

Проверка	*Проверяемые разряды*
1...	1, 3, 5, 7, 9, 11, 13, 15...
2...	2, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23...
3...	4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23...
4...	8, 9, 10, 11, 12, 13, 14, 15, 24...
..	...

Теперь нужно решить, какие из позиций целесообразнее применить для передачи информации, а какие — для ее контроля. Преимущество использования позиций 1, 2, 4, 8, ... для контроля в том, что данные позиции встречаются только в одной проверяемой группе символов.

В таблице 8.5 представлены примеры кодирования информации по методу Хэминга для семиразрядного кода.

Как видно из таблицы 8.5, в этом случае $n = 7$, $m = 4$, $k = 3$ и контрольными будут разряды 1, 2, 4.

По методу Хэминга могут быть построены коды разной длины. При этом чем больше длина кода, тем меньше относительная избыточность . Например, для контроля числа, имеющего 48 двоичных разрядов, потребуется только шесть дополнительных (избыточных) разрядов. Коды Хэминга используют в основном для контроля передачи информации по каналам связи, что имеет место в вычислительных системах с телеобработкой данных или в системах коллективного пользования.

Таблица 8.5

Разряды двоичного кода							Кодируемая десятичная информация
1	2	3	4	5	6	7	
k_1	k_2	m_1	k_3	m_2	m_3	m_4	
0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1
0	1	0	1	0	1	0	2
1	0	0	0	0	1	1	3
1	0	0	1	1	0	0	4
0	1	0	0	1	0	1	5
1	1	0	0	1	1	0	6
0	0	0	1	1	1	1	7
1	1	1	0	0	0	0	8
0	0	1	1	0	0	1	9
1	0	1	1	0	1	0	10
0	1	1	0	0	1	1	11
0	1	1	1	1	0	0	12
1	0	1	0	1	0	1	13
0	0	1	0	1	1	0	14
1	1	1	1	1	1	1	15

Пример 8.2. Определить правильность передачи информации $A = 0{,}111000$ по каналу, если для контроля использован метод Хэминга.

Решение Прежде всего определим контрольное число, проводя проверки по правилам, указанным на с. 90: $k_1 = 1; k_2 = 0; k_3 = 1$.

Контрольное число говорит о том, что произошла ошибка в разряде 5.

Ответ правильная информация $A = 0{,}111100$.

8.6. Контроль по модулю

Разнообразные задачи можно решать с помощью метода контроля, основанного на свойствах сравнений. Развитые на этой основе методы контроля арифметических и логических операций называют *контролем по модулю.*

Рассмотрим основные положения из теории сравнений.

Если целым числам A и B соответствует один и тот же остаток от деления на третье число p, то числа A и B равноостаточны друг другу по модулю p или сравнимы по модулю p:

$$A \equiv B \,(\mathrm{mod}\, p). \tag{8.6}$$

Сравнения — уравнения типа (8.6).

Сравнимость двух чисел равносильна возможности представить их в алгебраическом виде

$$A = B + pl. \tag{8.7}$$

Сравнения обладают рядом свойств:

1. Сравнения можно почленно складывать. Если $A_1 \equiv B_1 (\mathrm{mod}\, p)$; $A_2 \equiv B_2 (\mathrm{mod}\, p)$; ...; $A_n \equiv B_n (\mathrm{mod}\, p)$, то $A_1 + A_2 + \ldots + A_n \equiv B_1 + B_2 + \ldots + B_n (\mathrm{mod}\, p)$.

Следовательно, слагаемое, стоящее в какой-либо части сравнения, можно переносить в другую часть, поменяв при этом его знак, т. е. $A + B \equiv C (\mathrm{mod}\, p)$ или $A \equiv C - B (\mathrm{mod}\, p)$.

2. Два числа, сравнимые с третьим числом, сравнимы и между собой: если $A \equiv B (\mathrm{mod}\, p)$; $C \equiv B (\mathrm{mod}\, p)$, то $A \equiv C (\mathrm{mod}\, p)$.

3. Сравнения можно почленно перемножить. Пусть $A_1 \equiv B_1 (\mathrm{mod}\, p)$; $A_2 \equiv B_2 (\mathrm{mod}\, p)$. Тогда на основании (8.7) $A_1 = B_1 + l_1 p$; $A_2 = B_2 + l_2 p$.

После умножения получаем $A_1 A_2 = B_1 B_2 + B_1 l_2 p + B_2 l_1 p + l_1 l_2 pp$. Следовательно, $A_1 A_2 = B_1 B_2 + Np$, или в общем случае:

$$A_1 A_2 A_3 \ldots A_m \equiv B_1 B_2 B_3 \ldots B_m \,(\mathrm{mod}\, p).$$

Из свойства 3 также следует, что обе части сравнения можно умножить на одно и то же целое число.

Пусть $A \equiv B (\mathrm{mod}\, p)$; $K \equiv K (\mathrm{mod}\, p)$. Тогда $AK \equiv BK (\mathrm{mod}\, p)$.

4. Обе части сравнения и модуль можно умножить на одно и то же число: $A = B + lp$; $Am = Bm + mlp$, т. е. $Am = Bm (\mathrm{mod}\, mp)$.

5. Обе части сравнения и модуль можно разделить на любой общий делитель. Пусть $A \equiv B (\mathrm{mod}\, p)$, где $A = ad$, $B = bd$, $P = p_1 d$. Тогда $A = B + lp$.

Подставив в это выражение значения A, B и P, получим

$$ad = bd + lp_1 d.$$

Разделив уравнение на d, получим $a = b + lp_1$, т. е. $a = b (\mathrm{mod}\, p_1)$.

6. Обе части сравнения можно возвести в степень. Если $A \equiv B(\mathrm{mod}\, p)$, то $A^n \equiv B^n(\mathrm{mod}\, p)$.

Из свойства 6 следует, что над сравнениями можно провести операцию извлечения корня n-й степени.

Рассмотренные выше свойства сравнений используются для осуществления операции контроля.

Существуют два метода получения контрольного кода: числовой и цифровой.

Числовой метод контроля. При числовом методе контроля код заданного числа определяется как наименьший положительный остаток от деления числа на выбранный модуль p:

$$r_A = A - \{A/p\}p, \tag{8.8}$$

где в фигурных скобках {} — целая часть от деления числа; A — контролируемое число.

Величина модуля p существенно влияет на качество контроля; если $p = q$ (q — основание системы счисления, в которой выражено число) и имеет место числовой контроль, то контролируется только младший разряд числа и контроль как таковой не имеет смысла; для $p = q^m$ справедливы аналогичные соображения, так как если $m < n$, то опять не все разряды числа участвуют в контроле и ошибки в разрядах старше m вообще не воспринимаются.

При числовом методе контроля по модулю p для определения остатка используют операцию деления, требующую больших затрат машинного времени. Для числового метода контроля справедливы основные свойства сравнений (сложение, умножение сравнений и т. д.). Поэтому, если $A \equiv r_1(\mathrm{mod}\, p)$; $B \equiv r_B(\mathrm{mod}\, p)$, где $0 \le r_A \le p-1$; $0 \le r_B \le p-1$, то $A + B \equiv r_1 + r_B(\mathrm{mod}\, p)$. Отсюда

$$r_{A+B} \equiv r_A + r_B(\mathrm{mod}\, p). \tag{8.9}$$

Аналогичным образом доказывается справедливость и следующих соотношений:

$$r_{A-B} \equiv r_A - r_B(\mathrm{mod}\, p); \tag{8.10}$$

$$r_{AB} \equiv r_A r_B(\mathrm{mod}\, p). \tag{8.11}$$

Пример 8.3. Для заданных чисел $A = 125$ и $B = 89$ определить контрольные коды самих чисел, их суммы и разности, если модуль $p = 11$.

Решение. Контрольные коды чисел определяем по (6.4):

$$r_A = 125 - \{125/11\}11 = 4;\ r_B = 89 - \{89/11\}11 = 1.$$

Аналогично находим контрольные коды для суммы и разности:

$$A + B = 214,\ r_{A+B} = 214 - \{214/11\}11 = 5;$$

$$A - B = 36,\ r_{A-B} = 36 - \{36/11\}11 = 3.$$

Проверку правильности определения контрольных кодов суммы и разности можно провести на основании (6.5) и (6.6):

$$r_{A+B} = 4 + 1 \equiv 5 (\mathrm{mod}\, 11);\ r_{A-B} = 4 - 1 \equiv 3 (\mathrm{mod}\, 11)$$

Ответ. $r_A = 4;\ r_B = 1;\ r_{A+B} = 5;\ r_{A-B} = 3$.

Цифровой метод контроля. При цифровом методе контроля контрольный код числа образуется делением суммы цифр числа на выбранный модуль:

$$r'_A = \sum_i a_i - \left\{ \frac{\sum_i a_i}{p} \right\} p$$

или

$$r' \equiv \sum_i a_i (\mathrm{mod}\, p). \qquad (8.12)$$

Возможны два пути получения контрольного кода: 1) непосредственное деление суммы цифр на модуль p; 2) суммирование цифр по модулю p.

Второй путь проще реализуется, так как если $a_i < p$, то контрольный код получается только операцией суммирования.

Пример 8.4. Определить контрольные коды чисел $A = 153$ и $B = 41$, их суммы и разности, если $p = 11$.

Р е ш е н и е. Контрольные коды исходных чисел определяем по (8.12). Для этого находим суммы цифр и делим их на модуль: $\sum_i a_i = 9$; $\sum_i b_i = 5$.

Следовательно, $r'_A = 9;\ r'_B = 5$.

Аналогично определяем контрольные коды суммы и разности:

$$C = A + B = 194;\ \Sigma c_i = 14;\ r'_C \equiv 3 (\mathrm{mod}\, 11);$$

$$D = A - B = 112;\ \Sigma d_i = 4;\ r'_D \equiv 4 (\mathrm{mod}\, 11).$$

Ответ. $r'_A = 9,\ r'_B = 5,\ r'_{A+B} = 3,\ r'_{A-B} = 4$.

Однако при цифровом методе свойства сравнений не всегда справедливы, и происходит это из-за наличия переносов (заемов) при выполнении арифметических действий над числами. Поэтому нахождение контрольного кода результата операции происходит обязательно с коррекцией.

Пусть заданы числа A и B и соответственно их контрольные коды

$$r'_A \equiv \sum_i a_i (\mathrm{mod}\, p);\ r'_B = \sum_i b_i (\mathrm{mod}\, p);\ C = A + B.$$

Найдем контрольный код r'_C.

Видимо, когда есть результат операции, тогда найти r'_C методом суммирования цифр по модулю не сложно. Какова будет возможность получения r'_C через контрольные коды слагаемых?

Сумму цифр c_i числа можно найти, зная цифры a_i и b_i и количество переносов в каждом разряде. Каждый перенос уносит из данного разряда q единиц и добавляет одну единицу в следующий разряд, т. е. сумма цифр уменьшится на величину $q-1$ на каждый перенос. Тогда

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i - l(q-1), \qquad (8.13)$$

где l — количество переносов, возникших при сложении.

Так как $r'_A \equiv \sum_i a_i (\mathrm{mod}\, p);\ r'_B \equiv \sum_i b_i (\mathrm{mod}\, p)$, то $r'_C \equiv \sum_i c_i (\mathrm{mod}\, p)$.

Подставив эти значения в (8.13), получим

$$r'_C = \left[r'_A + r'_B - l(q-1)\right](\mathrm{mod}\, p). \qquad (8.14)$$

Аналогичными рассуждениями можно показать, что для разности чисел $C = A - B$

$$r'_C \equiv \left[r'_A - r'_B + s(q-1)\right](\mathrm{mod}\, p), \qquad (8.15)$$

где s – количество заемов при выполнении операции.

Пример 8.5. Определить контрольные коды чисел $A = 589$ и $B = 195$, их суммы и разности, если $p = 11$.

Р е ш е н и е. Контрольные коды исходных чисел определяем по (8.12). При этом используем второй путь, т. е. нахождение контрольного кода суммированием цифр по модулю:

$$\Sigma a_i = 5 \oplus 8 \oplus 9 \equiv 0(\mathrm{mod}11);\ r'_A \equiv 0(\mathrm{mod}11);$$
$$\Sigma b_i = 1 \oplus 9 \oplus 5 \equiv 4(\mathrm{mod}11);\ r'_B \equiv 0(\mathrm{mod}11).$$

Контрольный код суммы определяем по (8.14) — в этом случае $l = 2$:

$$A + B = 784;\ r'_{A+B} \equiv 0 + 4 - 2(10-1) \equiv 8(\mathrm{mod}11).$$

В случае, когда имеет место отрицательный остаток, к сравнению надо добавить модуль p столько раз, сколько необходимо для получения ближайшего положительного остатка.

Контрольный код разности получим по (8.15) — в этом случае $s = 1$:

$$A - B = 394;\ r'_{A-B} \equiv 0 - 4 + 1(10-1) \equiv 5(\mathrm{mod}11).$$

Ответ $r'_A = 0,\ r'_B = 4,\ r'_{A+B} = 8,\ r'_{A-B} = 5$.

8.7. Выбор модуля для контроля

Достоинства числового метода контроля — в справедливости свойств сравнений для контрольных кодов, что облегчает контроль арифметических операций; достоинства цифрового метода в возможности достаточно просто получать контрольные коды без значительных затрат времени. Чтобы сохранить эти достоинства, необходимо выполнение условия $r_A = r'_A$.

Так как $r_A \equiv A(\mathrm{mod}\ p)$; $r'_A \equiv \sum a_i (\mathrm{mod}\ p)$, имеем $\sum_i a_i q^i \equiv \sum a_i (\mathrm{mod}\ p)$.

Это равенство возможно тогда, когда почленно обе части выражения равны: $a_i q^i \equiv a_i (\mathrm{mod}\ p)$, или $q^i \equiv 1(\mathrm{mod}\ p)$.

Последнее выражение можно получить, если в сравнении $q \equiv 1(\mathrm{mod}\ p)$ возводить обе части в одну и ту же степень. Следовательно, $q \equiv 1(\mathrm{mod}\ p)$, или

$$q = mp + 1, \tag{8.16}$$

где m — целое число.

Из (8.16) следует, что

$$p = (q-1)/m. \tag{8.17}$$

В результате получено, что для сохранения условия $r_A = r'_A$ необходимо наложить ограничения на модуль p.

Анализ (8.17) показывает, что для двоичной системы счисления нет целочисленного решения. Это значит, что контролируемую информацию надо представлять в некоторой промежуточной системе счисления. Выбор промежуточной системы счисления определяется величиной модуля p.

К модулю p предъявляют следующие общие требования:

1) величина модуля p должна быть такой, чтобы возникновение любой арифметической или логической ошибки нарушало сравнимость контрольных кодов;

2) образование контрольного кода должно осуществляться по возможности простыми средствами;

3) величина модуля p должна быть по возможности небольшой, так как необходимость выполнения контрольных операций ведет к увеличению вспомогательного оборудования.

Ввиду того, что цифровая информация в ЭВМ должна представляться символами двоичного алфавита, для контроля целесообразно перейти к системам счисления с основанием $q = 2^s$, где s — некоторое целое положительное число $(s \geq 2)$. Переход от двоичного представления исходной ин-

формации к новому представлению с основанием $q = 2^s$ осуществляется разбиением информации на группы по s разрядов с последующим суммированием этих групп по модулю $p = (2^s - 1)/m$ или при $m = 1$, $p = 2^s - 1$.

В самом деле, если $s = 2$, то исходная информация разбивается на диады, при $s = 3$ — на триады, при $s = 4$ — на тетрады и т. д.

Свертывание — процесс разбиения кодовой комбинации на группы и получения контрольного кода. Как правило, свертки (свернутые коды) образуются в результате суммирования выделенных групп (днад, триад и т. п.) по модулю p. В теории кодирования показано, что модуль можно выбирать из условия

$$p = (2^s \pm 1)/m. \qquad (8.18)$$

Рассмотрим частные случаи образования сверток при наиболее распространенных значениях модуля p.

1. Контроль по модулю 3 ($m = 1, s = 2, p = 3$). Здесь контролируемая информация представляется символами четверичной системы и свертки образуются суммированием диад по модулю 3. Так как $2^2 \equiv 1 (\mathrm{mod}\, 3)$, потребуется двухразрядный двоичный сумматор с цепью циклического переноса из старшего разряда в младший.

Пример 8.6. Найти контрольные коды для чисел $A = 46 = 101110_2$, $B = 29 = 011101_2$, если $p = 3$.

Решение. Контрольные коды для чисел определяем по формуле (8.12) и цифры представляем диадами

$$r_A = 10 \oplus 11 \oplus 10 \equiv 01 (\mathrm{mod}\, 3);$$
$$r_B = 01 \oplus 11 \oplus 01 \equiv 10 (\mathrm{mod}\, 3).$$

Ответ. $r_A = 01$, $r_B = 10$.

2. Контроль по модулю 7 ($m = 1, s = 3, p = 7$). Здесь контролируемая информация разбивается на триады и представляется символами восьмеричной системы. Так как $2^3 \equiv 1 (\mathrm{mod}\, 7)$, для получения свертки нужно иметь трехразрядный двоичный сумматор с цепью циклического переноса.

Пример 8.7. Найти контрольный код для числа $C = 153 = 01111001_2$ при $p = 7$.

Решение. Исходное число разбиваем на триады, которые суммируются по mod 7: $r_i = 011 \oplus 111 \oplus 001 \equiv 100 (\mathrm{mod}\, 7)$.

Ответ. $r_i = 100$.

3. Контроль по модулю 5 ($m=1, s=2, p=5$). Из теории чисел известно, что для того, чтобы число, выраженное в системе с основанием q, делилось на число $q+1$, необходимо и достаточно, чтобы разность между суммой цифр, стоящих на четных и нечетных местах, или наоборот, делилась на величину $q+1$ без остатка.

Из этого правила можно сделать следующий вывод: контрольный код по $\mathrm{mod}(q+1)$ определяется по формуле

$$r_A \equiv \sum_i (-1)^i b_i \bmod(q+1), \tag{8.19}$$

где b_i — двоичное изображение цифр в системе с основанием 2^r.

Так как, по условию, $r_A \le p-1$, то для получения свертки потребуется трехразрядный двоичный сумматор, работающий по модулю 5.

Пример 8.8. Найти контрольный код для числа $A = 0101101110$ при $p=5$

Р е ш е н и е. Сначала исходное число разбивается на диады:

$$\begin{array}{ccccc} A = 01 & 01 & 10 & 11 & 10 \\ b_5 & b_4 & b_3 & b_2 & b_1 \end{array}$$

Затем диады суммируем по правилу (8.19): $r_A = b_1 \oplus b_3 \oplus b_5 \ominus b_2 \ominus b_4 = 10 \oplus 10 \oplus 01 \ominus 01 \ominus 11 = 001 (\mathrm{mod}\, 5)$.

Если получается отрицательный остаток, то его надо заменить на дополнение до модуля.

Ответ. $r_A = 001$.

8.8. Контроль логических операций

К логическим операциям относятся операции сдвига, логического сложения и умножения, выполняемые по правилам, описанным в других главах.

Несмотря на кажущуюся простоту этих правил, осуществление операций контроля сталкивается с рядом трудностей, объясняемых тем, что логические операции являются поразрядными операциями.

О п е р а ц и и с д в и г а. Пусть задано число $A = a_n a_{n-1} \ldots a_1 a_0$, имеющее контрольный код $r_A = a_{k_r} \ldots a_{k_1}$.

Обозначим код числа A, сдвинутый влево, через $\overleftarrow{A}$ (без циклического переноса) и $\overleftarrow{A}_{\text{ц}}$ (с циклическим переносом) (при сдвиге вправо стрелка в обозначении будет повернута направо). Соответствующим образом обозначим и контрольный код: $A \equiv r_A (\mathrm{mod}\, p)$; $\overleftarrow{A} \equiv r_A (\mathrm{mod}\, p)$; $\vec{A} \equiv r_{\overleftarrow{A}} (\mathrm{mod}\, p)$; $\overleftarrow{A}_{\text{ц}} \equiv r_{\overleftarrow{A}_{\text{ц}}} (\mathrm{mod}\, p)$.

Сдвиг влево двоичного числа эквивалентен умножению на два. Так как при сдвиге числа происходит потеря некоторых его разрядов, можно предполагать, что контрольный код сдвинутого числа изменится на величину Δ:

$$r_{\tilde{A}} \equiv \tilde{r}_A + \Delta(\mathrm{mod}\, p), \qquad (8.20)$$

где $\tilde{r}_A = 2r_A$ — сдвинутый влево контрольный код.

Величина Δ зависит от значений a_n и a_{k_s}, которые при сдвиге выходят за пределы разрядной сетки.

Если при сдвиге n-разрядного числа старшая единица выйдет за пределы разрядной сетки, то это эквивалентно вычитанию $a_n\sigma_{n+1}$ единиц из контрольного кода сдвинутого числа [σ_{n+1} — вес $(n+1)$-го разряда].

Если при сдвиге контрольного кода выходит за пределы разрядной сетки разряд $a_{k_s} = 1$, то это эквивалентно уменьшению контрольного кода на 2^s. Такую потерю надо восстановить прибавлением к контрольному коду единицы.

В общем случае (8.20) принимает вид

$$r_{\tilde{A}} \equiv (\tilde{r}_A - a_n\sigma_{n+1} + a_{k_s})\,\mathrm{mod}(2^s - 1). \qquad (8.21)$$

Веса разрядов кодовой комбинации, представленной в системе с основанием 2^s, назначаются следующим образом:

$s = 3$	a_n	a_{n-1}	a_{n-2}	$a_{n-3} \ldots a_3$	a_2	a_1
Вес σ	2^2	2^1	2^0	$2^2 \ldots 2^2$	2^1	2^0

В результате значения поправок Δ для контроля выполнения левого сдвига по модулю будут:

Значение a_n	0	1	0	1
Значение a_{k_s}	0	0	1	1
Поправка Δ	0	−1	+1	0

Значение поправки Δ можно заменить ее дополнением до модуля.

Для выполнения сдвига влево с циклическим переносом из старшего разряда в младший разряд необходимо уменьшить контрольный код на величину $a_n(\sigma_{n+1} - 1)$; так как $\sigma_{n+1} = 1$, этот член равен 0. Следовательно, формула (8.21) примет вид

$$r_{\tilde{A}_u} \equiv \tilde{r}_A + a_{k_s}\,\mathrm{mod}(2^s - 1). \qquad (8.22)$$

Пример 8.9. Найти контрольные коды для числа $A = 1{,}01011010$, сдвигаемого влево, при $p = 7$ $(s = 3)$

Р е ш е н и е . Сначала определяем контрольный код исходного числа путем сложения триад по модулю 7: $r_A \equiv 101 \oplus 011 \oplus 010 \equiv 011 (\mathrm{mod}\, 7)$.

Затем сдвигаем влево число $\vec{A} = 0{,}10110100$ и его контрольный код $\vec{r}_A = 110$.

На основании (8.21) при $a_n = 1$, $a_k = 0$ определяем контрольный код сдвинутого числа: $r_{\vec{A}} \equiv 110 - 1 + 000 \equiv 101 (\mathrm{mod}\, 7)$.

Проводится сдвиг с циклическим переносом: $\vec{A}_{\text{ц}} = 0{,}10110101$, для которого контрольный код $r_{\vec{A}_{\text{ц}}} = 110 + 000 \equiv 110 (\mathrm{mod}\, 7)$

Ответ $r_{\vec{A}} = 101$, $r_{\vec{A}_{\text{ц}}} = 110$

При сдвиге вправо происходит потеря младших разрядов числа и контрольного кода этого числа. Так как сдвиг вправо эквивалентен делению на 2, то

$$\vec{A} = (A - q_1)/2;\ \vec{r}_A = (r_A - a_k)/2 . \qquad (8.23)$$

Эти потери надо компенсировать. Значит, контрольный код сдвинутого вправо числа можно найти по формуле

$$r_{\vec{A}} = \vec{r}_A + \Delta (\mathrm{mod}\, 2^s - 1) . \qquad (8.24)$$

В зависимости от модуля поправка к контрольному коду в случае простого сдвига принимает следующие значения:

Значение a_1	0	0	1	1
Значение a_{k_1}	0	1	0	1
Поправка Δ для:				
$p = 3$	00	10	01	00
$p = 7$	000	100	011	000

При модифицированном сдвиге вправо, который выполняется по правилу $A = 1, a_{n-1}, a_{n-2} \ldots a_2 a_1$; $\vec{A}_{\text{м}} = 1, 1 a_{n-1} \ldots a_3 a_2$, происходит также потеря младших разрядов кодовой комбинации числа и контрольного кода. Для этого случая формула (8.24) сохраняет свой вид, но поправки должны быть следующими:

Значение a_1	0	0	1	1
Значение a_{k_1}	0	1	0	1
Поправка $\Delta_{\text{м}}$ для:				
$p = 3$	10	01	00	10
$p = 7$	100	001	000	100

Пример 8.10. Найти контрольные коды для числа $A = 1{,}01110111101$, сдвигаемого вправо, при $p = 7$

Р е ш е н и е . Сначала определяем контрольный код для исходного числа путем сложения триад по модулю 7: $r_A = 101 \oplus 110 \oplus 111 \oplus 101 \equiv (\mathrm{mod}\,7)$. Затем сдвигаем вправо число $\vec{A} = 0,10111011110$ и его контрольный код $\vec{r}_A = 001$.

На основании (8.24) при $a_1 = 1$; $a_{k_1} = 0$, поправки $\Delta = 011$ определяем контрольный код: $r_{\vec{A}} \equiv 001 + 011 \equiv 100(\mathrm{mod}\,7)$.

Проводим модифицированный сдвиг числа $\vec{A}_м = 1,10111011110$, для которого контрольный код при $\Delta = 000$: $r_{\vec{A}_м} \equiv 001 + 000 \equiv 001(\mathrm{mod}\,7)$.

Ответ $r_A = 010$, $r_{\vec{A}} = 100$, $r_{\vec{A}_м} = 001$.

О п е р а ц и я с л о ж е н и я п о м о д у л ю 2 . Операцию сложения по модулю 2 можно выразить через другие арифметические операции, например: $A \oplus B = A + B - 2(A \wedge B)$.

Если применить к этому выражению уже известные приемы, то получим $A \oplus B = (A + B) + (\overline{A \wedge B})_{сдв}$.

Тогда, используя переход от арифметических выражений к сравнениям, получим следующую формулу для образования контрольного кода:

$$r_{\oplus} \equiv r_{A+B} + \bar{r}_{\wedge} (\mathrm{mod}\, p), \qquad (8.25)$$

где r_{A+B} — контрольный код суммы двух чисел; $\bar{r}_{\wedge}$ — инверсия контрольного кода логического произведения двух чисел со сдвигом влево на один разряд.

Пример 8.11. Найти контрольный код логической суммы чисел $A = 010000111$ и $B = 101110011$ по модулю 7

Р е ш е н и е Прежде всего по изложенным выше правилам определим контрольные коды для исходных чисел: $r_A = 010$, $r_B = 000$, $r_{A+B} = 010$.

Затем вычислим следующие величины: $A \wedge B = 000000011$, $r_{\wedge} = 011$; $A \wedge B_{сдв} = 000000110$, $\bar{r}_{\wedge} = 110$

После этого определим инверсное значение $\bar{r}_{\wedge} = 001$.

По формуле (8.25) находим контрольный код: $r_{\oplus} = 010 + 001 \equiv 011(\mathrm{mod}\,7)$.

Ответ $r_{\oplus} = 011$.

О п е р а ц и я л о г и ч е с к о г о у м н о ж е н и я . Операцию логического умножения двух чисел можно выразить через другие арифметические и логические операции: $A \wedge B = 2^{-1}(A + B) - 2^{-1}(A \oplus B)$.

Умножение на 2^{-1} означает сдвиг кода числа, стоящего в скобках, вправо на один разряд. После перехода к сравнениям контрольный код для логического умножения получается по формуле

$$r_{\wedge} \equiv r_{\overline{A+B}} + \bar{r}_{\oplus} (\mathrm{mod}\, p), \qquad (8.26)$$

где $r_{\overline{A+B}}$ — контрольный код суммы, сдвинутый вправо на один разряд; $\bar{r}_{\oplus}$ — инверсия контрольного кода логической суммы чисел, сдвинутой на разряд вправо.

При выполнении сдвигов необходима коррекция контрольных кодов в соответствии с изложенными выше правилами.

Пример 8.12. Найти контрольный код логического произведения чисел по модулю 3 $A = 10011001$, $r_A = 00$, $B = 0,001111$, $r_B = 01$

Решение Прежде всего найдем сумму чисел и контрольный код $A + B = 11101000$, $r_{A+B} = 01$

Затем по 8.24 вычислим $r_{\overline{A+B}} + \Delta = 10$. Определим: $A \oplus B = 11010110$, $r_{\oplus} = 01$. $r_{\oplus} = 01$, $r_{\bar{\oplus}} = \bar{r}_{\oplus} + \Delta = 10$, $\bar{r}_{\bar{\oplus}} = 01$. Следовательно, контрольный код логического произведения $r_{\wedge} \equiv 10 + 10 \equiv 00 (\mathrm{mod}\, 3)$.

Ответ $r_{\wedge} = 00$.

8.9. Контроль арифметических операций

Арифметические операции выполняют на сумматорах прямого, обратного и дополнительного кодов. Предположим, что изображение чисел (операнды) хранятся в машине в некотором коде, т. е. операция преобразования в заданный код или обратно проводится на выходе или входе машины. Методика реализации операций контроля представляется следующим образом.

Прежде всего рассмотрим изображение числа в соответствующем коде как единую кодовую комбинацию, к которой можно приложить все сформулированные выше правила получения сверток. При этом требуется только обязательная кратность общего числа разрядов избранному модулю.

Рассмотрим последовательность действий на примере сумматора прямого кода.

Так как на сумматоре прямого кода складываются только цифровые части изображений чисел, а знак сохраняется, то контроль можно осуществить двумя способами:

1) раздельный контроль знаковой и цифровой частей изображений результата;

2) обобщенный контроль всего изображения.

При раздельном способе для контроля знаковых разрядов можно использовать средства для обнаружения переполнения, так как в случае модифицированного кода появление ошибок в знаковых разрядах приведет к

несовпадению информации в них. При проверке правильности обработки цифровых частей изображений также не возникает особых трудностей.

При обобщенном способе контроля требуется коррекция контрольного кода результата из-за того, что знак результата при сложении повторяет знак слагаемых. Следовательно, можно констатировать, что контрольный код суммы чисел должен быть

$$r_{(A+B)_{пр}} \equiv r_A + r_B - Sg \cdot \sigma_s (\mathrm{mod}\, p), \qquad (8.27)$$

где Sg — значение знакового разряда операндов; σ_s — вес старшего разряда свертки.

Пример 8.13. Провести контроль операции сложения чисел на сумматоре прямого кода: $[A]_{пр} = 1{,}01101011$, $[B]_{пр} = 1{,}0110010$, $p = 7$.

Решение. Прежде всего определим по (8.12) контрольные коды исходных чисел: $r_A = 110$, $r_B = 101$. Результат операции $[A+B]_{пр} = 1{,}10011101$. Тогда контрольный код результата по (8.12) $r_{A+B} \equiv 110 \oplus 011 \oplus 101 (\mathrm{mod}\, 7)$, или $r_{A+B} \equiv 000$.

На основании (8.27) находим $r_{(A+B)_{пр}} \equiv 110 + 101 - 1 \cdot 100 (\mathrm{mod}\, 7)$, или $r_{(A+B)_{пр}} \equiv 000$.

Контрольные коды совпадают, что свидетельствует о правильном выполнении операции.

Ответ: $r_{A+B} = 000$.

Обобщенный способ контроля может быть применен и для сумматоров обратного и дополнительного кодов.

Пример 8.14. Провести контроль операции сложения кодов для сумматора обратного кода: $[A]_{об} = 1{,}011001001$; $[B]_{об} = 0{,}110001111$, $p = 3$.

Решение. Определяем на основании (8.12):

$$r_{A_{об}} \equiv 10 \oplus 11 \oplus 00 \oplus 10 \oplus 01 (\mathrm{mod}\, 3);\ r_{A_{об}} = 10,$$

$$r_{B_{об}} \equiv 01 \oplus 10 \oplus 00 \oplus 11 \oplus 11 (\mathrm{mod}\, 3);\ r_{B_{об}} = 00.$$

Результат $(A+B)_{об} = 0{,}001011001$ и $r_{(A+B)_{об}} = 10$.

Проверка. $r_{(A+B)_{об}} = r_{A_{об}} + r_{B_{об}} \equiv 10 (\mathrm{mod}\, 3)$.

Ответ: $r_{(A+B)} = 10$.

При сложении чисел на сумматоре дополнительного кода потребуется коррекция контрольного кода в случае, если знаковые разряды изображений содержат единицы, так как при этом возникает единица переноса из знакового разряда. Очевидно, что контрольный код суммы будет равен

$$r_{(A+B)} = r_{A_д} + r_{B_д} - \alpha, \qquad (8.28)$$

где α — коррекция ($\alpha = 1$, если возник перенос из знакового разряда, и $\alpha = 0$ — если переноса нет).

Арифметическим *кодом 1-го вида* будем называть код $A \cdot N$, где A — контролируемое число, N — модуль. Для таких кодов несколько изменяются понятия «расстояния» и «веса».

Весом арифметического кода принято считать количество ненулевых символов в кодовой комбинации, а *расстояние*, определяемое как вес разности кодовых комбинаций, называют *арифметическим расстоянием*.

Если расстояние между двумя числами A_1 и A_2 равно d, то это означает, что переход от одного числа к другому достигается прибавлением величины d. В этом случае все комбинации чисел, находящиеся между A_1 и A_2, являются запрещенными. Следовательно, для обнаружения d-кратной ошибки необходимо иметь расстояние не меньше $d+1$. Если $d=1$, то такой код не сможет обнаруживать ошибок. Величина расстояния для кодов $A \cdot N$-вида зависит от величии A и N.

Предполагается (и это доказано в теории кодирования [10]), что для любого числа A в системе основания $q=2$ существует единственное представление вида

$$A_q = a_n \cdot 2^n + a_{n-1} \cdot 2^{n-1} + \ldots + a_0, \qquad (8.34)$$

где $a_i = \pm 1$ или 0, и в котором нет двух соседних коэффициентов, отличных от нуля.

Представление (8.34) содержит минимальное число ненулевых коэффициентов и называется *каноническим*. В каноническом представлении вес любого числа, начиная с 2^{t+1} и вплоть до числа $2^t + 2^{t-1}$, на единицу больше, чем вес чисел от 1 до 2^{t-1}. Вес чисел, начиная с $2^t + 2^{t-1} + 1$ и вплоть до 2^{t+1}, совпадает с весом чисел $2^t + 2^{t-1} - 1$, $2^t + 2^{t-1} - 2$ и т. д.

Количество разрядов для представления числа $A \cdot N$ равно $\log_2(A \cdot N) = \log_2 A + \log_2 N$, где $\log_2 N$ — избыточность кода. Таким образом, выбор модуля определяет не только избыточность, но и расстояние. В качестве модуля целесообразно выбирать некоторое взаимно простое с основанием системы q число, превосходящее само основание. Можно положить, что для двоичной системы $N=3$, и тогда любой код вида $A \cdot 3$ будет обнаруживать все одиночные ошибки. Следовательно, минимальная избыточность при произвольном основании определяется как $\log_q(q+1)$, т. е. всегда будет требоваться не менее одного, но и не более двух дополнительных разрядов.

Коды с минимальным расстоянием, большим двух, характеризуются величиной $M_q(N, d)$. Величина $M_q(N, d)$ — наименьшее число, дающее

при умножении его на N число, вес которого меньше d в представлении по основанию q. Другими словами, если число N имело вес d в представлении по основанию q, то произведение $NM_q(N, d)$ будет иметь вес меньше d по этому же основанию q.

Если число A изменяется в пределах $0 \le A \le M_q(N, d)$, то при любых N и q минимальное расстояние $A \cdot N$-кода будет равно по меньшей мере d, что вытекает из определения числа $M_q(N, d)$.

В теории кодирования доказано, что

$$M_2(N, 3) = (2^{(N-1)/2} + 1)/N. \qquad (8.35)$$

Основной способ для отыскания значения N_q — способ непосредственных вычислений. Рассмотрим его на конкретном примере.

Пример 8.17. Рассчитать величину M_2 для разных значений модуля N

Р е ш е н и е Возьмем код с расстоянием 3. Тогда возможны случаи:

а) $N = 11$.

1 $N = 11 = 8 + 2 + 1$, $d = 3$;

2 $N = 22 = 16 + 4 + 2$, $d = 3$;

3 $\cdot N = 33 = 32 + 1$, $d = 2$; $M_2(11, 3) = 2$.

б) $N = 13$;

1 $\cdot N = 13 = 8 + 4 + 1$, $d = 3$;

2 $\cdot N = 26 = 16 + 8 + 2$, $d = 3$;

3 $N = 39 = 32 + 8 - 1$ $d = 3$;

5 $N = 65 = 64 + 1$, $d = 2$; $M_2(13, 3) = 5$.

Ответ: а) $M_2(11, 3) = 3$, б) $M_2(13, 3) = 5$.

Кроме $A \cdot N$-кодов существуют *арифметические коды 2-го рода* с большим минимальным расстоянием. Арифметическое расстояние между числами A_1 и A_2 есть вес их разности, а вес, в свою очередь, определяется количеством ненулевых символов в представлении числа по основанию q. Каждое число имеет каноническую форму представления по основанию 2, т. е. минимальное количество равно 1.

Если N простое число, то 2^{N-1} делится на N. Пусть

$$B = (2^{N-1} - 1)/N. \qquad (8.36)$$

Рассмотрим $B \cdot N$-код.

Число знаков в кодовом слове $n = N - 1$. Возьмем некоторое число l $(0 \le l < A)$. Тогда

$$l \cdot B = b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \ldots + b_0, \qquad (8.37)$$

где $b_i \equiv (l \cdot 2^{n-1} \bmod A) \bmod 2$.

И действительно: $N = 11$, $B = (1024 - 1)/11 = 93$. Если $l = 10$, то

$$l \cdot B = 930 = 1 \cdot 2^9 + 1 \cdot 2^8 + 1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0.$$

Проверим коэффициенты b_i по формуле (8.37):

$$b_0 \equiv (10 \cdot 2^{10} \bmod 11) \bmod 2 \equiv 0;$$
$$b_1 \equiv (10 \cdot 2^9 \bmod 11) \bmod 2 \equiv 1;$$
$$b_2 \equiv (10 \cdot 2^8 \bmod 11) \bmod 2 \equiv 0;$$
$$b_3 \equiv (10 \cdot 2^7 \bmod 11) \bmod 2 \equiv 0;$$
$$b_4 \equiv (640 \bmod 11) \bmod 2 \equiv 0;$$
$$b_5 \equiv (320 \bmod 11) \bmod 2 \equiv 1;$$
$$b_6 \equiv (160 \bmod 11) \bmod 2 \equiv 0;$$
$$b_7 \equiv (80 \bmod 11) \bmod 2 \equiv 1;$$
$$b_8 \equiv (40 \bmod 11) \bmod 2 \equiv 1;$$
$$b_9 \equiv (20 \bmod 11) \bmod 2 \equiv 1.$$

Итак, десятичное число 10 в двоичном представлении имеет вид 1010, в $A \cdot N$-коде будет иметь вид 1101110, а в $B \cdot N$-коде при $N = 11$, $B = 93$ число 10 будет представлено в виде 1110100010.

При использовании $B \cdot N$-кода коэффициенты b_i можно вычислять сразу же, не раскладывая числа в многочлен $b_i \equiv (l \cdot 2^{n-i} \bmod A) \bmod 2$; $n = N - 1$ известно заранее, так как $B = (2^{N-1} - 1)/N$ вычисляется через N.

Таким образом, при изображении числа l в $B \cdot N$-коде половина знаков равна единице, а половина — нулю. Это следует из выражения

$$b_i \equiv (l \cdot 2^{n-i} \bmod N) \bmod 2,$$

так как 2 и N взаимно простые числа, значит, остатки от деления чисел 2^i на N — целые числа от 1 до $N - 1$. Так как число N — нечетное, количество остатков — всегда четное число, а значит половина из остатков числа нечетные, а половина — четные числа. Следовательно, половина остатков по $\bmod 2$ дает единицу, а половина — нуль.

Пример 8.18. Пусть $N = 11$, $B = 93$. Найти изображения чисел l от 1 до 10 в $B \cdot N$-коде.

Решение. Количество символов для изображения чисел в $93 \cdot N$-коде равно $10(N-1)$:

$l = 1$, $B \cdot 1 = 93 = 1011101$,
$l = 2$, $B \cdot 2 = 186 = 10111010$,
$l = 3$, $B \cdot 3 = 279 = 100010111$,
$l = 4$, $B \cdot 4 = 372 = 101110100$,
$l = 5$, $B\ 5 = 465 = 111010001$,
$l = 6$, $B\ 6 = 558 = 1000101110$. (а)
$l = 7$, $B\ 7 = 651 = 1010001011$,
$l = 8$, $B\ 8 = 744 = 1011101000$,
$l = 9$, $B\ 9 = 837 = 1101000101$,
$l = 10$, $B \cdot 10 = 930 = 1110100010$.

Ответ см. формулу (а) данного примера.

Арифметический вес числа в $B \cdot N$-коде равен $(N-1)/3$, или $(N+1)/3$ в зависимости от того, какое из этих чисел целое $(N > 3)$.

В случае $B \cdot N$-кодов по известному N можно не только определить количество символов и их вид, но и заранее определять арифметическое расстояние, что непосредственно влияет на корректирующие способности кода.

Существуют еще *самодополняющиеся* $(A \cdot N + B)$ коды. В этом случае код для дополнительного числа N — дополнительный код самого числа A.

Если числа, которые кодируются, имеют основание b, то дополнение для числа A будет $(b-1-A)$. Дополнение же числа $A \cdot N$ при представлении его по основанию q равно $q^n - 1 - A \cdot N$:

$$\underbrace{q^n - 1 - (A \cdot N + B)}_{\text{дополнение кодового числа}} = \underbrace{N(b-1-A) + B}_{\text{код дополнительного числа}},$$

$$B = [q^n - 1 - N(b-1)]/2.$$

Код возможен только при целых B.

Расстояния для $(A \cdot N + B)$ кода те же, что и для $A \cdot N$-кода.

Пример 8.19. Пусть нужно найти $(A \cdot N + B)$-двоичный код для кодирования десятичных знаков, исправляющий одиночные ошибки.

Решение. Возьмем $b = 10$. Чтобы исправлять одиночные ошибки, нужно иметь код с кодовым расстоянием $d = 3$.

Так как если $0 \le A < M_2(N, d)$, то при любых N и q расстояние $A \cdot N$-кода равно по меньшей мере d

По таблице ищем $M_2(N,3) > 10$ (так как A — десятичный знак, т. е. от 1 до 9) $M_2(N,3) = 27$, при этом $N = 19$. Наибольшее число в этом случае равно $19 \cdot 9 = 171$, для него потребуется по крайней мере 8 двоичных знаков (11010101).

Из формулы $B = [q'' - 1 - N(b-1)]/2$ получим: $B = (2^8 - 1 - 19 \cdot 9)/2 = 42$.

$A \cdot N + B = 9 \cdot 19 + 42 = 213 < 2^8$ (нужно 8 знаков двоичных, значит, код возможен).

Окончательно таблица кодирования имеет такой вид:

Таблица 8.6

A	$A \cdot N + B$	Код
0	42	0010 1010
1	61	0011 1101
2	80	0101 0000
3	99	0110 0011
4	118	0111 0110
5	137	1000 1001
6	156	1001 1100
7	175	1010 1111
8	194	1100 0010
9	213	1101 0101

Ответ двоичные коды представлены в таблице 8.6.

Рассмотрим несколько случаев применения описанных выше кодов.

1. Пусть $A_1 + A_2 < 10$, т. е. нет переноса в кодовых словах. Тогда $N \cdot A_1 + B + N \cdot A_2 + B = N(A_1 + A_2) + 2B$. Отсюда видно, что нужна поправка «—B».

Пример 8.20. Сложить числа A_1 и A_2 в коде $A \cdot N + B$, если $N = 19$, $B = 42$ ($A_1 = 3$, $A_2 = 4$).

Решение.

$A_1 = 3 =$ 0110 0011
$A_2 = 4 =$ +0111 0110
1101 1001 — запрещенная комбинация
−0010 1010
$A_1 + A_2 = 7 =$ 1010 1111 — разрешенная комбинация, соответствует числу 7

Пусть произошла одиночная ошибка в 5-м разряде.

0110 0011
+0111 0110
1100 1001 — запрещенная комбинация
−0010 1010
1001 1111 — запрещенная комбинация наиболее близка к 1010 1111
+ ↑
1010 1111

Ответ $A_1 + A_2 = 10101111$.

2. Пусть $A_1 + A_2 \geq 10$. Тогда

$$N \cdot A_1 + B + N \cdot A_2 + B = N(A_1 + A_2) + 2B = \underbrace{10N}_{\text{уйдет в перенос}} + k \cdot N + 2B.$$

Следовательно, при переносе в старшее кодовое слово нужно добавить поправку $+(A-1)$. В этом слове, откуда происходит перенос, нужно сделать поправку $-(N-1+B)$.

Пример 8.21. Сложить числа $A_1 = 328$ и $A_2 = 563$, $N = 19$, $B = 42$.

Здесь $N - 1 = 18(10010_2)$.

Решение

$$\begin{array}{rl} A_1 = [328] = & 01100011\ \ 01010000\ \ 11000010 \\ A_2 = [563] = & +\ 10001001\ \ 10011100\ \ 01100011 \\ \hline A_1 + A_2 = [891] = & 11101100\ \ 11101101\ \ 00100101 \\ & -\ 00101010\ +\ 00010010\ +\ 00010010 \\ \hline & 11000010\ \ 11111111\ \ 00010011 \\ & \underbrace{}_{8}\ -\ 00101010\ +\ 00101010 \\ \hline & \underbrace{11010101}_{9}\ \ \underbrace{00111101}_{1} \end{array}$$

Пусть при сложении произошли ошибки (они указаны знаком –×–):

$$\begin{array}{lll} 01100011 & 01010000 & 11000010 \\ 10001001 & 10011100 & 01100011 \\ \hline 10101100 & 11101100 & 10100101 \\ -\ 00101010 & -\ 00101010 & -\ 00010010 \\ \hline 10000010 & 11000010 & 10010011 \\ +\ & +\ 00010010 & +\ 00101010 \\ \hline 10000010 & 11010100 & 10111101 \quad \text{результат} \\ 11000010 & 11010101 & 00111101 \quad \text{образец} \end{array}$$

Ответ $A_1 + A_2 = 1100001011010101 00111101$.

Сравнив результат с образцом разрешенной комбинации, можно определить местоположение ошибки.

В современных вычислительных машинах используется контроль на четность и нечетность. Этот контроль позволяет лишь зафиксировать наличие ошибки, но не позволяет ее исправить. Но он нашел широкое применение благодаря своей простоте, так как и наличие информационной избыточности требует применения контрольной аппаратуры, которая может оказаться слишком объемной, труднореализуемой. Например, в случае самодополняющихся $(A \cdot N + B)$-кодов необходимо умножить число на N, где N — простое число, прибавить B; для представления четырехзначных чисел по величине используют восемь разрядов (см. пример 8.21), помимо этого

9. СПОСОБЫ ЗАЩИТЫ ИНФОРМАЦИИ

9.1. Особенности систем защиты информации

В связи с широким распространением персональных компьютеров не только как средств обработки информации, но также как оперативных средств коммуникации (электронная, телефаксная почта), возникают проблемы, связанные с обеспечением защиты информации от преднамеренных или случайных искажений.

Актуальность этих проблем подчеркивается также тем обстоятельством, что персональный компьютер или автоматизированное рабочее место (АРМ) является частью систем обработки информации, систем коллективного пользования, вычислительных сетей. В таких случаях предъявляются достаточно жесткие требования по надежности и достоверности передаваемой информации.

Любой канал связи характеризуется наличием в нем помех, приводящих к искажению информации, поступающей на обработку. С целью уменьшения вероятности ошибок принимается ряд мер, направленных на улучшение технических характеристик каналов, на использование различных видов модуляции, на расширение пропускной способности и т. п. При этом также должны приниматься меры по защите информации от ошибок или несанкционированного доступа.

Доступ — это получение возможности использовать информацию, хранящуюся в ЭВМ (системе).

Всякая информация в машине или системе требует той или иной *защиты*, под которой понимается совокупность методов, позволяющих управлять доступом выполняемых в системе программ к хранящейся в ней информации.

Существуют методы физической защиты каналов связи, каналов передач информации, помещений, где обрабатывается информация (например, экранирование). Однако физические методы не являются предметом рассмотрения в этой главе.

Задача защиты информации в информационных вычислительных системах решается, как правило, достаточно просто: обеспечиваются средства

контроля за выполнением программ, имеющих доступ к хранимой в системе информации. Для этих целей используются либо списки абонентов, которым разрешен доступ, либо *пароли*, что обеспечивает защиту информации при малом количестве пользователей. Однако при широком распространении вычислительных и информационных систем, особенно в таких сферах, как обслуживание населения, банковское дело, этих средств оказалось явно недостаточно. Система, обеспечивающая защиту информации, не должна позволять доступа к данным пользователям, не имеющим такого права. Такая система защиты является неотъемлемой частью любой системы коллективного пользования средствами вычислительной техники, независимо от того, где они используются. Данные экспериментальных исследований различных систем коллективного пользования показали, что пользователь в состоянии написать программы, дающие ему доступ к любой информации, находящейся в системе. Как правило, это обусловлено наличием каких-то ошибок в программных средствах, что порождает неизвестные пути обхода установленных преград.

В процессе разработки систем защиты информации выработались некоторые общие правила, которые были сформулированы Ж. Солцером и М. Шредером (США):

1. *Простота механизма защиты.* Так как средства защиты усложняют и без того уже сложные программные и аппаратные средства, обеспечивающие обработку данных в ЭВМ, естественно стремление упростить эти дополнительные средства. Чем лучше совпадает представление пользователя о системе защиты с ее фактическими возможностями, тем меньше ошибок возникает в процессе работы.

2. *Разрешения должны преобладать над запретами.* Нормальным режимом работы считается отсутствие доступа, а механизм защиты должен быть основан на условиях, при которых доступ разрешается. Допуск дается лишь тем пользователям, которым он необходим.

3. *Проверка полномочий любого обращения к любому объекту информации.* Это означает, что защита выносится на общесистемный уровень и предполагает абсолютно надежное определение источника любого обращения.

4. *Разделение полномочий* заключается в определении для любой программы и любого пользователя в системе минимального круга полномочий. Это позволяет уменьшить ущерб от сбоев и случайных нарушений и сократить вероятность непреднамеренного или ошибочного применения полномочий.

5. *Трудоемкость проникновения в систему.* Фактор трудоемкости зависит от количества проб, которые нужно сделать для успешного проникно-

вения. Метод прямого перебора вариантов может дать результат, если для анализа используется сама ЭВМ.

6. *Регистрация проникновений в систему.* Иногда считают, что выгоднее регистрировать случаи проникновения, чем строить сложные системы защиты.

9.2. Криптографические методы защиты информации

Обеспечение защиты информации от несанкционированного доступа — дело сложное, требующее широкого проведения теоретических и экспериментальных исследований по вопросам системного проектирования. Наряду с применением разных приоритетных режимов и систем разграничения доступа разработчики информационных систем уделяют внимание различным криптографическим методам обработки информации.

Криптографические методы можно разбить на два класса:

1) обработка информации путем замены и перемешивания букв, при котором объем данных не меняется (шифрование);

2) сжатие информации с помощью замены отдельных сочетаний букв, слов или фраз (кодирование).

По способу реализации криптографические методы возможны в аппаратном и программном исполнении.

Для защиты текстовой информации при передачах на удаленные станции телекоммуникационной сети используются аппаратные способы шифрования и кодирования. Для обмена информацией между ЭВМ по телекоммуникационной сети, а также для работы с локальными абонентами возможны как аппаратные, так и программные способы. Для хранения информации на магнитных носителях применяются программные способы шифрования и кодирования.

В простейшем случае для построения правил шифрования текстовой информации используется некоторый смешанный алфавит, например перестановка обычного алфавита. В примере 9.1 показан исходный алфавит, смешанный алфавит и шифрование короткого сообщения, в котором каждая буква заменяется соответствующей буквой смешанного алфавита.

Пример 9.1. Зашифруйте с помощью представленного кода фразу «Попробуйте прочитать шифрованный текст».

Алфавит	АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ
Код	ЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬЬЮ
Шифруемая информация	ПОПРОБУЙТЕ ПРОЧИТАТЬ ШИФРОВАННЫЙ ТЕКСТ

Ответ.

Код	АВАПВЦЛЗОН АПВЯЩОЙОТ ЧЩДВУЙЫЬИЗ ОНХРО

Сообщения, зашифрованные с помощью простой подстановки, расшифровываются следующим образом. Определяется частота появления каждой буквы в шифрованном сообщении и сравнивается с частотами букв алфавита. Таблица частот букв русского алфавита (табл. 9.1) приведена ниже

Таблица 9.1

Буква	Частота	Буква	Частота
О	0,0940	Б	0,0197
А	0,0896	З	0,0193
Е	0,0856	У	0,0179
И	0,0739	Г	0,0153
Н	0,0662	Ь	0,0125
Т	0,0611	Ч	0,0118
Р	0,0561	Й	0,0094
С	0,0554	Х	0,0093
Н	0,0421	Ц	0,0087
М	0,0417	Ж	0,0064
В	0,0400	Ю	0,0063
Л	0,0358	Щ	0,0048
К	0,0322	Ф	0,0034
Д	0,0280	Э	0,0033
Я	0,0243	Ш	0,0032
Ы	0,0225	Ъ	0,0002

Для сообщений длиной 40 символов и более по распределениям частот можно определить буквы исходного текста.

Более надежное шифрование текстов обеспечивается с помощью ключевых слов и нескольких алфавитов шифрования. Ниже (табл. 9.2) показан квадрат Виженера, построенный на основе смешаниого алфавита, приведенного в примере 9.1. Каждая строка квадрата образуется из предыдущей с помощью циклического сдвига на одну позицию. Таким образом, квадрат состоит из 32 смешанных алфавитов, каждому из них соответствуют буквы исходного алфавита. Эти буквы записываются в квадрате слева перед каждым смешанным алфавитом.

Пример 9.2. Зашифруйте с помощью представленного кода следующую фразу:

Информация ПОПРОБУЙТЕ ПРОЧИТАТЬ ШИФРОВАННЫЙ ТЕКСТ

Ключевое слово КЛЮЧКЛЮЧКЛ ЮЧКЛЮЧКЛЮ ЧКЛЮЧКЛЮЧКЛ ЮЧКЛЮ

Ответ ОСЫШЧФРЙТП ЫШЧУГЗХЪМ АОЮВНФЬЬЕНД ПТАТП

или

ОСЫШ ЧФРЙ ТПЫШ ЧУГЗ ХЪМА ОЮВН ФЬЬЕ НДПТ ДТП

Таблица 92

	АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ
А	ЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮ
Б	ЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙ
В	УКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦ
Г	КЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУ
Д	ЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУК
Е	НГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕ
Ж	ГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕН
З	ШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГ
И	ЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШ
Й	ЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩ
К	ХЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗ
Л	ЪФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХ
М	ФЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪ
Н	ЫВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФ
О	ВАПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФЫ
П	АПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФЫВ
Р	ПРОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФЫВА
С	РОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФЫВАП
Т	ОЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФЫВАПР
У	ЛДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФЫВАПРО
Ф	ДЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛ
Х	ЖЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛД
Ц	ЭЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖ
Ч	ЯЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭ
Ш	ЧСМИТЬБЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯ
Щ	СМИТЬБЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧ
Ъ	МИТЬБЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧС
Ы	ИТЬБЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМ
Ь	ТЬБЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИ
Э	ЬБЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТ
Ю	БЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬ
Я	ЮЙЦУКЕНГШЩЗХЪФЫВАПРОЛДЖЭЯЧСМИТЬБ

На примере 9.2 показано шифрование фразы при помощи ключевого слова КЛЮЧ и данного квадрата. Ключевое слово многократно записывается под исходным текстом, и буквы исходного текста шифруются при помощи соответствующих смешанных алфавитов. Этот метод шифрования уже не поддается раскрытию с помощью простого подсчета частот букв.

Если использовать несколько ключевых слов различной длины, то можно еще более повысить защищеиность информации, поскольку разным сообщениям будут соответствовать разные ключевые слова.

Основным отличием двоичной иифорлации при передаче данных от информации любых других видов (текстовой, графической) является ее полная формализованность, т. е. отсутствие каких-либо логических связей между отдельными элементами передаваемой иифорлации. Это приводит к тому, что ошибка в таком сообщении не может быть обиаружена по смысловому содержанию принятой информации.

Основным методом повышения достоверности является помехозащищенное кодирование в сочетании с автоматическим запросом и повторением искаженных кодовых комбинаций. Кодироваиие заключается во введении определенным образом в информационную последовательность дополнительных проверочных разрядов таким образом, что между отдельными разрядами устанавливается определенная математическая связь. Тогда ошибка, возникающая при прохождении сообщения по каналу связи, вызовет нарушение этой закономерности, что будет обнаружено при декодировании.

Наиболее широкое применение получили в настоящее время *циклические* коды, которые обладают высокими корректирующими свойствами и относительно просто реализуются технически. Циклические коды обнаруживают как независимые, так и групповые ошибки*. Обнаруживающие свойства циклических кодов определяются видом порождающего многочлена. Циклический код, образованный многочленом степени «K» , обнаруживает все пакеты ошибок длины $n \le K$. Под пакетом ошибок длиной «n» подразумевается любой ряд ошибок, для которого число знаков между первой и последней ошибками, включая эти ошибки, равно «n».

Как показывает опыт, ошибки во всех реальных каналах связи не являются независимыми. Они обычно возникают группами. Закон распределения ошибок в пакетах и самих пакетов очень сложен. Он определяется огромным количеством факторов, учесть которые не просто. В целях упрощения математического аппарата используем аппроксимацию действительного закона распределения.

Предположим, что пакет ошибок и ошибки в пределах пакета возникают независимо друг от друга. Вие пакетов ошибки отсутствуют.

Оценка канала проводится по трем параметрам:

— вероятности возникновения пакета любой длины — P_n;

* Более подробно о циклических кодах см. в книге *У. Питерсона, Э. Уэлдона* «Коды, исправляющие ошибки»

— вероятности ошибки в пределах пакета — P_E;

— распределению вероятности $P(l)$ возникновения пакета ошибок от длины пакета l.

Для телефонных каналов, работающих по кабельным линиям, вероятность возникновения ошибок внутри пакета ошибок составляет $P_E = 0{,}6–0{,}4$. При этом предполагается, что ошибки внутри пакета независимы и подчиняются биномиальному закону:

$$P(t) = C_n^t P_E^t (1 - P_E)^{n-t}. \qquad (9.1)$$

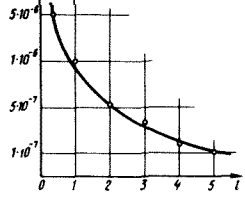


Рис. 9.1. График изменения вероятности ошибок

Зависимость вероятности возникновения пакета ошибок от его длины ориентировочно определяется соотношением

$$P(l) = P_1(1 - P_1)^{l-1}, \qquad (9.2)$$

где $P_1 = (P_n / P_{ср}) P_E$ — относительная вероятность возникновения пакета ошибок длиной в один элемент.

Вероятность возникновения пакета ошибок длиной l_1 составляет

$$P_{l_1} = P_n P_1 (1 - P_1)^{l_1 - 1}. \qquad (9.3)$$

Вероятность возникновения пакета ошибок длиной свыше l_1 определяется следующим образом:

$$P_{l > l_1} = P_n (1 - P_1)^{l}. \qquad (9.4)$$

Согласно приведенным формулам для телефонных каналов различных достоверности и типов рассчитаны и приводятся в литературе зависимости вероятности возникновения пакета ошибок от его длины. Эта зависимость ориентировочно имеет вид, представленный на рис. 9.1.

Пример 9.3. Рассчитать для указанной на рис. 9.1 зависимости длину пакетов ошибок, суммарная вероятность возникновения которых меньше заданной.

Решение. Пусть информация разделена на 30-разрядные слова, т. е. длина пакета $l_{max} \le 30$. Закон распределения кратности ошибок ориентировочно можно признать нормальным, тогда средняя кратность ошибок $K \approx 10$. Требуемая достоверность канала равна $P_K = 10^{-6}$. Вероятность возникновения пакетов ошибок длиной l_1:

$$P_{l>l_1} = (10^{-6}/10) = 10^{-7}.$$

Графически на кривой вероятности (см. рис. 9.1) определяем, что $l_1 = 5$.

Ответ: длина пакета $l_1 = 5$.

9.3. Аппаратные средства защиты

Аппаратные способы шифрования информации применяются для передачи защищенных данных по телекоммуникационной сети. Для реализации шифрования с помощью смешанного алфавита используется перестановка отдельных разрядов в пределах одного или нескольких символов.

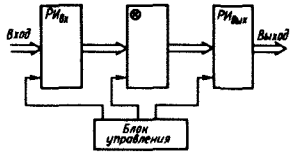


Рис. 9.2. Схема шифратора с перестановкой разрядов: $РИ_{вх}$ — входной регистр; $РИ_{вых}$ — выходной регистр; ⊗ — блок перестановки

На рис. 9.2 показана схема аппаратного шифратора, использующего операцию перестановки разрядов в пределе одного байта информации. Для расшифровки сообщений применяется симметричная перестановка. Блок перестановки может быть сменным или управляемым. Блок управления синхронизирует работу шифровального устройства. Возможное число перестановок для n-разрядных символов равно $(n!-1)$. Если перестановка разрядов выполняется в пределах нескольких байт, то такая операция называется *запутыванием*.

Для шифрования с помощью ключевых слов применяется операция сложения по модулю 2 (исключающее ИЛИ). Схема шифрования показана на рис. 9.3.

Ключевое слово хранится в 64-разрядном регистре ключа РК. Информация, подлежащая шифрованию, записывается в 64-разрядный регистр информации $РИ_{вх}$. После заполнения этого регистра выполняется операция сложения по модулю 2 с содержимым регистра ключа. Результат представляет собой зашифрованную информацию, которая поступает в выходной 64-разрядный регистр $РИ_{вых}$.

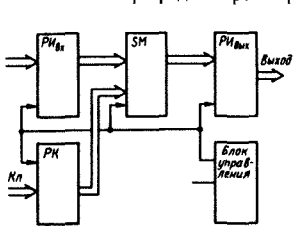


Рис. 9.3. Схема ключевого шифратора: РК — регистр ключей; *SM* — сумматор

На практике используются несколько чередующихся операций запутывания и сложения по модулю 2 с различными ключами. Пример такой схемы приведен на рис. 9.4. Блок управления на рисунке не показан.

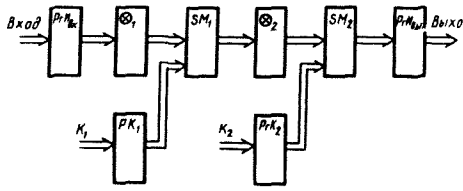


Рис. 9.4. Схема шифратора с двойным ключом

При определении количества операций запутывания и сложения по модулю 2 приходится искать компромиссное решение между сложностью шифрования и временем преобразования.

Ключевые слова для работы схем шифрования выбираются с помощью специальных генераторов случайных чисел и передаются в приемное устройство в зашифрованном виде по предыдущим ключам. Дешифрование информации выполняется в обратной последовательности.

9.4. Программные средства защиты

Программные способы применяются для шифрования информации, хранящейся на магнитных носителях (дисках, лентах). Это могут быть данные различных информационно-справочных систем АСУ, АСОД и др. Программные способы шифрования сводятся к операциям перестановки, перекодирования и сложения по модулю 2 с ключевыми словами. При этом используются команды ассемблера TR (перекодировать) и XC (исключающее ИЛИ).

Кодирование информации. Особое место в программах обработки информации занимают операции кодирования. Преобразование информации, в результате которого обеспечивается изменение объема памяти, занимаемой данными, называется *кодированием*. На практике кодирование всегда используется для уменьшения объема памяти, так как экономия памяти ЭВМ имеет большое значение в информационных системах. Кроме того, кодирование можно рассматривать как криптографический метод обработки информации.

Естественные языки обладают большой избыточностью. Не составит большого труда исправить все ошибки, которые есть в следующей фразе: «Есл нсклко бкв удлть, эт прдлжн ещ м б прчтно». Для экономии памяти, объем которой ограничен, имеет смысл ликвидировать избыточность текста или уплотнить текст.

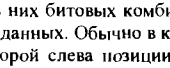
Существуют несколько способов уплотнения текста.

1. Переход от естественных обозначений к более компактным. Этот способ применяется для сжатия записи дат, номеров изделий, уличных адресов и т. п. Идея способа показана на примере сжатия записи даты. Обычно мы записываем дату в виде 22.09.83, что требует 6 байтов памяти ЭВМ. Однако ясно, что для представления дня достаточно 5 битов, месяца — 4, года — не более 7, т. е. вся дата может быть записана в 16 битах или в 2 байтах.

Другой распространенный способ представления дат был предложен Жозефом Скалигером в 1582 г. для астрономических целей. По этому способу дата записывается как общее число дней, прошедшее к данному дню, считая с 1 января 4713 г. до н. э. По этой схеме 1 января 1983 г. записывается как 2 444 323. Обычно для реальных расчетов ограничиваются четырьмя последними цифрами этого представления. 24 мая 1967 г. записывается в виде 0000, и отсчет дней от этой даты требует 16 битов в упакованном десятичном формате.

2. Подавление повторяющихся символов. В различных информационных текстах часто встречаются цепочки повторяющихся символов, например пробелы или нули в числовых полях. Если имеется группа повторяющихся символов длиной более 3, то ее длину можно сократить до трех символов. Сжатая таким образом группа повторяющихся символов представляет собой триграф $S \cdot P \cdot N$, в котором S — символ повторения; P — признак повторения; N — количество символов повторения, закодированных в триграфе.

В других схемах подавления повторяющихся символов используют особенность кодов ДКОИ, КОИ-7, КОИ-8, заключающуюся в том, что большинство допустимых в них битовых комбинаций не используется для представления символьных данных. Обычно в коде ДКОИ не используются комбинации с нулем во второй слева позиции. Этот бит может являться признаком повторяемости последующего или предыдущего символа:



3. Кодирование часто используемых элементов данных. Этот способ уплотнения данных также основан на употреблении неиспользуемых комбинаций кода ДКОИ. Для кодирования, например, имен людей можно использовать комбинации из двух байтов триграф $P \cdot N$, где P — признак кодирования имени, N — номер имени. Таким образом может быть закодировано 256 имен людей, чего обычно бывает достаточно в

информационных системах. Если в байте N старший байт использовать в качестве признака пола, то этим байтом можно закодировать 128 мужских и 128 женских имен.

Другой способ основан на отыскании в тексте наиболее часто встречающихся сочетаний букв и даже слов и замене их на неиспользуемые байты кода ДКОИ.

4. Посимвольное кодирование. Семибитовые и восьмибитовые коды не обеспечивают достаточно компактного кодирования символьной информации. Более пригодными для этой цели являются 5-битовые коды, например международный телеграфный код МТК-2. Перевод информации в код МТК-2 возможен с помощью программного перекодирования или с использованием специальных элементов на основе больших интегральных схем (БИС). Пропускная способность каналов связи при передаче алфавитно-цифровой информации в коде МТК-2 повышается по сравнению с использованием 8-битовых кодов почти на 40 %.

5. Коды переменной длины. Коды с переменным числом битов на символ позволяют добиться еще более плотной упаковки данных. Метод заключается в том, что часто используемые символы кодируются короткими кодами, а символы с низкой частотой использования — длинными кодами. Идея такого кодирования была впервые высказана Хаффманом, и соответствующий код называется кодом Хаффмана. Использование кодов Хаффмана позволяет достичь сокращения исходного текста почти на 80 %.

Использование различных методов уплотнения текстов кроме своего основного назначения — уменьшения информационной избыточности — обеспечивает определенную криптографическую обработку информации. Однако наибольшего эффекта можно достичь при совместном использовании как методов шифрования, так и методов кодирования информации.

9.5. Надежность средств защиты информации

Надежность защиты информации может быть оценена временем, которое требуется на расшифрование (разгадывание) информации и определение ключей.

Если информация зашифрована с помощью простой подстановки, то расшифровать ее можно было бы, определив частоты появления каждой буквы в шифрованном тексте и сравнив их с частотами букв русского алфавита. Таким образом определяется подстановочный алфавит и расшифровывается текст.

Рассмотрим вариант, когда информация шифруется с помощью ключевого слова. Суть такого шифрования заключается в следующем. Ключевое

слово последовательно складывается по модулю 2 с фрагментами текста длиной, равной длине ключевого слова. Полученную в результате такого преобразования информацию расшифровать уже не так просто, так как здесь требуется разгадать само ключевое слово.

Правильной отправной точкой в этом случае будет нахождение длины ключевого слова. Обозначим длину ключевого слова буквой «*k*». Отметим также, что, когда период равен длине ключевого слова, символы текста обрабатываются одной буквой ключевого слова. Таким образом, весь шифрованный текст можно разбить на k групп $G_1, G_2, \ldots, G_k$. Каждая группа начинается с позиции $i\,(1 \le i \le k)$ и содержит каждую k-ю букву текста, начиная с i-й буквы. Каждая из этих групп обрабатывается одним символом ключевого слова, что может быть представлено шифрованием при помощи одного алфавита.

Процесс шифрования с помощью сложения по модулю 2 с ключевым словом может быть выражен следующей формулой:

$$y_{j,\,k+i} = x_{j,\,k+i} \oplus w_i \text{ при } 0 \le j < N/k,$$

где $y_{j,\,k+i}$ — буква шифрованного текста из группы G_i; $x_{j,\,k+i}$ — буква исходного текста из группы G_i; w_i — i-й символ ключевого слова; j — номер наложения ключевого слова; N — длина текста.

Подсчитаем количество появлений одинаковых букв $y_{j,\,k+i}$ в группе G_i и, разделив это число на общее количество букв в этой группе (N/k), определим частоты букв в группе. Сравнивая полученные частоты с частотами букв русского алфавита, определим предположительные соответствия букв в группе G_i. Теперь, используя свойства сложения по модулю 2, можно определить возможные значения i-го символа в ключе:

$$V_{ij} = y_{j,\,k+i} \oplus x_{j,\,k+i} \text{ при } 0 \le j < N/k.$$

Подсчитаем максимальное количество одинаковых символов V_{ij} при $0 \le j < N/k$ и обозначим это число через α_i.

Если провести указанные выше вычисления для всех $i\,(1 \le i \le k)$, то можно определить процент совпадения:

$$P_k = (100/N)\sum_{i=1}^{i=k} \alpha_i.$$

Таким образом, процесс расшифрования текста может выглядеть следующим образом. Значения предполагаемой длины ключевого слова (k)

изменяются от 1 до некоторой величины L (возможна длина логической записи). Интервал изменений равен 1. Для каждого k ($1 \le k \le L$) определяется процент совпадения P_k. Если возникает резкое увеличение значения P_k, то текущее значение предполагаемого ключа и обрабатываемого текста выдается пользователю на экран дисплея для визуального анализа и возможной корректировки редко используемых символов.

Дальнейшие изменения величины «k» должны привести к появлению периодических скачков процента совпадения. Длина периода таких скачков равна длине ключевого слова.

Естественно, нет необходимости проводить указанные вычисления до значения $k = L$, если уже при меньших значениях k получены удовлетворительные результаты и длина ключевого слова найдена. В большинстве случаев бывает достаточно обнаружения первого скачка процента совпадения.

Приведем временные оценки обработки шифрованного текста. Обозначим через t_k время на вычисление процента совпадения P_k. Тогда для полной автоматической обработки текста потребуется время

$$T_k = kt_k .$$

Если при шифровании был добавлен циклический сдвиг (шифрование по схеме «ключ—сдвиг»), то время обработки текста и получения наилучшего процента совпадения возрастет:

$$T_{k-c_{max}} = 8kt_k .$$

Если шифрование выполнялось с помощью двойного ключа и двойного циклического сдвига, т. е. по схеме «ключ—сдвиг—ключ—сдвиг», то процесс расшифровывания текста значительно усложняется, так как в этом случае нельзя подсчитать частоты появления символов в тексте. Но и здесь правильным началом будет поиск длины ключевых слов. Алгоритм такого поиска может быть следующим.

Найти два места в шифрованном тексте, где две одинаковые буквы идут в том же порядке. Такое повторение могло произойти по двум причинам. Возможно, различные сочетания букв и соответствующие разные части ключевого слова случайно выразились в одинаковые сочетания букв или в исходном тексте были повторения, которые попали на одинаковые части ключевого слова, и, таким образом, оказались зашифрованными дважды одним и тем же способом. Во втором случае расстояние между началами повторяющихся сочетаний букв должно быть кратно длине ключевого слова. Определить, по какой из двух причин произошло повторение данного сочетания букв, невозможно. Но если в шифрованном тексте повторяются

сочетания из трех букв или более, то вероятность повторения ключа в этом случае очень велика. Таким образом, исследуя зашифрованный текст на повторяющиеся сочетания, можно с большой вероятностью определить длину ключа или кратную ей величину.

Дальнейшая обработка сводится к подбору ключевого слова, предварительному расшифровыванию текста и передаче его в обработку по одинарной схеме «ключ—сдвиг». Это следует повторять в цикле до тех пор, пока текст не будет расшифрован.

При обработке зашифрованной информации значительная роль в анализе и принятии окончательного решения отводится оператору терминала, выполняющего визуальную обработку вариантов, предлагаемых машиной.

Если предположить, что для шифрования использовались ключевые слова с буквами русского алфавита, а длина ключа составляла 8 символов, то время обработки текста, зашифрованного по схеме «ключ—сдвиг—ключ—сдвиг», составит

$$T_{k-c-k-c_{max}} = kt_k \cdot 2^{46} \text{ с}.$$

Пусть $k = 8, t_k = 0{,}1$ (при самом удачном алгоритме определения процента совпадения ЭВМ с быстродействием 1 млн опер/с и длиной обрабатываемого шифрованного текста 1000 символов). Тогда чистое процессорное время обработки составит

$$T_{k-c} = 6{,}4 \text{ с},$$

$$T_{k-c-k-c_{max}} = 1{,}56 \cdot 10^{10} \text{ ч},$$

$$T_{k-c\ k-c-k-c_{max}} = 1{,}38 \cdot 10^{23} \text{ ч},$$

$$T_{k-c-k-c-k-c-k-c_{max}} = 1{,}21 \cdot 10^{36} \text{ ч}.$$

Следует отметить также, что все вычисления выполнялись при условии, что алгоритмы шифрования известны. Неизвестными были лишь ключевые слова и количества разрядов циклических сдвигов.

10. Методы логического проектирования

10.1. Числовое и геометрическое представление фуикций алгебры логики

Часто для упрощения записи функций алгебры логики вместо полного перечисления термов используют номера наборов, для которых функция принимает единичное значение. Например, функция, заданная таблицей 2.14, может быть записана в виде $f(x_1, x_2, x_3) = \vee_1 F(3, 4, 6)$; это означает, что функция принимает значение 1 на наборах, номера которых равны 3, 4, 6. Такую форму записи называют *числовой*.

Многие преобразования, выполняемые над булевыми функциями, удобно интерпретируются с использованием их геометрических представлений. Так, фуикцию двух переменных можно интерпретировать как некоторую плоскость, заданную в системе координат x_1, x_2 (рис. 10.1). Отложим по каждой оси единичные отрезки x_1 и x_2. Получится квадрат, вершины которого соответствуют комбинациям переменных. Из такого геометрического представления функции двух переменных следует: две вершины, принадлежащие одному и тому же ребру и называемые соседними, «склеиваются» по переменной, меняющейся вдоль этого ребра.

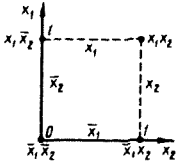


Рис. 10.1. Геометрическое представление функции двух переменных

Таким образом, **правила склеивания** для минтермов можно записать для функции трех переменных в следующем виде:

$$x_1\bar{x}_2\bar{x}_3 + x_1\bar{x}_2x_3 = x_1\bar{x}_2 .$$

Пример 10.11. Определить соседние минтермы и применить правило склеивания:

$$f_1(x_1, x_2, x_3) = x_1x_2x_3,\ f_4(x_1, x_2, x_3) = \bar{x}_1x_2\bar{x}_3 ;$$

$$f_2(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2\bar{x}_3;\ f_5(x_1, x_2, x_3) = x_1\bar{x}_2x_3 ;$$

$$f_3(x_1, x_2, x_3) = x_1x_2\bar{x}_3 .$$

Решение. В соответствии с определением выпишем пары соседних термов: f_1 и f_5; f_2 и f_4; f_1 и f_6; f_3 и f_4.

Применим правило склеивания к этим парам, получим новые термы.

Ответ

$$x_1x_2x_3 + x_1x_2\bar{x}_3 = x_1x_2;$$

$$\bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 = \bar{x}_1x_2;$$

$$x_1x_2x_3 + x_1\bar{x}_2x_3 = x_1x_3;$$

$$x_1x_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 = x_1\bar{x}_3.$$

Для функций трех переменных геометрическое представление выполняют в виде куба, вершины которого обозначены десятичными цифрами, двоичными цифрами и произвольными переменными x_i на рис. 10.2, *а*, *б* и *в* соответственно. Ребра куба поглощают вершины. Грани куба поглощают свои ребра и, следовательно, вершины.

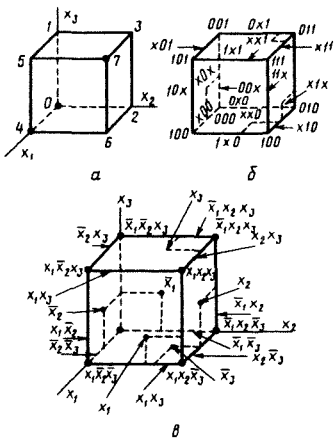


Рис. 10.2. Геометрическое представление функции трех переменных

Функция четырех переменных представляется уже в виде четырехмерного куба (рис. 10.3). В геометрическом смысле каждый набор $x_1x_2\ldots x_n$ может рассматриваться как n-мерный вектор, определяющий точку n-мерного пространства. Исходя из этого все множество наборов, на которых определена функция n переменных, представляется в виде вершин n-мерного куба. Координаты вершин куба должны быть указаны в порядке, соответствующем порядку перечисления переменных в записи функций. Отметив точками вершины, в которых функция принимает значение, равное единице, получим геометрическое представление ФАЛ.

Терм максимального ранга принято называть 0-кубом (точкой) и обозначать K^0.

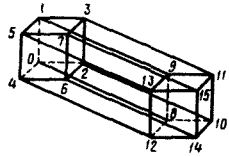


Рис. 10.3. Геометрическое представление функции четырех переменных

Например, для $f(x_1, x_2, x_3) = \vee(0, 4, 7)$

$$K^0 = \begin{Bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{Bmatrix}.$$

Если два 0-куба из комплекса K^0 различаются только по одной координате, то они образуют 1-куб (отрезок): $K^1 = \{1 \quad 0 \quad 0\}$, где x – независимая координата.

Если два 1-куба имеют общую независимую компоненту и различаются только по одной координате, то они образуют 2-куб.

Таким образом, для построения одномерного единичного куба берут два 0-куба (точки) и соединяют отрезком прямой. Двумерный куб (грань) получается, если вершины двух 1-кубов соединить параллельными отрезками. Трехмерный куб получается при соединении соответствующих вершин двух двумерных кубов отрезками единичной длины. Геометрическое представление будет использовано при разработке методов минимизации с использованием минимизирующих карт.

10.2. Минимизация логических функций. Метод неопределенных коэффициентов для базиса И—ИЛИ—НЕ

На основании теоремы Жегалкина любую логическую функцию можно представить в нормальной форме. Например, пусть функция $f(x_1, x_2, x_3)$ записана в виде следующей нормальной дизъюнктивной формы (НДФ):

$$\begin{aligned} f(x_1, x_2, x_3) = {} & k_1^1 x_1 + k_1^0 \bar{x}_1 + k_2^1 x_2 + k_2^0 \bar{x}_2 + k_3^1 x_3 + k_3^0 \bar{x}_3 + k_{12}^{11} x_1 x_2 + \\ & + k_{12}^{10} x_1 \bar{x}_2 + k_{12}^{01} \bar{x}_1 x_2 + k_{12}^{00} \bar{x}_1 \bar{x}_2 + k_{13}^{11} x_1 x_3 + k_{13}^{10} x_1 \bar{x}_3 + k_{13}^{01} \bar{x}_1 x_2 + k_{13}^{00} \bar{x}_1 \bar{x}_3 + \\ & + k_{23}^{11} x_2 x_3 + k_{23}^{10} x_2 \bar{x}_3 + k_{23}^{01} \bar{x}_2 x_3 + k_2^{00} \bar{x}_2 \bar{x}_3 + k_{123}^{111} x_1 x_2 x_3 + k_{123}^{110} x_1 x_2 \bar{x}_3 + \\ & + k_{123}^{101} x_1 \bar{x}_2 x_3 + k_{123}^{100} x_1 x_2 \bar{x}_3 + k_{123}^{100} x_1 \bar{x}_2 \bar{x}_3 + k_{123}^{011} \bar{x}_1 x_2 x_3 + k_{123}^{010} \bar{x}_1 x_2 \bar{x}_3 + \\ & + k_{123}^{001} \bar{x}_1 \bar{x}_2 x_3 + k_{123}^{000} \bar{x}_1 \bar{x}_2 \bar{x}_3, \end{aligned} \tag{10.1}$$

где k_{nk}^{lmn} — неопределенные коэффициенты, принимающие значение 0 или 1 и подбираемые так, чтобы получающаяся после этого НДФ была минимальной.

Критерий минимальности — минимальное количество букв в записи НДФ. При определении НДФ пользуются следующими свойствами: $x_1 + x_2 + \ldots + x_n = 0$, если $x_1 = x_2 = \ldots = x_n = 0$ и $x_1 + x_2 + \ldots + x_n = 1$, если хотя бы один член уравнения равен единице:

$$\begin{aligned}
&k_1^0 + k_2^0 + k_3^0 + k_{12}^0 + k_{13}^0 + k_{23}^{00} + k_{123}^{000} = f_0(0,0,0);\\
&k_1^0 + k_2^0 + k_3^1 + k_{12}^{00} + k_{13}^{01} + k_{23}^{01} + k_{123}^{001} = f_1(0,0,1);\\
&k_1^0 + k_2^1 + k_3^0 + k_{12}^{01} + k_{13}^{00} + k_{23}^{00} + k_{123}^{010} = f_2(0,1,0);\\
&k_1^0 + k_2^1 + k_3^1 + k_{12}^{01} + k_{13}^{01} + k_{23}^{01} + k_{123}^{011} = f_3(0,1,1);\\
&k_1^1 + k_2^0 + k_3^0 + k_{12}^{10} + k_{13}^{10} + k_{23}^{10} + k_{123}^{000} = f_4(1,0,0);\\
&k_1^1 + k_2^0 + k_3^1 + k_{12}^{10} + k_{13}^{11} + k_{23}^{11} + k_{123}^{101} = f_5(1,0,1);\\
&k_1^1 + k_2^1 + k_3^0 + k_{12}^{11} + k_{13}^{10} + k_{23}^{10} + k_{123}^{110} = f_6(1,1,0);\\
&k_1^1 + k_2^1 + k_3^1 + k_{12}^{11} + k_{13}^{11} + k_{23}^{11} + k_{123}^{111} = f_7(1,1,1).
\end{aligned} \qquad (10.2)$$

Если $f_i = 0$ на соответствующем наборе переменных, то все коэффициенты, входящие в данное урввнение, равны нулю. Тогда в остальных уравнениях системы (10.2) надо вычеркнуть члеиы, содержащие нулевые коэффициенты, а из оставшихся уравнений, равных единице, найти коэффициенты, определяющие конъюнкцию наименьшего ранга в каждом из уравнений.

На основании изложенного можно сформулировать следующий алгоритм нахождения неопределенных коэффициентов:

1. Выбрать очередную строку, в которой $f_i = 0$. Все коэффициенты этой строки приравнять нулю.

2. Если все нулевые строки просмотрены, то перейти к п. 3, если нет, то к п. 1.

3. Просмотреть строки, в которых $f_i = 1$, и вычеркнуть из них все коэффициенты, встречающиеся в строках, где $f_i = 0$.

4. Переписать все модифицированные уравнения.

5. Выбрать очередную строку $f_i = 1$ и вычеркнуть максимально возможное количество коэффициентов так, чтобы ранг остающихся членов был минимальным.

Метод неопределенных коэффициеитов наиболее применим для дизъюнктивной формы и практически непригоден для конъюнктивной формы.

Пример 10.12. Найти минимальную форму для функции

$$f(x_1, x_2, x_3) = \vee(0, 2, 4, 7) = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 + x_1\bar{x}_2\bar{x}_3 + x_1x_2x_3 .$$

Решение Составим систему уравнений (10.2), запишем ее в виде таблицы 10.1

Таблица 10.1

k_1^0	k_2^0	k_3^0	k_{12}^{00}	k_{13}^{00}	k_{23}^{00}	k_{123}^{000}	f_0	1
k_1^0	k_2^0	k_3^1	k_{12}^{00}	k_{13}^{01}	k_{23}^{01}	k_{123}^{001}	f_1	0
k_1^0	k_2^1	k_3^0	k_{12}^{01}	k_{13}^{00}	k_{23}^{10}	k_{123}^{010}	f_2	0
k_1^0	k_2^1	k_3^1	k_{12}^{01}	k_{13}^{01}	k_{23}^{11}	k_{123}^{011}	f_3	1
k_1^1	k_2^0	k_3^0	k_{12}^{10}	k_{13}^{10}	k_{23}^{00}	k_{123}^{100}	f_4	0
k_1^1	k_2^0	k_3^1	k_{12}^{10}	k_{13}^{11}	k_{23}^{01}	k_{123}^{101}	f_5	1
k_1^1	k_2^1	k_3^0	k_{12}^{11}	k_{13}^{10}	k_{23}^{10}	k_{123}^{110}	f_6	0
k_1^1	k_2^1	k_3^1	k_{12}^{11}	k_{13}^{11}	k_{23}^{11}	k_{123}^{111}	f_7	1

После вычеркивания нулевых коэффициентов уравнения (10.2) принимают такой вид:

$$k_{123}^{111} = 1;$$
$$k_{23}^{00} + k_{123}^{100} = 1;$$
$$k_{13}^{00} + k_{123}^{010} = 1;$$
$$k_{13}^{00} + k_{23}^{00} + k_{123}^{000} = 1.$$

Результат: $k_{13}^{00} = 1$; $k_{23}^{00} = 1$; $k_{123}^{111} = 1$.

Ответ: $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_3 + \bar{x}_2\bar{x}_3 + x_1x_2x_3$.

10.3. Метод Квайна

При минимизации по методу Квайна (базис 1) предполагается, что исходная функция задана в СНДФ.

Импликанта функции — некоторая логическая функция, обращаемая в нуль при наборе переменных, на котором сама функция также равна нулю.

Поэтому любой конъюнктивный терм, входящий в состав СНДФ, или группа термов, соединенных знаками дизъюнкции, являются импликантами исходной НДФ.

Первичная импликанта функции — импликанта типа элементарной конъюнкции некоторых переменных, никакая часть которой уже не является импликантой.

Задача минимизации по методу Квайна состоит в попарном сравнении всех импликант, входящих в СНДФ, с целью выявления возможности поглощения какой-то переменной:

$$Fx_i \vee F\bar{x}_i = F. \qquad (10.3)$$

Таким образом, удается снизить ранг термов. Эта процедура проводится до тех пор, пока не останется ни одного члена, допускающего поглощение с каким-либо другим термом. Термы, подвергшиеся поглощению, отмечаются. Неотмеченные термы представляют собой первичные импликанты.

Полученное логическое выражение не всегда оказывается минимальным. Поэтому исследуется возможность дальнейшего упрощения. Для этого составляется таблица, в строках которой записываются найденные первичные импликанты, а в столбцах указываются термы исходного уравнения. Клетки этой таблицы отмечаются в случае, если первичная импликанта входит в состав какого-либо терма. После этого задача упрощения сводится к тому, чтобы найти такое минимальное количество первичных импликант, которые покрывают все столбцы.

Метод Квайна выполняется в несколько этапов, рассмотрим его применение на конкретном примере.

Пусть необходимо минимизировать логическую функцию, заданную в виде

$$f(x_1, x_2, x_3, x_4) = \vee_1(3, 4, 5, 7, 9, 11, 12, 13) = \bar{x}_1\bar{x}_2x_3x_4 \vee \bar{x}_1x_2\bar{x}_3\bar{x}_4 \vee$$

$$\vee \bar{x}_1x_2\bar{x}_1x_4 \vee \bar{x}_1x_2x_1x_4 \vee x_1\bar{x}_2\bar{x}_1x_4 \vee x_1\bar{x}_2x_3x_4 \vee x_1x_2\bar{x}_3\bar{x}_4 \vee x_1x_2\bar{x}_1x_4.$$

Задача решается в несколько этапов.

Этап 1. Нахождение первичных импликант. Прежде всего составляется таблица (табл. 10.2) и находятся импликанты четвертого и третьего ранга, т. е. снижается ранг членов, входящих в СНДФ.

Затем составляется другая таблица (табл. 10.3), которая включает все термы, не подвергшиеся поглощению, а также первичные импликанты третьего ранга. Составление таблиц продолжается до тех пор, пока нельзя будет применить правило (10.3). В рассматриваемом примере можно дойти до первичной импликанты второго ранга (табл. 10.3) — $x_2\bar{x}_3$.

Первичные импликанты наименьшего ранга выделены в таблице 10.3.

Этап 2. Расстановка меток. Составляется таблица, число строк которой равно числу полученных первичных импликант, а число столбцов совпадает с числом минтермов СНДФ. Если в некоторый минтерм СНДФ входит какая-либо из первичных импликант, то на пересечении соответствующего столбца и строки ставится метка (табл. 10.4).

Таблица 10.2

Исходные термы	1 0011	2 0100	3 0101	4 0111	5 1001	6 1011	7 1100	8 1101
$x_1x_2x_3x_4$ (0011)	1			$\bar{x}_1x_3x_4$		$x_2x_3x_4$		
$x_1x_2x_3x_4$ (0100)		1	$x_1x_2x_3$				$x_2x_3x_4$	
$x_1x_2\bar{x}_3x_4$ (0101)		$x_1x_2x_3$	1	$x_1x_2x_4$				$x_2x_3x_4$
$\bar{x}_1x_2x_3x_4$ (0111)	$x_1x_3x_4$		$\bar{x}_1x_2x_4$	1				
$x_1\bar{x}_2\bar{x}_3x_4$ (1001)					1	$x_1x_2x_4$		$x_1x_3x_4$
$x_1\bar{x}_2x_3x_4$ (1011)	$\bar{x}_2x_3x_4$				$x_1\bar{x}_2x_4$	1		
$x_1x_2\bar{x}_3\bar{x}_4$ (1100)		$x_2\bar{x}_3\bar{x}_4$					1	$x_1x_2\bar{x}_3$
$x_1x_2\bar{x}_3x_4$ (1101)			$x_2\bar{x}_3x_4$		$x_1\bar{x}_3x_4$		$x_1x_2x_3$	1

Этап 3. Нахождение существенных импликант. Если в каком-либо из столбцов таблицы 10.4 имеется только одна метка, то первичная импликанта в соответствующей строке является существенной, так как без нее не будет получено все множество заданных минтермов. В таблице 10.4 существенной

Таблица 10.3

Первичные импликанты ранга 3	$\bar{x}_1x_3x_4$	$\bar{x}_2x_3x_4$	$\bar{x}_1x_2\bar{x}_3$	$x_2\bar{x}_3\bar{x}_4$	$\bar{x}_1x_2x_4$	$x_2\bar{x}_3x_4$	$x_1\bar{x}_2x_4$	$x_1\bar{x}_3x_4$	$x_1x_2\bar{x}_3$
$\bar{x}_1x_3x_4$	1								
$\bar{x}_2x_3x_4$		1							
$\bar{x}_1x_2\bar{x}_3$			1						$x_1\bar{x}_3$
$x_2\bar{x}_3\bar{x}_4$				1		x_2x_3			
$\bar{x}_1x_2x_4$					1				
$x_2\bar{x}_3x_4$				$x_2\bar{x}_3$		1			
$x_1\bar{x}_2x_4$							1		
$x_1\bar{x}_3x_4$								1	
$x_1x_2\bar{x}_3$			$x_2\bar{x}_3$						1

импликантой является терм $x_2\bar{x}_3$. Столбцы, соответствующие существенным импликантам, из таблицы вычеркиваются.

Таблица 10.4

Первичные импликанты	1	2	3	4	5	6	7	8
	Исходные термы							
	0011	0100	0101	0111	1001	1011	1100	1101
$\bar{x}_1x_3x_4$	V			V				
$\bar{x}_2x_3x_4$	V					V		
$\bar{x}_1x_2x_4$			V	V				
$x_1\bar{x}_2x_4$					V	V		
$x_1\bar{x}_3x_4$					V		V	V
$x_2\bar{x}_3$		V	V				V	V

Этап 4. Вычеркивание лишних столбцов. После третьего этапа в результате вычеркивания столбцов 2, 3, 7 и 8 получается таблица 10.5. Если в таблице есть два столбца, в которых имеются метки в одинаковых строках, то один из них вычеркивается. Покрытие оставшегося столбца будет осуществлять отброшенный минтерм. В примере такого случая нет.

Этап 5. Вычеркивание лишних первичных импликант. Если после отбрасывания некоторых столбцов на этапе 4 в таблице 10.5 появляются строки, в которых нет ни одной метки, то первичные импликанты, соответствующие этим строкам, исключаются из дальнейшего рассмотрения, так как они не покрывают оставшиеся в рассмотрении минтермы.

Таблица 10.5

Первичные импликанты	1	4	5	6
	Исходные термы			
	0011	0111	1001	1011
$\bar{x}_1x_3x_4$	V	V		
$\bar{x}_2x_3x_4$	V			V
$\bar{x}_1x_2x_4$		V		
$x_1\bar{x}_2x_4$			V	V
$x_1\bar{x}_3x_4$				

Этап 6. Выбор минимального покрытия. Выбирается в таблице 10.5 такая совокупность первичных импликант, которая включает метки ао всех

столбцах по крайней мере по одной метке в каждом столбце. При нескольких возможных вариантах такого выбора отдается предпочтение варианту покрытия с минимальным суммарным числом букв в импликантах, образующих покрытие. Этому требованию удовлетворяют первичные импликанты $\bar{x}_1 x_3 x_4$ и $x_1 \bar{x}_2 x_4$.

Таким образом, минимальная форма заданной функции складывается из суммы существенных импликант (этап 3) и первичных импликант, покрывающих оставшиеся минтермы (этап 6):

$$f(x_1, x_2, x_3, x_4) = x_1 \bar{x}_3 \vee \bar{x}_1 x_3 x_4 \vee x_1 \bar{x}_2 x_4 .$$

10.4. Метод Квайна—Мак-Класки

Недостаток метода Квайна — необходимость полного попарного сравнения всех минтермов на этапе нахождения первичных импликант. С ростом числа минтермов увеличивается количество попарных сравнений. Числовое представление функций алгебры логики позволяет упростить этап 1 (см. § 10.3). Все минтермы записываются в виде их двоичных номеров, а все номера разбиваются по числу единиц на непересекающиеся группы, так как условие образования r-куба — наличие расхождения в $(r-1)$-кубах только по одной координате в одном двоичном разряде и наличие общих независимых координат. Поэтому группы, которые различаются в двух разрядах или более, просто не имеет смысла сравнивать. При этом в i-группу войдут все номера наборов, имеющие в своей двоичной записи i единиц. Попарное сравнение можно проводить только между соседними по номеру группами.

Пример 10.3. Пусть задана функция

$$f(x_1, x_2, x_3, x_4) = \vee(3, 4, 5, 7, 9, 11, 12, 13) .$$

Р е ш е н и е . Сначала выпишем 0-кубы:

$$K^0 = 0011, 0100, 0101, 0111, 1001, 1011, 1100, 1101 .$$

Разобьем 0-кубы на три группы по количеству единиц в каждом двоичном наборе

$$K_1^0 = \{0100\}; K_2^0 = \begin{Bmatrix} 0011 \\ 0101 \\ 1001 \\ 1100 \end{Bmatrix}; K_3^0 = \begin{Bmatrix} 0111 \\ 1011 \\ 1101 \end{Bmatrix}.$$

Э т а п 1. Нахождение первичных импликант:

а) сравнение K_1^0 и K_2^0:

0100*	0011
	0101*
	1001
	1100*

Здесь и ниже символом «*» отмечены наборы, которые склеиваются.

На основании сравнения строим куб K_1^1, в котором поглощеиная координата заменяется символом X.

$$K_1^1 = \begin{Bmatrix} 010X \\ X100 \end{Bmatrix};$$

б) сравнение K_2^0 и K_3^0:

0011*	0111*
0101*	1011*
1001*	1101*
1100*	

Строим куб K_2^1.

$$K_2^1 = \begin{Bmatrix} 0X11 \\ X011 \\ 01X1 \\ 10X1 \\ 1X01 \\ 110X \\ X101 \end{Bmatrix}.$$

Первичной импликанты ранга 4 нет;

в) разобьем все 1-кубы на четыре группы в зависимости от положения независимой координаты X

$$K_1^2 = \begin{Bmatrix} 010X \\ 110X \end{Bmatrix}; K_1^3 = \begin{Bmatrix} 01X1 \\ 10X1 \end{Bmatrix}; K_1^4 = \begin{Bmatrix} 0X11 \\ 1X01 \end{Bmatrix}; K_1^5 = \begin{Bmatrix} X100 \\ X011 \\ X101 \end{Bmatrix};$$

г) сравнение K_1^2 и K_1^3, K_1^4 и K_1^5 внутри каждой группы:

010X	01X1*	0X11*	X100
110X	10X1*	1X01*	X011*
			X101

На основании сравнения строим кубы $K_1^{2'} = X10X$; $K_1^{2''} = X10X$; K_1^3 и K_1^4 не сравниваются

Следовательно, символом «•» отмечены первичные импликанты ранга 3:

$$K^1 = \{01X1;\ 10X1;\ 0X11;\ 1X01;\ X011\};$$

д) сравнение $K_1^{2'}$ и K_1^{2IV}

$$K_1^2 = K_1^{2IV}.$$

Следовательно, получаем первичную импликанту ранга 2:

$$K^2 = \{X10X\}.$$

Этап 2. Расстановка меток (табл. 10.6).

Таблица 10.6

Первичные импликанты	Исходные термы							
	0011	0100	0101	0111	1001	1011	1100	1101
01X1			*	*				
10X1					*	*		
0X11	*			*				
1X01					*		*	*
X011	*					*		
X10X		*	*				*	

Этап 3. Нахождение существенных импликант.

Существенной импликантой ранга 2 будет терм

$$\{X10X\} = x_2\bar{x}_3.$$

Этапы 4 и 5. Отсутствуют.

Этап 6. Выбирается минимальное покрытие оставшихся термов {10X1} и {0X11} (табл. 10.7).

Таблица 10.7

Первичные импликанты	Исходные термы			
	0011	0111	1001	1011
01X1		*		
10X1			*	*
0X11	*	*		
1X01			*	
X011	*			*

Ответ $f(x_1, x_2, x_3, x_4) = x_2\bar{x}_3 \vee x_1\bar{x}_2x_4 \vee \bar{x}_1x_3x_4$.

При использовании метода Квана для СКНФ необходимо рассматривать значение фуикции $f = 0$ и термы, соответствующие этим значениям.

В результате получим $f = \vee_1(x_1, x_2, \ldots, x_n)$. Далее необходимо воспользоваться соотношениями де Моргана, с тем чтобы привести функцию к СНДФ. Все дальнейшие действия аналогичны вышеизложенным.

10.5. Метод минимизирующих карт

Один из способов графического представления булевых функций от небольшого числа переменных — карты Карно. Их разновидность — карты Вейча, которые строят как развертки кубов на плоскости. При этом вершины куба представляются клетками карты, координаты которых совпадают с координатами соответствующих вершин куба. Карта заполняется так же, как таблица истинности: значение 1 указывается в клетке, соответствующей набору, на котором функция имеет значение единицы. Значение 0 обычно на картах не отражается. На рис. 10.4 показаны примеры условного размещения переменных на минимизирующих картах для двух (рис. 10.4, *а*), трех (рис. 10.4, *б*) и четырех (рис. 10.4, *в*) переменных. Примеры использования карт Карно для минимизации указанных ниже трех функций даны на рис. 10.5, *а*, *в* соответственно:

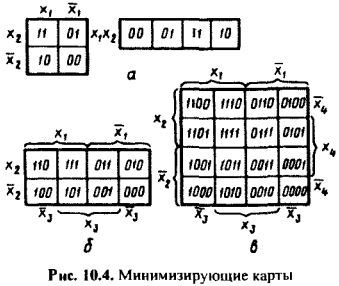


Рис. 10.4. Минимизирующие карты

$$f_1 = x_1\bar{x}_2 \vee x_1x_2 = x_1 \qquad \text{(рис. 10.5, } a\text{)};$$

$$f_2 = \bar{x}_1\bar{x}_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1x_2x_3 \qquad \text{(рис. 10.5, } б\text{)};$$

$$f_3 = \vee_1(3, 4, 5, 7, 9, 11, 12, 13) = x_2\bar{x}_3 \vee \bar{x}_1x_3x_4 \vee x_1\bar{x}_2x_4 \qquad \text{(рис. 10.5, } в\text{)}.$$

Карты Карно обычно используют для ручной минимизации булевых функций при небольшом числе переменных (не более 5).

Правила минимизации следующие.

1. Две соседние клетки (два 0-куба) образуют один 1-куб. При этом клетки, лежащие на границах карты, также являются соседними по отношению друг к другу (пример образования 1-куба см. на рис. 10.5, *а*). Независимая координата обозначена символом X.

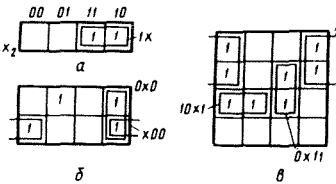


Рис. 10.5. Минимизация логических функций с помощью карт

2. Четыре вершины могут объединяться, образуя 2-куб, содержащий две независимые координаты (пример образования 2-куба изображен на рис. 10.5, *в*).

3. Восемь вершин могут объединяться, образуя один 3-куб.

4. Шестнадцать вершин, объединяясь, образуют один 4-куб и т. д.

При числе переменных, равном или большем пяти, отобразить графически функцию в виде единой плоской карты невозможно. В таких случаях строят комбинированную карту, состоящую из совокупности более простых карт, например четырехмерных. Процедура минимизации в этом случае состоит в том, что сначала находят минимальные формы внутри четырехмерных кубов, а затем, расширяя понятие соседних клеток, отыскивают минимальные термы для совокупности карт. Соседними клетками являются клетки, совпадающие при совмещении карт поворотом вокруг общего ребра.

При получении минимальной формы для СКНФ функция задается термами, *принимающими нулевое значение на соответствующих наборах*. Поэтому в клетках минимизирующей карты пишут нули.

Рис. 10.6. Пример минимизации для конъюнктивной формы

Пример 10.4. Найти минимальную конъюнктивную форму для функции $f(x_1, x_2, x_3, x_4) = \bigvee_0(0, 1, 2, 5, 6, 8, 9, 10, 11, 12, 14)$ методом минимизирующих карт (рис. 10.6).

Решение. С помощью минимизирующих карт находим

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1\bar{x}_3x_4 \vee x_3\bar{x}_4 \vee x_1\bar{x}_3\bar{x}_4 \vee \bar{x}_2\bar{x}_3.$$

Применяя правила де Моргана, получаем:

$$f = \overline{\bar{x}_1\bar{x}_3x_4 \vee x_3\bar{x}_4 \vee x_1\bar{x}_3\bar{x}_4 \vee \bar{x}_2\bar{x}_3} = \overline{\bar{x}_1\bar{x}_3x_4} \wedge$$
$$\wedge \overline{x_3\bar{x}_4} \wedge \overline{x_1\bar{x}_3\bar{x}_4} \wedge \overline{\bar{x}_2\bar{x}_3} = (x_1 \vee x_3 \vee \bar{x}_4) \wedge$$
$$\wedge (\bar{x}_3 \vee x_4) \wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3).$$

Ответ: $f(x_1, x_2, x_3, x_4) = (x_1 \vee x_3 \vee x_4) \wedge (\bar{x}_3 \vee x_4) \wedge$
$\wedge (\bar{x}_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3)$.

10.6. Минимизация логических фуикций в базисе ⊕, ∧, ¬

Метод неопределенных коэффициентов может быть применен для минимизации функций, заданных в разных базисах.

Рассмотрим применение метода неопределенных коэффициентов на примере базиса ⊕, ∧, ¬. Функцию $f(x_1, x_2, x_3)$ представим в виде, аналогичном нормальной дизъюнктивной форме, где вместо дизъюнкции стоит знак операции сложения ⊕ по модулю 2. Эта операция имеет особенности, отличающие ее от операций дизъюнкции:

$$0 = 0 \oplus 0 \oplus 0 \oplus \ldots \oplus 0, \tag{1}$$

или

$$\underbrace{0 = 1 \oplus 1 \oplus 1 \oplus \ldots \oplus 1}_{m}, \tag{2}$$

где $m = 2k$ — четное количество единиц:

$$\underbrace{1 = 1 \oplus 1 \oplus 1 \oplus \ldots \oplus 1}_{m}, \tag{3}$$

где $m = 2k + 1$ — нечетное количество единиц.

Для операции дизъюнкции всегда $1 = 1 \vee 1 \vee 1 \ldots$. Наличие свойств (2) и (3) операции сложения по модулю 2 усложняет минимизацию. Так как основными критериями минимизации по-прежнему являются минимальный ранг каждого терма и минимальное количество термов, при минимизации в базисе ⊕, ∧, ¬ целесообразно приравнять к нулю все коэффициенты на наборах, где $f = 0$, так как тогда в единичных строках могут остаться термы высокого ранга. Поэтому особой разницы между выбором очередной строки нулевой или единичной нет. Количество коэффициентов, остающихся в нулевых строках, должно быть четным, а в единичных — нечетным. Лучше начать с единичных строк и оставлять те коэффициенты минимального ранга, которые чаще повторяются в этих строках.

Пример 10.5. Задана функция $f(x_1, x_2, x_3) = \oplus(0, 1, 5, 6)$.

Найти минимальное представление в базисе ⊕, ∧, ¬.

Решение. Составим таблицу 10.8.

В таблице коэффициент k_2^0 повторяется три раза и его целесообразно оставить. В нулевой строке надо оставить еще какой-нибудь коэффициент минимального ранга, который также повторяется в тех единичных строках, в которых еще не было оставлено ни одного коэффициента. Для получения минимальной формы выполним следующие действия.

1. Посчитаем, сколько раз встречаются в единичных строках термы первого минимального ранга, и оставим те из них, которые встречаются максимальное число раз.

Б. Нахождение первичных импликант ранга 2.

Разобьем все множество импликант на три группы в зависимости от положения независимой координаты X:

$$K_1'' = \begin{Bmatrix} 10X \\ 01X \\ 11X \end{Bmatrix};\ K_2'' = \begin{Bmatrix} 0X0 \\ 1X0 \\ 1X1 \end{Bmatrix};\ K_3'' = \begin{Bmatrix} X00 \\ X10 \\ X11 \end{Bmatrix};$$

а) сравнение K_1'', K_2'' и K_3''.

$$K_1^2 = \{1XX, X1X\};$$
$$K_2^2 = \{XX0, 1XX\};$$
$$K_3^2 = \{XX0, X1X\}$$

Таким образом, $K^2 = \{1XX, X1X, XX0\}$.

Этап 3. Расстановка меток, выбор покрытия (табл. 10.9).

Таблица 10.9

Первичные импликанты	Минтермы						
	A				*B*		
	000	010	100	111	101	011	110
000	v						
010		v					
100			v				
111				v			
101					v		
011						v	
110							v
X00	v		v				
1X0			v				
X10		v					
0X0	v	v					
11X				v			v
01X		v				v	
10X			v		v		
XX0	v	v	v				v
1XX							
X1X		v		v		v	v

На основании таблицы 10.9 получим минимальную форму

$$f(x_1, x_2, x_3) = x_1x_2 \oplus \bar{x}_3.$$

Для сравнения приведем выражение минимальной формы для СПДФ:

$$f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_3 + \bar{x}_2\bar{x}_3 + x_1x_2x_3.$$

На рис. 10.7, *а* и *б* показано графическое решение для функции двух переменных.

Ответ: $f(x_1, x_2, x_3) = x_1x_2 \oplus \bar{x}_3$.

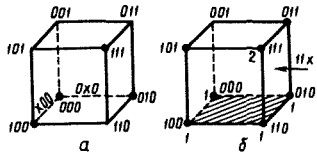


Рис. 10.7. Графические решения задачи минимизации для примера 10.6

10.7. Минимизация функций в базисах Шеффера и Пирса

Рассмотрим сначала функцию Шеффера:

$$x_1/x_2 = \overline{x_1 \,\&\, x_2}\,.$$

Запись логической функции в базисе Шеффера осуществляется следующим образом:

1. Чтобы по таблице истинности записать терм в базисе Шеффера, переменные можно оставить без изменения.

2. Если задавать функцию на нулевых наборах, то в конце выражения следует ставить знак общего отрицания: $(f(x))/1$; если задавать функцию на единичных наборах, то инверсии не нужно.

Пусть функция задана следующей таблицей:

000	1
001	1
010	0
.....	

В базисе И—ИЛИ—НЕ эта функция имеет вид

$$f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2x_3\,.$$

В базисе Шеффера $f(x_1, x_2, x_3) = (x_1/x_2/x_3)/(x_1/x_2)/(x_3/1)$, или, после преобразований:

$$(x_1/x_2)/1)/((x_3/1)x_3) = (x_1/x_2)/1\,.$$

Рассмотрим функцию Вебба:

$$x_1 \downarrow x_2 = \overline{x_1 + x_2}\,.$$

Запись логической функции в базисе Вебба происходит следующим образом:

1. Чтобы по таблице истинности записать терм в базисе Вебба, переменные в нем можно взять с инверсией.

2. Если функция задана на единичных наборах, то в конце выражения следует ставить знак общего отрицания: $(f(x)) \downarrow 0$.

Если функция задана на нулевых наборах, то инверсии не нужно.

Для заданной выше таблицы функция в базисе Вебба запишется так: $(x_1 \downarrow x_2(x_1 \downarrow 0)) \downarrow (x_1 \downarrow x_2 \downarrow x_3) \downarrow 0$, или, после преобразований: $((x_1 \downarrow x_2) \downarrow 0) \downarrow 0$.

Для минимизации логических функций в базисах Шеффера или Вебба можно использовать метод Квайна—Мак-Класки.

После нахождения всех минимальных термов проводится отыскание минимального покрытия с помощью частотно-минимального метода.

Обычно модели определяются своей матрицей инцидентности $Q = [q_{ij}]$, в которой каждой строке взаимно однозначно соответствует слово M_i, столбцу — буква m_j:

$$q_{ij} = \begin{cases} 1, \text{ если } m_j \in M_i, \\ 0, \text{ если } m_j \notin M_i. \end{cases}$$

При ранге матрицы $r = 2$ получается двумерная частотная матрица отношений.

Двумерная частотная матрица отношений $\mathbf{F} = f(i, j)$ связана с матрицей инцидентности $\mathbf{Q} = [q_{ij}]$ так:

$$\mathbf{F} = \mathbf{Q}^{\mathrm{T}}\mathbf{Q},$$

где $\mathbf{Q}^{\mathrm{T}}$ — транспонированная матрица.

Для булевой матрицы $\mathbf{Q} = [q_{ij}], q_{ij} = \{0, 1\}$ покрытием столбцов строками будем называть такое множество строк, что для каждого столбца найдется хотя бы одна строка, на пересечении с которой этот столбец имеет единицу, причем такое свойство множества строк отсутствует при вычеркивании хотя бы одного элемента из этого множества строк. Покрытие столбцов минимальным количеством строк в дальнейшем будем называть минимальным покрытием.

Одним из первых применений минимального покрытия было использование его при нахождении покрытий импликантных таблиц Квайна в связи с возникновением необходимости более компактных представлений булевых функций в практике построения логических схем, а именно, для нахождения МДНФ. Минимальное покрытие находится для матрицы $\mathbf{Q}$, где

строкам соответствуют найденные простые импликанты булевых функций, а столбцам — наборы аргументов, соответствующие единичным значениям минимизируемой функции:

$$q_{ij} = \begin{cases} 1, \text{ если } i\text{-я простая импликанта содержит } j\text{-ю конституенту 1,} \\ 0, \text{ в противном случае.} \end{cases}$$

Простые импликанты, соответствующие строкам, которые входят в минимальное покрытие, соединяются знаками дизъюнкции и образуют МДНФ булевой функции.

Квайн предложил вычеркивать те столбцы из импликантных таблиц, часть которых полностью эквивалентна по 1 какому-либо другому столбцу, и для выбора строки в формулируемое покрытие начинать просмотр со строк, имеющих максимальное количество единиц.

Основные правила для упрощения заданной булевой матрицы **Q** при нахождении ее покрытий следующие:

1) правило поглощения строк.

Если в матрице **Q** имеется такая пара строк Q^i и Q^j, что $Q^i \geq Q^j$, то строка Q^j вычеркивается (проводится сжатие матрицы **Q** по строкам);

2) правило поглощения столбцов.

Если в матрице **Q** имеется такая пара столбцов k^i и k^j, что $k^i \leq k^j$, то столбец k^j вычеркивается (проводится сжатие матрицы **Q** по столбцам).

Последовательным применением этих двух преобразований можно сжать исходную матрицу до «циклической» матрицы, которая содержит среди своих покрытий хотя бы одно из минимальных покрытий исходной матрицы **Q**.

Частотно-минимальный метод минимизации основан на изложенных выше положениях и содержит следующие этапы.

1. Из матрицы **Q** вычеркиваем те строки, которые полностью повторяют всю или часть любой другой строки (поглощаемые строки).

2. Из **Q** вычеркиваем те столбцы, которые полностью повторяют весь или часть любого другого столбца (поглощаемые столбцы).

Пункты 1 и 2 выполняются последовательно до тех пор, пока в матрице **Q** останутся только неполощаемые строки и столбцы.

3. Из оставшихся столбцов матрицы **Q** отыскиваем столбец, в котором находится минимальное количество единиц, находящихся в невычеркнутых строках.

4. Из всех невычеркнутых строк, имеющих единицу в выбранном согласно п. 3 столбце, находим ту строку, в которой имеется максимальное количе-

ство единиц. Если количество столбцов, удовлетворяющих п. 3, больше одного, то строку, соответствующую условиям данного пункта, находим средн всех столбцов, выделенных по п. 3. Эту строку выбираем в оптимальное покрытие.

При равном количестве единиц в строках, удовлетворяющих п. 3 и п. 4, первая из них выбирается для оптимального покрытия или же за счет незначительного усложнения алгоритма подсчитывается вес строк s_k, которые имеют единицы в столбцах, удовлетворяющих п. 3. Затем выбираем строку, имеющую наибольший s_k в оптимальное покрытие. При одинаковых s_k в оптимальное покрытие выбирается первая из них.

5. Найденную строку и все столбцы, в которых эта строка имеет единицу, вычеркиваем из **Q**.

6. Оставшиеся строки проверяем на количество единиц в строке. Если находится строка с числом единиц, равным числу невычеркнутых столбцов, то эта строка вносится в оптимальное покрытие и процесс поиска оптимального покрытия на этом заканчивается; если каждая строка имеет только по одной единице в каждом из оставшихся столбцов, то эти строки вносятся в оптимальное покрытие и процесс поиска также на этом заканчивается. Проверять специально после пяти пунктов алгоритма последнее условие необязательно, так как оптимальное покрытие такой матрицы найдется последовательным выполнением пунктов алгоритма.

В противном случае весь алгоритм, начиная с п. 1, повторяется с оставшимися строками и столбцами.

10.8. Реализация частотно-минимального метода

Для реализации вышеизложенного алгоритма на ЭВМ используются частотные матрицы отношений **F** двух типов $\mathbf{F}^{\text{ст}}$ и $\mathbf{F}^{\text{сб}}$. Матрица $\mathbf{F}^{\text{ст}}$ получается умножением **Q** на $\mathbf{Q}^{\text{т}}$. Эта частотная матрица отношений дает информацию о взаимном влиянии строк **Q** друг на друга. На диагонали $\mathbf{F}^{\text{ст}}$ находится число f_i, которое определяет количество единиц в i-й строке (f_i — собственная частота i-й строки). Элементы f_{ij} (f_{ij} — взаимная частота i-й строки с j-й строкой) матрицы $\mathbf{F}^{\text{ст}}$ определяют количество единиц, которые находятся в тех же самых столбцах как в i-й, так и в j-й строке. Матрица $\mathbf{F}^{\text{сб}}$ получается умножением $\mathbf{Q}^{\text{т}}$ на **Q** ($\mathbf{F}^{\text{сб}} = \mathbf{Q}^{\text{т}}\mathbf{Q}$) и дает информацию о взаимном влиянии столбцов. Здесь f_i показывает количество единиц в i-м столбце, а f_{ij} — количество единиц, которые находятся в тех же самых строках как в i-м, так и в j-м столбцах..

В матрице $\mathbf{F}^{ст}$ находим номера строк матрицы **Q**, которые, согласно п. 1 алгоритма, необходимо вычеркнуть. Это — j-е строки, у которых $f_{ij} = f_{ii}$

В этом случае i-я и j-я строки или полностью эквивалентны ($f_i = f_j$), или j-я строка является частью i-й строки ($f_i > f_j$).

В матрице $\mathbf{F}^{сб}$ находим номера столбцов матрицы **Q**, которые, согласно п. 2 алгоритма, необходимо вычеркнуть. Это — i-е столбцы, у которых $f_{ij} = f_{ii}$. При этом условии j-й столбец матрицы **Q** будет покрыт теми же строками, которые будут покрывать j-й столбец.

Рассмотрим частотно-минимальный метод на примере. Частотная матрица **Q** составляется следующим образом.

Минимальные термы	Термы					
	$\bar{x}_1\bar{x}_2\bar{x}_3x_4$	$\bar{x}_1\bar{x}_2x_3\bar{x}_4$	$\bar{x}_1x_2x_3x_4$	$x_1\bar{x}_2x_3x_4$	$\bar{x}_1x_2\bar{x}_3x_4$	$x_1x_2x_3x_4$
x_1x_2	0	0	0	0	1	1
$\bar{x}_2x_3$	0	1	0	1	0	0
$\bar{x}_1x_2\bar{x}_3$	0	0	0	0	1	0
x_2x_4	0	0	1	0	1	1
$\bar{x}_1x_4$	1	0	1	0	1	0

Если i-й терм входит в состав j-го выражения, то элемент матрицы **Q** $q_{ij} = 1$, иначе $q_{ij} = 0$.

$$\mathbf{Q} = \begin{vmatrix} 000011 \\ 010100 \\ 000010 \\ 001011 \\ 101010 \end{vmatrix}.$$

Составляется матрица $\mathbf{F}^{ст} = \mathbf{Q}\mathbf{Q}^{T}$:

$$\begin{vmatrix} 20121 \\ 2000 \\ 111 \\ 32 \\ 3 \end{vmatrix}$$

матрица симметричная.

Если $f_{ii} = f_{ij}$ или $f_{ii} = f_{ji}$, то сравниваются диагональные члены f_{ii} и f_{jj}.

Если $f_{ii} > f_{ij}$, то из матрицы **Q** вычеркивается j-я строка.

Если $f_{ii} < f_{ij}$, то вычеркивается j-я строка.

В данном примере: $f_{14} = f_{11} = 2$; $f_{44} = 3$; $f_{44} > f_{11} \Rightarrow (3 > 2)$ из матрицы вычеркивается 1-я строка:

$$f_{33} = f_{34} = f_{35} = 1;\ f_{44} = 3;\ f_{44} > f_{55} = 3;\ f_{44} > f_{33};$$

$f_{55} > f_{33} \Rightarrow (3 > 1)$ и из матрицы **Q** вычеркивается 3-я строка.

Количество строк, покрываемых i-м минимальным термом, указывает f_{ii}; количество строк, покрываемых i-м и j-м минимальными термами, вместе указывает f_{ij}.

Равенство $f_{ii} = f_{ij}$ или $f_{ij} = f_{jj}$ показывает, что i-й терм и i-й и j-й термы вместе покрывают одинаковое количество строк. Если $f_{ii} > f_{ij}$, то выбираем $i(j)$ минтерм, так как i-й минтерм покрывает большее количество строк, чем j-й, и наоборот.

Составляем матрицу-столбец $\mathbf{F}^{c6} = \mathbf{Q}^{T}\mathbf{Q}$:

$$\mathbf{F}^{c6} = \begin{vmatrix} 101010 \\ 10100 \\ 2021 \\ 100 \\ 42 \\ 22 \end{vmatrix}.$$

Если $f_{ii} = f_{ij}$ или $f_{ij} = f_{jj}$, то сравнивается f_{ii} и f_{jj}. Если $f_{ii} > f_{jj}$, то вычеркиваем i-й столбец из матрицы **Q**.

В данном примере $f_{11} = f_{13} = f_{15}$; $f_{11} < f_{33} < f_{55}$ (из матрицы **Q** вычеркиваем 3-й и 5-й столбцы); $f_{22} = f_{24}$; $f_{22} = f_{44}$ (из матрицы **Q** вычеркиваем 2-й столбец), если $f_{33} = f_{35}$; $f_{55} > f_{33}$, то вычеркиваем 5-й столбец, а если $f_{56} = f_{66}$; $f_{55} = f_{66}$, то вычеркиваем 5-й столбец.

f_{ii} показывает, что i-я строка исходной матрицы покрывается термами в количестве f_{ii} штук.

f_{ij} показывает количество минимальных термов, покрывающих i и j строки вместе.

Если $f_{ii} = f_{ij}$ или $f_{ij} = f_{jj}$, то это показывает, сколько минтермов покрывают одновременно i-ю и j-ю строки вместе.

Если $f_n > f_{\overline{n}}$ (или наоборот), то вычеркиваем i-й столбец матрицы **Q**, так как для покрытия i-й строки исходной матрицы требуется большее количество термов, чем для покрытия j-й строки.

В результате этих операций получим новую матрицу **Q**:

$$\begin{array}{|c|c|}\hline 010 & 2 \\ \hline 001 & 4 \\ \hline 100 & 5 \\ \hline \end{array}$$

$$\mathbf{F}_2^{\text{ст}} = \begin{array}{|c|}\hline 100 \\ \hline 10 \\ \hline 1 \\ \hline \end{array}.$$

Поглощений нет.

Из матрицы **Q** следует, что оптимальное покрытие составляют минтермы 2, 4, 5, т. е.

$$F = \bar{x}_2 x_3 + x_2 x_4 + \bar{x} x_4 .$$

Алгоритм, реализующий минимально-частотный метод, представлен на рис. 10.8.

1. Вводятся исходные данные. В качестве исходных данных выступают следующие параметры: $I1$ — количество наборов переменных $I1 = 2^m$, где m — количество переменных; $N1$ — количество переменных (m), $N = N + 1$; матрица **IA** — таблица истинности.

2. Вызывается подпрограмма *STOLB*.

В этой подпрограмме осуществляется подсчет количества наборов логических переменных, на которых функция $F(x)$ принимает значение «1» — «истина». Это число фиксируется с помощью переменной M, которая определяется путем подсчета единиц в крайнем правом столбце матрицы **IA**.

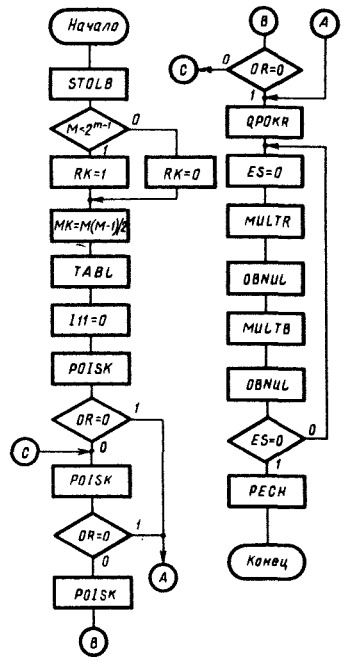


Рис. 10.8. Алгоритм минимизации

3. Сравниваются M и 2^{m-1}. На этом шаге определяется, на каких наборах задавать логическую функцию (1 или 0). Если $M < 2^{m-1}$, то функция задается на единичных наборах (признак $RK = 1$). Если $M \geq 2^{m-1}$, то функция задается на нулевых наборах ($RK = 0$).

4. Вычисляются: $MK = M(M-1)/2$, определяющего максимальное количество промежуточных термов на одном шаге минимизации; определение **IX, IY** — матриц, в которых расположены минимальные термы.

5. Вызывается подпрограмма *TABL*.

В этой подпрограмме анализируется, на каких наборах задается функция, и в соответствии с этим переписываются строки матрицы **IA** (строки — термы, образующие логическую функцию) в матрицу **IT** (матрица исходных термов) и далее будем использовать только матрицу **IT**. Анализ осуществляется через признак *RK*.

6. $I11 = 0$. Переменная $I11$ определяет количество минтермов, находящихся в матрице **ITERM**.

7. Вызов подпрограммы *POISK*.

В этой подпрограмме осуществляется поиск минтермов путем склеивания термов.

Под склеиванием термов понимается операция вида $x_1\bar{x}_2 + x_1x_2 = x_1$.

В машине это выглядит следующим образом:

$$\begin{array}{c} 110 \\ \underline{111} \\ 1\ 2 \end{array}$$

Если хотя бы один раз произошло склеивание термов, то вырабатывается признак $OR = 1$, в противном случае $OR = 0$. Если после выполнения подпрограммы *POISK* $OR = 0$, то это означает, что все минтермы найдены. Можно переходить к поиску минимального покрытия.

Поиск минимальных термов проводится путем сравнения строк матрицы **IT**: 1-я строка сравнивается с 3, 4, 5, ..., M-й, ($M-1$)-я строка сравнивается с M-й. Количество несовпадений по элементам строки фиксируется переменной *IR*. Склеивание термов произойдет лишь в том случае, если $IR = 1$.

При $IR = 1$ i-строку матрицы **IT** переписываем в i-ю строку матрицы **IX**. В то место, где произошло склеивание, записывается цифра «2». Номер этой позиции фиксируется переменной *IR*.

Если при сравнении строк i и j строк $IR = 0$, т. е. i-я и j-я строки одинаковые, то переходим к работе с ($i+1$)-й строкой. Если при сравнении i-й строки с другими хотя бы один раз произошло склеивание ($IR = 1$), то вырабатывается признак $RV = 1$.

Если $RV = 0$, то i-ю строку переписываем в матрицу термов **ITERM**, проверив предварительно, не склеивалась ли данная i-я строка с предыдущими строками с 1-й по $(i-1)$-ю. При сравнении i-й и j-й строк в случае склеивания номер j запомнится в одномерном массиве *KR*, на позиции под номером 17. Проверка осуществляется путем поиска номера i в массиве *KR* по позициям от 1 до 17 текущего.

8. Если после работы подпрограммы *POISK* признак $OR = 1$, то подпрограмма *POISK* выполняется еще раз, но здесь работаем в подпрограмме с матрицей **IX**, а переписываем в матрицу **IY**. Если после этого шага $OR = 1$, то еще раз вызывается подпрограмма *POISK*, но проводится перезапись из **IY** в **IX**. Это проводится до тех пор, пока признак $OR = 1$. Чтобы на этом этапе выполнения программы избежать зацикливания, вводится переменная *NRAZ*, она фиксирует количество обращений к подпрограмме *POISK*. Как только оно превысит заведомо большое число (100), то печатается сообщение *ОШИБКА В ITERM* и выполнение программы заканчивается.

9. $I12 = I11 + 1$, $M1 = M + 1$.

Параметры $I12$ и $M1$ описывают матрицу покрытия *IQ*. Истинный размер матрицы **IQ** (I11XM), кроме этого необходимы дополнительные строка и столбец для обозначения номеров строк и столбцов матрицы **IQ**.

10. Вызов подпрограммы *QPOKR*.

С помощью этой подпрограммы формируем матрицу покрытия **IQ**. Матрица образуется по следующему правилу: $q_{ij} = 1$ — если i-й минтерм входит в состав j неминимального терма; $q_{ij} = 0$ — в противном случае.

11. Вызов подпрограммы *MULTR*.

С помощью этой подпрограммы формируется матрица $\mathbf{F}^{ct}$, при этом 1-я строка **IQ** умножается на себя и попарные произведения суммируются, получаем элемент f_{11}, затем 1-ю строку умножаем на 2-ю, 3-ю и т. д., 1-ю на $I11$, получаем элементы $f_{12}, f_{13}, \ldots, f_{1I11}$. Затем 2-ю на 2-ю, 2-ю на 3-ю, 2-ю на 4-ю, ..., $I11 \times I11$. Это элемент f_{I11I11}. Размер матрицы $\mathbf{F}^{ct}$ равен $I11 \times I11$.

12. Вызов подпрограммы *OBNUL*.

С помощью этой подпрограммы проводится вычеркивание (обнуление) строк матрицы **IQ** вплоть до $I12$ элемента.

Собственно обнуление проводит программа *NR*, которая вызывается подпрограммой *OBNUL*. Подпрограмма *OBNUL* указывает номер строки, которую необходимо обнулнть. Поиск такой строки осуществляется следующим образом:

последовательно выбираются диагональные элементы f_{ii} матрицы **IFSTR**;

f_{ii} сравнивается с элементами строки столбца, на пересечении которых f_{ii} находится;

если есть равенство $f_{ii} = f_{ij}$ или $f_{ji} = f_{ji}$, то вырабатывается признак $ES = 1$ — признак наличия обнуления на данном этапе;

сравниваем f_{ii} и f_{ij}; если $f_{ii} > f_{ij}$, то выполняем:

выбор i-й строки из матрицы **ITERM;**

сравнение каждого элемента этой строки с цифрой «2»;

подсчет количества «2» в этой строке — число их обозначим K; если K — четное число, то вырабатывается признак $KCH = 1$; если K — нечетное число, то $KCH = 0$.

Признак KCH указывает, с отрицанием или без отрицания записывать минимальный терм;

печать строки матрицы **ITERM** по элементам.

Пример 10.7. Пусть задана функция $F(x_1, x_2, x_3, x_4) = \bigvee_{I}(1, 2, 3, 9, 10, 11)$

Найти минимальную форму функции в базисе Шеффера (Вебба).

Решение.

1. Составляем матрицу **IA**:

00000
00011
00101
00111
01000
01010
01100
01110
10000
10011
10101
10111
11000
11010
11100
11110

2. $M = 6$ (результат работы п/п. *STOLB*).

3.

IT =
0001
0010
0011
1001
1010
1011

(результат работы подпрограммы *TABL*).

4. Сравниваем 1-ю и 2-ю строки.

$$\begin{array}{l}0001\\0010\end{array}\quad \begin{array}{l}\text{два несовпадения}\\ IR = 2.\end{array}$$

Сравниваем 1-ю и 3-ю строки:

$$\begin{array}{ll}0001 & IR = 1,\\ 0011 & IR = 3.\end{array}$$

Переписываем 1-ю строку в матрицу **IX** и на 3-ю позицию заносим «2».

$$\mathbf{IX} = \left|0021\right|,\ KR(IY) = 3.$$

Сравниваем 1-ю и 4-ю строки:

$$\begin{array}{ll}0001 & IR = 1,\\ 1001 & IR = 1.\end{array}$$

$$\mathbf{X} = \left|\begin{array}{l}0021\\2001\end{array}\right|,\ KR(2) = 4.$$

Сравниваем 1-ю и 5-ю строки:

$$\begin{array}{ll}0001 & \\ 1010 & IR = 3.\end{array}$$

Сравниваем 1-ю и 6-ю строки:

$$\begin{array}{ll}0001 & IR = 2.\\ 1011 & \end{array}$$

Сравниваем 2-ю и 3-ю строки:

$$\begin{array}{ll}0010 & IR = 1,\ JR = 4,\ KR(3) = 3.\\ 0010 & \end{array}\qquad \mathbf{IX} = \left|\begin{array}{l}0021\\2001\\0012\end{array}\right|.$$

Сравниваем 2-ю и 4-ю строки:

$$\begin{array}{ll}0010 & IR = 3.\\ 1001 & \end{array}$$

Сравниваем 2-ю и 5-ю строки:

$$\begin{array}{ll}0010 & IR = 1,\quad IR = 1.\\ 1010 & 17 = 4,\quad KR(4) = 5.\end{array}\qquad \mathbf{IX} = \left|\begin{array}{l}0021\\2001\\0012\\2010\end{array}\right|.$$

Сравниваем 2-ю и 6-ю строки:

$$\begin{array}{ll}0010 & IR = 2.\\ 1011 & \end{array}$$

Сравниваем 3-ю и 4-ю строки:

$$\begin{array}{ll}0011 & IR = 2.\\ 1001 & \end{array}$$

Сравниваем 3-ю и 5-ю строки:

$$\begin{matrix}0011\\1010\end{matrix}\quad IR = 2.$$

Сравниваем 3-ю и 6-ю строки:

$$\begin{matrix}0011\\1011\end{matrix}\quad IR = 1,\ IR = 1.$$

$$\mathbf{IX} = \begin{vmatrix}0021\\2001\\0012\end{vmatrix}\quad KR(5) = 6.$$

Сравниваем 4-ю и 5-ю строки:

$$\begin{matrix}1001\\1010\end{matrix}\quad IR = 2.$$

Сравниваем 4-ю и 6-ю строки:

$$\begin{matrix}1001\\1011\end{matrix}\ IR = 1,\ IR = 3,\ KR(6) = 6.$$

$$\mathbf{IX} = \begin{vmatrix}0021\\2001\\0012\\2010\\2011\\1021\end{vmatrix}.$$

Сравниваем 5-ю и 6-ю строки:

$$\begin{matrix}1010\\1011\end{matrix}\quad \begin{matrix}IR = 1,\ IR = 4.\\KR(7) = 6.\end{matrix}$$

$$\mathbf{IX} = \begin{vmatrix}0021\\2001\\0012\\2010\\2011\\1021\\1012\end{vmatrix}.$$

Ни один из термов не является минимальным.

Из матрицы **IX** формируем **IY**. Сравниваем 1-ю и 2-ю строки:

$$\begin{matrix}0021\\2001\end{matrix}\quad IR = 2.$$

Сравниваем 1-ю и 3-ю строки:

$$\begin{matrix}0021\\0012\end{matrix}\quad IR = 2.$$

Сравниваем 1-ю и 4-ю строки:

$$\begin{matrix}0021\\2010\end{matrix}\quad IR = 3.$$

Сравниваем 1-ю и 5-ю строки:

0021 $IR = 2$
2011

Сравниваем 1-ю и 6-ю строки:

0021 $IR = 1$, $IR = 1$.
1021 $KR(1) = 6$.
$\mathbf{1Y} = |2021|$.

Сравниваем 1-ю и 7-ю строки:

0021 $IR = 3$.
1012

Сравниваем 2-ю и 3-ю строки:

2001 $IR = 3$.
0012

Сравниваем 2-ю и 4-ю строки:

2001 $IR = 2$.
2010

Сравниваем 2-ю и 5-ю строки:

2001 $IR = 1$, $IR = 3$.
2011 $KR(2) = 5$.

$$\mathbf{1Y} = \begin{vmatrix} 2021 \\ 2021 \end{vmatrix}.$$

Сравниваем 2-ю и 6-ю строки:

2001 $IR = 2$.
1021

Сравниваем 2-ю и 7-ю строки:

2001 $IR = 3$.
1012

Сравниваем 3-ю и 4-ю строки:

0012 $IR = 2$.
2010

Сравниваем 3-ю и 5-ю строки:

0012 $IR = 2$.
2011

Сравниваем 3-ю и 6-ю строки:

0012 $IR = 3$.
1021

Сравниваем 3-ю и 7-ю строки:

0012 $IR = 1$, $IR = 1$.
1012 $KR(3) = 7$.

$$\mathbf{1Y} = \begin{vmatrix} 2021 \\ 2021 \\ 2012 \end{vmatrix}.$$

Сравниваем 4-ю и 5-ю строки:

$$\begin{matrix}2010\\2011\end{matrix}\quad IR = 1,\ IR = 4,\ KR(4) = 5.$$

$$\mathrm{IY} = \begin{vmatrix}2021\\2021\\2012\\2012\end{vmatrix}.$$

Сравниваем 4-ю и 6-ю строки:

$$\begin{matrix}2010\\1021\end{matrix}\quad IR = 3$$

Сравниваем 4-ю и 7-ю строки:

$$\begin{matrix}2010\\1012\end{matrix}\quad IR = 2$$

Сравниваем 5-ю и 6-ю строки:

$$\begin{matrix}1021\\2011\end{matrix}\quad IR = 2.$$

Сравниваем 5-ю и 7-ю строки:

$$\begin{matrix}2011\\1012\end{matrix}\quad IR = 2.$$

Сравниваем 6-ю и 7-ю строки:

$$\begin{matrix}1021\\1012\end{matrix}\quad IR = 2.$$

Признак $RV = 1$, матрица **ITERM** пуста.
Рассмотрим матрицу **IY.**
Сравним 1-ю и 2-ю строки:

$$\begin{matrix}2021\\2021\end{matrix}\quad IR = 0.$$

Сравниваем 2-ю и 3-ю строки:

$$\begin{matrix}2021\\2012\end{matrix}\quad IR = 2,\ RV = 0.$$

Терм 2021 является минимальным, его записываем в матрицу **ITERM.** Сравниваем 3-ю и 4-ю строки:

$$\begin{matrix}2012\\2012\end{matrix}\quad IR = 0.$$

$$\mathbf{ITERM} = \begin{vmatrix}2021\\2012\end{vmatrix}.$$

6. Формируем матрицу **IQ:**

$$\mathbf{IQ} = \begin{vmatrix}101101\\011011\end{vmatrix}.$$

7. Получаем матрицу **IFSTR:**

$$\mathbf{IFSTB} = \begin{vmatrix}4 & 2\\2 & 4\end{vmatrix}.$$

8. Получаем матрицу **IFSTB:**

$$\mathbf{IFSTB} = \begin{vmatrix} 101101 \\ 011011 \\ 112011 \\ 100101 \\ 011011 \\ 111112 \end{vmatrix}.$$

$$f_{11} = f_{13} = f_{14} = f_{16};$$

$$f_{11} < f_{33};\ f_{11} = F_{44};\ f_{11} < f_{66}.$$

Из матрицы **IQ** *вычеркиваем 3-й и 6-й столбцы, а также 4-й.*

$$f_{22} = f_{23} = f_{25} = f_{26};$$

$$f_{22} < f_{33};\ f_{22} = f_{65};\ f_{22} < f_{66}.$$

Из **IQ**, вычеркиваем 5-й столбец.

9. Получаем новую матрицу:

$$\mathbf{IQ} = \begin{vmatrix} 01 \\ 10 \end{vmatrix}.$$

В минимальное покрытие вошли термы 2021, 2012.

Ответ. результат в базисе Вебба: $(x_2 \downarrow (x_4 \downarrow 0)) \downarrow (x_2 \downarrow (x_4 \downarrow 0))$, в базисе Шеффера: $((x_2/1)/x_4)/((x_2/1)/x_4)$.

11. ЛОГИЧЕСКОЕ ОПИСАНИЕ И АНАЛИЗ ЭЛЕКТРОННЫХ СХЕМ

11.1. Логические операторы электронных схем

По зависимости выходного сигнала от входного все электронные схемы [7, 17] можно условно разбить на:

схемы первого рода, содержащие комбинационные схемы, — схемы, выходной сигнал в которых зависит только от состояния входов (наличия входных сигналов), в каждый момент времени;

схемы второго рода, содержащие накапливающие схемы (элементы с памятью), — схемы, выходной сигнал в которых зависит как от входных сигналов, так и от состояния схемы в предыдущие моменты времени.

По количеству входов и выходов схемы бывают: с одним входом и одним выходом, с несколькими входами и одним выходом, с одним входом и несколькими выходами, с несколькими входами и выходами.

По способу осуществления синхронизации схемы бывают: с внешней синхронизацией (синхронные автоматы) и с внутренней синхронизацией (асинхронные автоматы являются их частным случаем).

Практически любая ЭВМ состоит из комбинации схем первого и второго родов разной сложности.

Рассмотрим некоторые конкретные примеры.

На рис. 11.1, *а* показана принципиальная электрическая схема, выполненная на транзисторе.

Схема работает следующим образом. В интервале времени от 0 до t_1 (рис. 11.1, *б*) на входе действует почти нулевое напряжение. За счет делителя $R1$–$R2$ и источника $E_б$ транзистор $ПТ1$ закрыт и на выходе напряжение равно $-E_к$. В момент t_1 происходит изменение напряжения на входе (теперь действует U_1), что изменяет ток в транзисторе $ПТ1$ до такой степени, что транзистор открывается. Через резистор $R_к$ течет ток, который изменя-

ет напряжение на выходе: оно становится близким нулю*. Так продолжается до момента t_2, когда состояние на входе снова изменяется, вызывая соответствующее изменение на выходе. На рис. 11.1, *б* показана временна́я диаграмма изменений состояний на входе и выходе схемы. Напряжения на выходе принимают два значения (высокий уровень — 0, низкий уровень — 1). Тогда логически работу рассмотренной схемы можно описать с помощью функции НЕ (рис. 11.1, *в*), что подтверждается следующим:

Момент времени	$0-t_1$	t_1-t_2	t_2-t_3
Вход	1	0	1
Выход	0	1	0

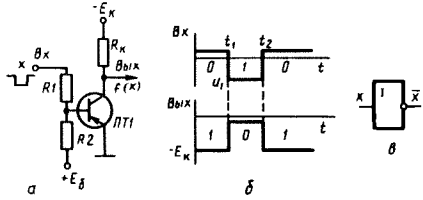


Рис. 11.1. Элементарная электронная схема

Логический оператор схемы — элементарная логическая функция, с помощью которой описывается работа схемы.

Таким образом, рассмотренная выше схема описывается функцией НЕ и называется *инвертором* (рис. 11.1, *в*).

На рис. 11.2, *а* показана схема дизъюнктора, описываемая логическим оператором ИЛИ (рис. 11.2, *в*). Временна́я диаграмма работы этой схемы представлена на рис. 11.2, *б*.

На рис. 11.3, *а* показана комбинация электронных схем дизъюнктора и инвертора, выполненная на транзисторах, а ее логический оператор ИЛИ—НЕ — на рис. 11.3, *в*. Временна́я днаграмма работы этой схемы дана на рис. 11.3, *б*.

Аналогичным образом проводится анализ работы схемы конъюнктора (рис. 11.4, *а*). Временна́я диаграмма работы схемы и ее логнческий оператор представлены на рис. 11.4, *б*, *в*. Основной особенностью этих схем является

* Здесь состояния схемы кодируются следующим образом: состоянию «1» соответствует бо́льший по модулю отрицательный потенциал, а состоянию «0» — близкий к «0» потенциал (положительная логика).

то, что они идентичны схемам, показанным на рис. 11.2, *а* и 11.3, *а*, но при отрицательной логике кодирования, т. е. состоянию «1» соответствует высокий потенциал, состоянию «0» — низкий потенциал. Этим лишний раз подтверждается справедливость законов де Моргана, которые описывают двойственный характер наборов логических функций И—НЕ и ИЛИ—НЕ (рис. 11.5, *а*, *б*, *в*).

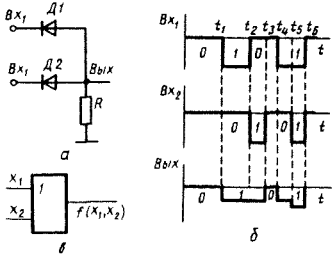


Рис. 11.2. Схема дизъюнктора

На основании вышеизложенного можно прийти к заключению, что различные электронные схемы или их комбинации на логическом уровне могут быть описаны с помощью логических операторов. Такое операторное описание электронных схем позволяет абстрагироваться от физической природы конкретных электронных элементов и осуществлять их анализ. При этом оказывается, что для анализа совсем не обязательно иметь саму схему. Для того чтобы получить значение функции на выходе какой-либо схемы, достаточно записать эту зависимость в виде логических операторов, связанных между собой в соответствии с выполняемой функцией.

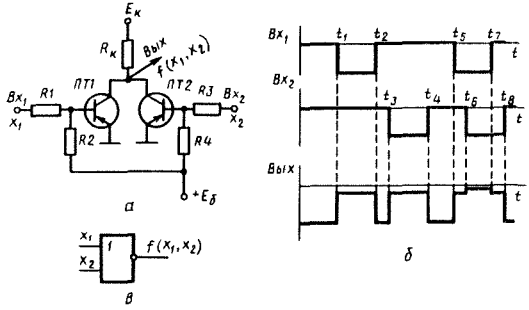


Рис. 11.3. Схема дизъюнктора с инверсией

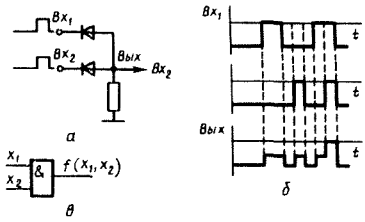


Рис. 11.4. Схема конъюнктора

Для анализа электронных схем с помощью аппарата алгебры логики нужно найти логическую функцию, описывающую работу заданной схемы. При этом исходят из того, что каждому функциональному элементу электронной схемы можно поставить в соответствие логический оператор. Этим самым устанавливается однозначное соответствие между элементами схемы и ее математическим описанием.

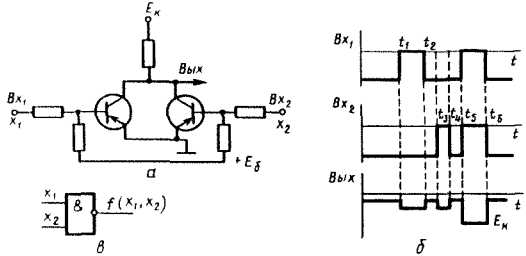


Рис. 11.5. Схема конъюнктора с инверсией

Анализ электронной схемы проводится в два этапа:

1) из принципиальной схемы удаляются все несущественные вспомогательные элементы, которые не влияют на логику работы схемы;

2) через логические операторы выражают все элементы, получая логическое уравнение, которое является моделью функции, выполняемой заданной схемой. Проводится анализ этой функции с целью устранения лишних частей.

Например, схема, представленная на рис. 11.6, может быть описана логическими выражениями $y_1 = \overline{x_2 + \bar{x}_4}$; $y_2 = \bar{x}_1 + \bar{x}_2\bar{x}_3$. Однако с точки зрения инженерного проектирования чаще приходится решать обратную задачу, называемую задачей синтеза электронных схем.

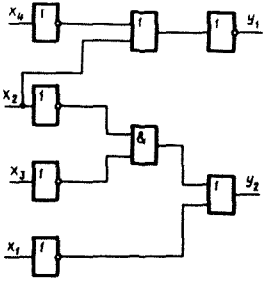


Рис. 11.6. Логическая схема с двумя выходами

Задачу синтеза электронных схем можно сформулировать следующим образом: при заданных входных переменных и известной выходной функции спроектировать логическое устройство, которое реализует эту функцию (при этом могут быть наложены дополнительные ограничения либо в виде системы использующихся логических элементов, либо в виде требоваиий по количеству логических операторов). Следовательно, в результате решения задачи синтеза возиикает логическая схема, воспроизводящая заданную функцию.

Обычно, решая задачи анализа и синтеза, используют полные базисы функций. При этом каждую логическую функцию, входящую в базис, сопоставляют с некоторым физическим элементом. Значит, логическую схему можно заменить структурной схемой, состоящей из физических элементов.

Таким образом, удается соединить математическую задачу синтеза логической схемы с инженерной задачей проектирования электронной схемы. При разработке электронной схемы за осиовные критерии принимают минимум аппаратуры, минимум типов применяемых элементов, максимум надежности.

С точки зрения математической логики задача синтеза решается при обеспечении минимального числа логических операторов, минимального количества типов логических операторов. Можно сформулировать последовательно решаемые задачи при синтезе электронной схемы:

составление математического описания (системы логических уравнений), адекватно отображающего процессы, происходящие в схеме;

анализ логических уравнений и получение минимальной формы для каждой из них в заданном базисе;

переход от логических уравнений к логической (структурной) схеме посредством применения логических операторов.

Проблема синтеза наиболее подробно исследована для конечных автоматов, а для бесконечных автоматов в большинстве случаев она имеет лишь теоретический интерес. Трудности синтеза автоматов зависят от того, как заданы условия функционирования автомата. Чем выразительнее язык задания (т. е. чем он удобнее для заказчика), тем сложнее метод синтеза. Синтез абстрактных цифровых автоматов — один из этапов синтеза автоматов, заключающийся в построении абстрактного автомата (например, его таблицы переходов и выходов) по одному из способов задания отображения «вход—выход», которое должен реализовать этот автомат. Другими словами, с помощью процедуры синтеза происходит переход от описания автомата на начальном языке к описанию его на автоматном языке. Этим вопросам посвящена глава 12.

11.2. Электронные схемы с одним выходом

Схемы с одним выходом и несколькими входами относятся к наиболее простым схемам. Основная сложность при синтезе этих схем состоит в том, чтобы найти выражение для выходной функции в заданном базисе.

Пример 11.1. Синтезировать схему в базисе «НЕ-импликация», если функция имеет вид $f(x_1, x_2, x_3) = \bar{x}_1 \to (x_1\bar{x}_2 + x_3)$

Решение. Перейдем от смешанной системы логических функций к системе «НЕ-импликация» на основе правил перехода:

$$x_1 \to x_2 = \bar{x}_1 + x_2 = \overline{\bar{x}_1 x_2},$$
$$x_1 x_2 = \overline{x_1 \to \bar{x}_2}. \qquad (11.1)$$

Получим $\varphi(x_1, x_2, x_3) = \bar{x}_1 \to \overline{(x_1 \to \bar{x}_2} + x_3) = \bar{x}_1 \to \overline{((x_1 \to x_2) \to x_3)}$.

Функция $\varphi(x_1, x_2, x_3)$ может быть реализована на основе логических операторов «НЕ» и «импликация» (рис. 11.7).

Рис. 11.7. Логическая схема на элементах импликации и инверсии к примеру 11.1

Ответ: логическая схема на рис. 11.7.

Задача синтеза, как правило, имеет различные решения в зависимости от выбранной системы логических элементов. Однако для любой заданной функции алгебры логики почти всегда можно синтезировать схему, соответствующую этой функции. Получение схемы с минимальным количеством логических связок требует нахождения минимальной формы для функции алгебры логики.

Некоторые более сложные схемы, имеющие несколько выходов, могут быть сведены в частном случае к набору схем с одним выходом. Тогда синтез осуществляется путем декомпозиции для каждой выделяемой схемы. Рассмотрим в качестве примера синтез однорядного двоичного сумматора методом декомпозиции.

Пример 11.2. Синтезировать схему, заданную таблицей 11.1, в базисе **И–ИЛИ–НЕ**

Таблица 11.1

a_i	b_i	n_{i-1}	c_i	n_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Здесь a_i, b_i — слагаемые i-го разряда операндов a и b; c_i — сумма слагаемых i-го разряда; n_i, n_{i-1} — соответственно переносы из i-го и $(i-1)$-го разрядов.

Решение. Синтезируемую схему можно рассматривать как схему, состоящую из двух частей: схемы для получения поразрядной суммы c_i (полусумматор) и схемы для получения переноса n_i.

На основе теоремы (10.20) зайдем СНДФ для функции c_i и n_i:

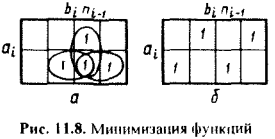


Рис. 11.8. Минимизация функций c_i и n_i к примеру 11.2

Используя минимизирующие карты Карно, получаем минимальные формы для каждой из функций c_i и n_i (рис. 11.8, *а, б*):

$$\begin{cases} c_i = n_{i-1}(\bar{a}_i\bar{b}_i + a_ib_i) + \bar{n}_{i-1}(\bar{a}_ib_i + a_i\bar{b}_i), \\ n_i = n_{i-1}(a_{i-1} + b_i) + a_ib_i. \end{cases} \qquad (11.2)$$

Функции (11.2) могут быть реализованы схемой, представленной на рис. 11.9.

Ответ: логическая схема на рис. 11.9.

Однако способ решения задачи, который показан в примере 11.2, не всегда дает минимальное решение. Например, исходное выражение для c_i может быть записано иначе. Из таблицы 11.1 видно, что поразрядная сумма c_i равна единице тогда, когда одно из слагаемых a_i, b_i или перенос n_{i-1} равен единице, а остальные слагаемые равны нулю и при этом $n_i = 0$ $(\bar{n}_i = 1)$ или когда все три слагаемых равны единице. Поэтому $c_i = (a_i + b_i + n_{i-1})\bar{n}_i + a_ib_in_{i-1}$.

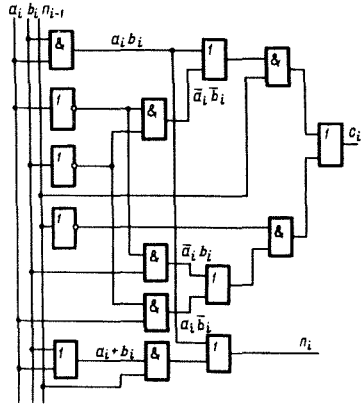


Рис. 11.9. Логическая схема двоичного сумматора к примеру 11.2

Таким образом, окончательное выражение имеет вид

$$\left.\begin{aligned} c_i &= (a_i + b_i + n_{i-1})\overline{n}_i + a_i b_i n_{i-1}; \\ n_i &= n_{i-1}(a_i + b_i) + a_i b_i. \end{aligned}\right\} \qquad (11.3)$$

Функции (11.3) могут быть реализованы логической схемой, представленной на рис. 11.10.

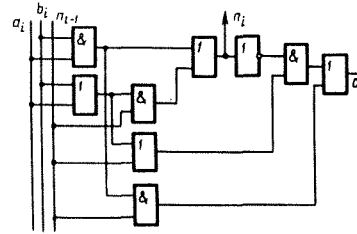


Рис. 11.10. Минимальная схема двоичного сумматора

11.3. Электронные схемы с несколькими выходами

Задача синтеза схемы с n входами и k выходами отличается от задачи синтеза k схем с n входами и одним выходом тем, что при решении необходимо исключить дублирование в k схемах синтезируемых функций.

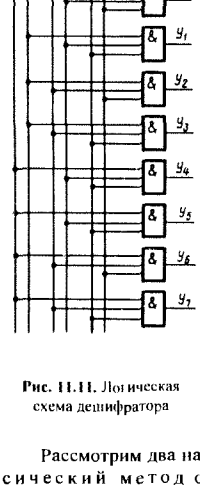


Рис. 11.11. Логическая схема дешифратора

Примером схем с несколькими входами и выходами служит схема дешифратора. Принцип работы дешифратора прост: при заданном наборе входных сигналов на выходе возбуждается одна шина или несколько шин в соответствии с заданной зависимостью. В таблице 11.2 представлена работа дешифратора от трех переменных, у которого возбуждается только один из выходов.

Синтез такой схемы может быть осуществлен, если рассматривать раздельно каждую выходную функцию:

$$y_0 = \bar{x}_1\bar{x}_2\bar{x}_3;\ y_3 = \bar{x}_1x_2x_3;\ y_6 = x_1x_2\bar{x}_3;$$
$$y_1 = \bar{x}_1\bar{x}_2x_3;\ y_4 = x_1\bar{x}_2\bar{x}_3;\ y_7 = x_1x_2x_3.$$
$$y_2 = \bar{x}_1x_2\bar{x}_3;\ y_5 = x_1\bar{x}_2x_3.$$

Реализация этих выражений в виде конъюнктора дает возможность создать логическую схему дешифратора (рис. 11.11).

Однако несложно убедиться в том, что такой подход к построению схем не дает оптимального решения из-за указанных выше особенностей схем с несколькими выходами.

Рассмотрим два наиболее простых метода синтеза таких схем. Классический метод основан на выделении простых импликант заданной системы функций подобно тому, как это делается в методе минимизации Квайна—Мак-Класки, и затем покрытии каждой заданной функции этими простыми импликантами. Далее синтез идет на уровне простых импликант. При этом требуется:

найти простые импликанты заданной системы функций;

выразить каждую заданную функцию через простые импликанты;

синтезировать схему, включающую только эти импликанты и связи между ними.

Таблица 11.2

Входы			Выходы							
x_1	x_2	x_3	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Пример 11.3. Синтезировать схему, функции на выходах которой имеют вид

$$y_1 = f_1(x_1, x_2, x_3) = x_1\bar{x}_2\bar{x}_3 + x_1x_2\bar{x}_3 + \bar{x}_1x_2x_3;$$
$$y_2 = f_1(x_1, x_2, x_3) = x_1x_2 + \bar{x}_2x_3.$$

В качестве базиса взять функции И, ИЛИ, НЕ.

Решение Найдем простые импликанты, разбив множество y_1 на три группы в соответствии с количеством единиц в каждой группе:

$$K_1^0 = \{100\};\ K_2^0 = \{110\};\ K_3^0 = \{111\}.$$

В результате сравнения групп получим

$$K^1 = \{1X0, 11X\} = \{x_1\bar{x}_3 + x_1x_2\}.$$

Простых импликант ранга 3 нет.

Простые импликанты ранга 2

$$K^1 = \{x_1x_2;\ \bar{x}_2x_3,\ x_1\bar{x}_3\}.$$

Окончательный вид выходных функций:

$$y_1 = \underline{x_1x_2} + x_1\bar{x}_3,\ y_2 = \underline{x_1x_2} + \bar{x}_2x_3.$$

В полученных выражениях подчеркнут член, являющийся общим для обоих уравнений, что позволяет упростить окончательный вариант схемы, представленный на рис. 11.12.

Ответ логическая схема на рис. 11.12.

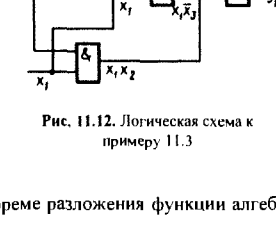


Рис. 11.12. Логическая схема к примеру 11.3

Метод каскадов основан на теореме разложения функции алгебры логики по k переменным (следствие 1):

$$f(x_1, x_2, \ldots, x_n) = x_n f(x_1, x_2, \ldots, x_{n-1}, 1) \vee \bar{x}_n f(x_1, x_2, \ldots, x_{n-1}, 0).$$

Эта формула попеременно применяется к заданной функции столько раз, сколько необходимо, чтобы получить простые логические выражения, которые легко синтезировать, используя двухвходовые элементы И, ИЛИ:

$$
\begin{aligned}
&f(x_1, x_2, \ldots, x_n) = x_n f_1 \vee \bar{x}_n f_2;\\
&f_1(x_1, x_2, \ldots, x_{n-1}) = x_{n-1} f_{11} \vee \bar{x}_{n-1} f_{12};\\
&f_2(x_1, x_2, \ldots, x_{n-1}) = x_{n-1} f_{21} \vee \bar{x}_{n-1} f_{22};\\
&f_{11}(x_1, x_2, \ldots, x_n) = x_{n-2} f_{111} \vee \bar{x}_{n-2} f_{112};\\
&f_{22}(x_1, x_2, \ldots, x_n) = x_{n-2} f_{221} \vee \bar{x}_{n-2} f_{222};\\
&f_{21}(x_1, x_2, \ldots, x_n) = x_{n-2} f_{211} \vee \bar{x}_{n-2} f_{212};\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots
\end{aligned}
\qquad (11.4)
$$

Построенная на основе этих выражений логическая схема на каждом этапе образует последний каскад искомой комбинационной схемы.

Процесс разложения происходит до тех пор, пока не будут получены функции $f_{ijk\ldots l}$, зависящие только от двух аргументов. Далее синтезируется схема, соответствующая системе уравнений минимального ранга.

Пример 11.4. Синтезировать схему в базисе И—ИЛИ—НЕ, выходные функции которой заданы в виде уравнений

$$
\begin{aligned}
\varphi_1 &= x_1\bar{x}_2x_3 \vee x_1x_2\bar{x}_3 \vee \bar{x}_1x_2\bar{x}_3;\\
\varphi_2 &= \bar{x}_1\bar{x}_2x_3 \vee x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3;\\
\varphi_3 &= \bar{x}_1x_2\bar{x}_3 \vee \bar{x}_1x_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3
\end{aligned}
$$

Р е ш е н и е. Применим разложения (11.4) к заданным функциям:

$$
\varphi_1 = \underbrace{x_1\bar{x}_2}_{f_{11}}x_3 \vee \bar{x}_1\underbrace{(x_1x_2 \vee \bar{x}_1x_2)}_{f_{12}};
$$

$$
\varphi_2 = \underbrace{\bar{x}_1\bar{x}_2}_{}x_3 \vee \bar{x}_1\underbrace{(x_1\bar{x}_2 \vee \bar{x}_1\bar{x}_2)}_{},
$$

$$
\varphi_3 = \bar{x}_1\underbrace{(x_1x_2 \vee \bar{x}_1x_2 \vee \bar{x}_1\bar{x}_2)}_{f_{12}=\bar{x}_1 \vee x_2}.
$$

После упрощений выходные уравнения можно записать так:

$$
\begin{aligned}
\varphi_1 &= x_1x_2x_3 \vee x_2\bar{x}_3;\\
\varphi_2 &= \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_2\bar{x}_3;\\
\varphi_3 &= \bar{x}_3(\bar{x}_1 \vee x_2).
\end{aligned}
$$

На рис. 11.13 изображена соответствующая этим уравнениям логическая схема.

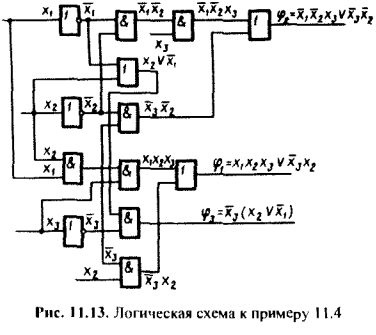


Рис. 11.13. Логическая схема к примеру 11.4

Ответ логическая схема на рис. 11.13.

11.4. Не полностью определенные функции алгебры логики

Не полностью определенная логическая функция n переменных — функция, заданная на числе наборов, меньших 2^n. Встречаются такие функции достаточно часто. Если количество неопределенных наборов m, то путем различных доопределений можно получить 2^m различных функций. В дальнейшем набор, на котором функция $f(x_1, x_2, \ldots, x_n)$ не определена, будем отмечать звездочкой (*). Для наборов, на которых функция не определена, значение логической функции может быть произвольным. Все зависит от некоторых других условий. Например, если взять какой-то десятичный код, то выходная функция определена на 10 из 16 возможных наборов. Остальные наборы — запрещенные. Значит, оставшиеся шесть наборов должны быть доопределены, так как иначе невозможно будет использовать аналитическое представление в виде совершенных нормальных форм (СНФ). Это доопределение очень важно, так как от него зависят результаты решения.

Пример 11.5. Доопределить следующую функцию:

$$f(x_1, x_2, x_3) = \vee(1^*, 2, 3^*, 4, 5, 7^*)$$

Решение Рассмотрим, какие функции получаются, если отмеченные наборы доопределить

На рис 11.14, *а* представлено множество функции, соответствующих заданной функции $f(x_1, x_2, x_3)$. Если функцию $f(x_1, x_2, x_3)$ доопределить на отмеченных наборах нулями, то получится в результате минимизации новая функция $f_6(x_1, x_2, x_3) = x_1\bar{x}_2 \vee \bar{x}_1x_2\bar{x}_3$ (рис 11.14, *б*).

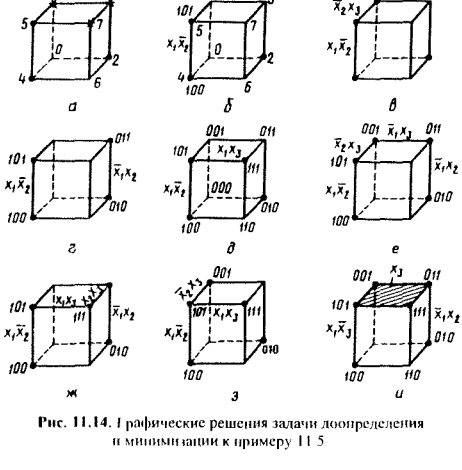


Рис. 11.14. Графические решения задачи доопределения и минимизации к примеру 11.5

Если на отмеченных наборах задать значения для f, равные единице, то $f_{11}(x_1, x_2, x_3) = x_3 \vee x_1\bar{x}_2 \vee \bar{x}_1x_2$ (рис 11.14, *и*).

Другие части рис. 11.14 демонстрируют разные варианты доопределения. Минимальное решение для заданной функции: $f_2(x_1, x_2, x_3) = x_1\bar{x}_2 \vee \bar{x}_1x_2$ (рис. 11.14, *г*)

Ответ: $f_2(x_1, x_2, x_3) = x_1\bar{x}_2 \vee \bar{x}_1x_2$

Пример 11.5 показывает, что доопределение функции существенно влияет на конечный результат минимизации.

При доопределении функций можно руководствоваться следующим правилом: минимальная дизъюнктивная нормальная форма не полностью определенной функции $f(x_1, x_2, \ldots, x_n)$ получается как дизъюнкция наиболее коротких по числу букв импликант функции $\varphi_1(x_1, x_2, \ldots, x_n)$, принимающей значение, равное единице, на всех наборах, где функция не определена, которые в совокупности покрывают все импликанты в совершенной

нормальной форме для функции $\varphi_0(x_1, x_2, \ldots, x_n)$, принимающей значение, равное нулю, на всех наборах, где f не определена.

В самом деле, функция $\varphi_1(x_1, x_2, \ldots, x_n)$ содержит все простые импликанты, которые могут встретиться в i-ом покрытии f, а φ_0 содержит минимальное число простых импликант. Поэтому надо выбрать из такого набора те импликанты из φ_1, которые оптимально покрывают φ_0.

Пример 11.6. Найти минимальную форму для функции четырех переменных:

$$f_2(x_1, x_2, x_3, x_4) = \vee(0, 1^*, 2^*, 4^*, 6, 7^*, 8^*, 9, 11^*, 13^*, 14, 15^*)$$

Решение. На рис. 11.15, *а* представлена функция f_2, отображаемая также таблицей 11.3. Звездочкой на рисунке отмечены неопределенные значения. Для функции $\varphi_0(x_1, x_2, x_3, x_4)$ СНФ может быть получена из таблицы 11.3 при замене символа * на 0.

Соответственно функция $\varphi_1(x_1, x_2, x_3, x_4)$ определяется, если в таблице 11.3 символ * заменить на единицу. Этот случай изображен на рис. 11.15, *б*. Минимизируя показанную функцию, получим

$$\varphi_1(x_1, x_2, x_3, x_4) = \bar{x}_2 x_4 \vee \bar{x}_2 \bar{x}_3 \vee x_2 x_3 \vee x_1 x_4 .$$

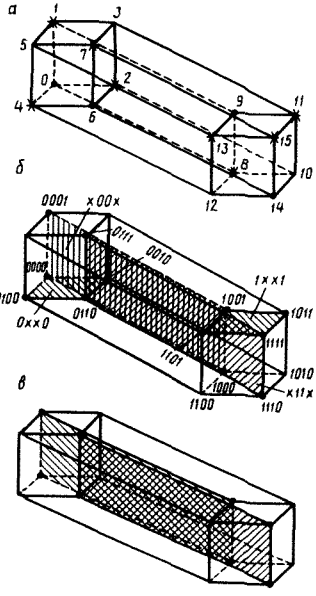


Рис. 11.15. Графический метод минимизации к примеру 11.6

Таблица 11.3

x_1	x_2	x_3	x_4	f	x_1	x_2	x_3	x_4	f
0	0	0	0	1	1	0	0	0	*
0	0	0	1	*	1	0	0	1	1
0	0	1	0	*	1	0	1	0	0
0	0	1	1	0	1	0	1	1	*
0	1	0	0	*	1	1	0	0	0
0	1	0	1	0	1	1	0	1	*
0	1	1	0	1	1	1	1	0	1
0	1	1	1	*	1	1	1	1	*

Таблица 11.4

φ_i	$\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4$	$\bar{x}_1x_2x_3\bar{x}_4$	$x_1\bar{x}_2\bar{x}_3x_4$	$x_1x_2x_3\bar{x}_4$
$\bar{x}_1\bar{x}_4$	∨	∨		
$\bar{x}_2\bar{x}_3$	∨		∨	
x_2x_3		∨		∨
x_1x_4			∨	

Оптимальное доопределение функции, соответствующее минимальному покрытию, может быть найдено по методу Квайна (табл. 11.4). Таким образом, минимальное покрытие $f(x_1, x_2, x_3, x_4) = x_2x_3 \vee \bar{x}_2\bar{x}_3$ (рис. 11.15, *а*).

Ответ: $f(x_1, x_2, x_3, x_4) = x_2x_3 \vee \bar{x}_2\bar{x}_3$.

11.5. Синтез электронных схем с использованием свойств не полностью определенных функций

Пусть необходимо построить функциональную схему для системы, заданной уравнениями:

$$y_1 = f_1(x_1, x_2, \ldots, x_n),\ y_2 = f_2(x_1, x_2, \ldots, x_n). \qquad (11.5)$$

Предположим, что функция y_1 уже построена. Тогда новая функция

$$y_2^* = f_2^*(y_1, x_1, \ldots, x_n).$$

Функция y_2^* — не полностью определенная и в таблице значений функции только половина значений определена. При этом y_1 используется в качестве $(n+1)$-го аргумента, хотя фактически f_2^* зависит только от n аргументов.

Последовательность этапов синтеза схемы будет такой.

Система уравнений вида (11.5) может быть задана либо аналитически, либо в виде таблицы. Теперь нужно построить таблицу значений функции y_2^*, полагая, что y_1 является аргументом, и значения y_2^* не определены для тех строк таблицы, которые отсутствуют в таблице значений y_2. Функция y_2^* записывается в СПДФ:

$$y_2^* = f_2^*(x_1, x_2, \ldots, x_n, y_1) = \bigvee_1 x_1^{\alpha_1}x_2^{\alpha_2}\ldots x_n^{\alpha_n}y_1^{\sigma_1}, \qquad (11.6)$$

где

$$\sigma_1 = \begin{cases} 1, \text{если } y_1 = 1, \\ 0, \text{если } y_1 = 0. \end{cases}$$

Положим, что на неопределенных значениях $f_2^* = 0$, тем самым находим функцию $\varphi_0(x_1, x_2, \ldots, x_n)$ (см. § 11.5). Далее положим $y_2 = 1$ и запишем СНДФ функции $f_2^*(1)$. После этого найдем минимальную форму для $f_2^*(1)$ (рис. 11.15, *б*). Затем составим таблицу покрытия $f_2^*(0)$ и выделим минимальное покрытие (рис. 11.15, *в*). В завершение определим оптимальный набор неопределенных значений f_2^*. При этом f_2^* станет определенной на всех наборах. В результате находится соответствующая МДНФ, на основе которой синтезируется схема.

Пример 11.7. Синтезировать одноразрядный двоичный сумматор с использованием свойств не полностью определенных функций

Решение. При решении будем использовать описание работы двоичного сумматора в виде таблицы 11.1. Предположим, что схема, реализующая функцию u_i, уже синтезирована. Теперь построим таблицу для функции c_i^* (табл. 11.5).

Введем обозначения $u_i = y_1$, $c_i = y_2$. Тогда $c_i^* = y_2^* = f_2^*(a_i, b_i, u_{i-1}, u_i)$. Эта функция c_i^* графически показана на рис. 11.16, *а*.

Доопределение функции c_i^*, записанное в СНДФ, имеет вид: $c_i^*(1) = \bigvee_1(1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 14, 15)$ (рис. 11.16, *б*), $c_i^*(0) = \bigvee_1(2, 4, 8, 15)$.

Минимальную форму для функции $c_i^*(1)$ найдем графическим методом: $c_i^*(1) = a_i\bar{u}_i + b_i\bar{u}_i + u_{i-1}\bar{u}_i + a_ib_iu_{i-1} + a_ib_i\bar{u}_{i-1} + a_iu_{i-1}u_i + \bar{a}_ib_iu_{i-1}$.

Теперь составим таблицу покрытия для функции $c_i^*(0)$ (табл. 11.6).

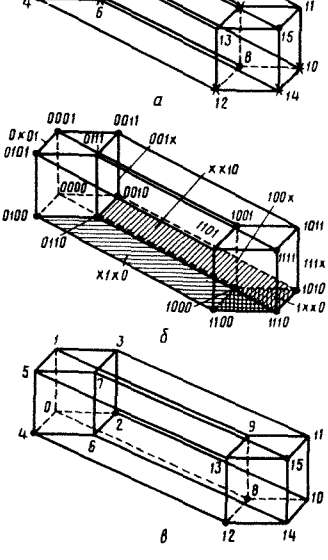


Рис. 11.16. Минимизация двоичного сумматора графическим методом

Таблица 11.5

a_i	b_i	n_{i-1}	n_i	c_i^*	№
0	0	0	0	0	0
0	0	0	1	X	1
0	0	1	0	1	2
0	0	1	1	X	3
0	1	0	0	1	4
0	1	0	1	X	5
0	1	1	0	0	6
0	1	1	1	X	7
1	0	0	0	1	8
1	0	0	1	X	9
1	0	1	0	X	10
1	0	1	1	X	11
1	1	0	0	0	12
1	1	0	1	0	13
1	1	1	0	X	14
1	1	1	1	1	15

Следовательно, минимальное покрытие $a_i\bar{n}_i$; $b_i\bar{n}_i$; $n_{i-1}\bar{n}_i$; $a_ib_in_{i-1}$

По минимальному покрытию находим оптимальное доопределение функции $c_i^*(0)$, в соответствии с которым получаем окончательный вид уравнений (рис. 11.16, *в*).

Таблица 11.6

Первичные импликанты	Исходные термы			
	$\bar{a}_i\bar{b}_in_{i-1}\bar{n}_i$	$\bar{a}_ib_i\bar{n}_{i-1}\bar{n}_i$	$a_i\bar{b}_i\bar{n}_{i-1}\bar{n}_i$	$a_ib_in_{i-1}n_i$
$a_i\bar{n}_i$			∨	
$b_i\bar{n}_i$		∨		
$n_{i-1}\bar{n}_i$	∨			
$a_ib_in_{i-1}$				∨
$a_i\bar{b}_i\bar{n}_{i-1}$			∨	
$\bar{a}_i\bar{n}_{i-1}n_i$				
$\bar{a}_ib_in_{i-1}$	∨			

Ответ $c_i = (a_i + b_i + n_{i-1})\bar{n}_i + a_ib_in_{i-1}$, $n_i = (a_i + b_i)n_{i-1} + a_ib_i$

11.6. Временные булевы функции

Ранее были рассмотрены способы анализа и синтеза схем первого рода (комбинационных), которые невозможно использовать при описании схем второго рода (схем с памятью). Основная особенность схем с

памятью состоит в том, что алгоритм их работы зависит от времени. Следовательно, в число переменных, от которых зависит выходная функция схемы с памятью, должно входить время t. Но время не является двоичной переменной. Поэтому вводится понятие автоматного времени, принимающего дискретные целочисленные значения 0, 1, 2 и т. д. Это означает, что работа схемы с памятью распадается на ряд интервалов, в течение которых автоматное время условно принимает постоянное значение.

Временна́я булева функция (ВБФ) — логическая функция $y = \varphi(x_1, x_2, \ldots, x_n, t)$, принимающая значение {0, 1} при $0 \le t \le s-1$, где s — количество интервалов автоматного времени.

Можно утверждать, что число различных ВБФ равно 2^{s2^n}. В самом деле, если функция времени принимает s значений, т. е. $t = 0, 1, 2, \ldots, s-1$, и каждому интервалу времени соответствует 2^n различных двоичных наборов, то всегда будет $s2^n$ различных наборов. Следовательно, общее число ВБФ равно 2^{s2^n}.

Любая временна́я булева функция может быть представлена в виде

$$y = \varphi(x_1, x_2, \ldots, x_n, t) = \varphi_0\tau_0 \vee \varphi_1\tau_1 \vee \ldots \vee \varphi_{s-1}\tau_{s-1}, \qquad (11.7)$$

где φ_i — конъюнктивный или дизъюнктивный терм от переменных $(x_1, x_2, \ldots, x_n)$; τ_i — вспомогательная функция, принимающая значение $\tau_i = \{0, 1\}$ в момент времени t_i.

Форма представления временных логических функций (11.7) позволяет применить к функциям y все методы упрощения и минимизации, рассмотренные ранее.

Пример 11.8. Преобразовать функцию, представленную таблицей 11.7, в вид (11.7).

Таблица 11.7

x_1	x_2	t	$\varphi(x_1, x_2, t)$	x_1	x_2	t	$\varphi(x_1, x_2, t)$
0	0	0	0	1	0	1	1
0	1	0	0	1	1	1	0
1	0	0	1	0	0	2	0
1	1	0	0	0	1	2	0
0	0	1	0	1	0	2	1
0	1	1	1	1	1	2	1

Решение. Функцию $y = \varphi(x_1, x_2, t)$ представляем совокупностью трех логических функций $\varphi_0(x_1, x_2)$, $\varphi_1(x_1, x_2)$, $\varphi_2(x_1, x_2)$, которые для таблицы 11.7 имеют вид

$$\varphi_0(x_1, x_2) = x_1\bar{x}_2; \ \varphi_1(x_1, x_2) = \bar{x}_1x_2 \vee x_1\bar{x}_2; \ \varphi_2(x_1, x_2) = x_1\bar{x}_2 \vee x_1x_2 = x_1.$$

На основании (11.7) записываем окончательный вид временной логической функции

$$y = x_1\bar{x}_2\tau_0 \vee (\bar{x}_1x_2 \vee x_1\bar{x}_2)\tau_1 \vee \tau_1\tau_2 . \qquad (11.8)$$

Ответ: см. формулу (11.8).

Разложение (11.7) можно применить только к периодическим временным функциям. Переход к схеме от логического выражения вида (11.7) можно осуществить следующим образом.

Предположим, что на выходах некоторой схемы (дешифратора) в моменты времени t появляются сигналы:

если $t_1 = 0$, то на выходе 1 сигнал $\tau_0 = 1$, при $\tau_1 = 0$, $\tau_2 = 0$;

если $t_2 = 1$, то на выходе 2 сигнал $\tau_1 = 1$, при $\tau_0 = 0$, $\tau_2 = 0$;

если $t_3 = 2$, то на выходе 3 сигнал $\tau_2 = 1$, при $\tau_0 = 0$, $\tau_1 = 0$.

Для каждой функции φ_t строим соответствующую логическую схему, не зависящую от переменной t. После этого все схемы соединяем между собой в соответствии с (11.7).

Рекуррентная булева функция (РБФ) — логическая функция, зависящая как от текущих значений x_{t_i} входных переменных, так и от предшествующих значений самой функции $y_{(t-1)}$. Полная аналитическая запись такой функции

$$y_t = \varphi(x_{t_1}, x_{t_2}, \ldots, x_{t_n}, y_{t-1}, y_{t-2}, \ldots, y_{t-k}), \qquad (11.9)$$

$$y_t = \{0, 1\} \text{ при } t > 0,$$

где x'_{t_i} — текущие значения входных переменных; y_t — значения выходных функций в момент времени $j = t - 1$; $t - 2$ и т. д.

Чтобы представить необходимость рекуррентных булевых функций, рассмотрим некоторый физический элемент, работа которого описывается соответствием:

$t \ldots 0$	1	$2 \ldots t_{+1}$	t_{+}
$\lambda \ldots \tau_0$	x_1	$\tau_2 \ldots \tau_{+1}$	τ_t
$y_t = f(\tau, t) \ldots 0$	x_0	$x_1 \ldots x_{+2}$	γ_{+1}

Следовательно, $y_{t+1} = x_t$. Отсюда значение выходного сигнала в момент времени $t+1$ равно значению входного сигнала в момент времени t. Такой элемент называют *задержкой*; $D(t)$ — его логический оператор.

Рассмотрим логическую схему, имеющую цепь обратной связи с включенной в нее схемой задержки (рис. 11.17, *а*). Предположим, что в качестве

схемы с функцией $f(x, y)$ взята логическая схема ИЛИ. Тогда в совокупности эта схема работает так, как показано на временной диаграмме (рис. 11.17, *б*), т. е. $f(x, y) = x_{t+1} \vee y_t$.

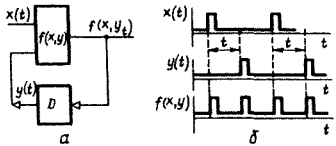


Рис. 11.17. Схема с обратной связью

В схеме рис. 11.17 выходной сигнал зависит как от входного сигнала в данный момент времени, так и от выходного сигнала в предшествующий момент времени. В самом общем случае при наличии n входов и k цепей обратной связи, в которых осуществляется равная задержка, такие схемы могут быть описаны с помощью рекуррентных временных логических функций.

Следовательно, *любая рекуррентная булева функция может быть реализована с помощью набора логических операторов функциональных элементов, представляющих обычные функции алгебры логики, и операторов схем задержки.*

11.7. Последовательностный автомат

Рассмотрим частный случай рекуррентной временной логической функции, так называемой *вырожденной*. Это соответствует случаю, когда на вход функциональной схемы не подаются входные переменные, а поступают только сигналы по цепям обратных связей (рис. 11.18). Для таких случаев рекуррентные функции имеют вид

$$y_{1(t+1)} = \varphi_1(y_{1t}, y_{1(t-1)}, \ldots, y_{1(t-s)}, \ldots, y_{mt}, \ldots, y_{m(t-s)}). \qquad (11.10)$$

Для функций (11.10) должны быть заданы начальные условия при $t = 0$. Предположим, что обратная связь осуществляется с задержкой на 1 такт. Тогда уравнения (11.10) примут вид

$$y_{1(t+1)} = \varphi_1(y_{1t}, y_{2t}, \ldots, y_{mt}). \qquad (11.11)$$

Последовательностные автоматы — это функциональные схемы, описываемые уравнениями типа (11.11), при заданных начальных усло-

виях. Следовательно, в последовательностном автомате сигнал на входе появляется при наличии некоторой последовательности сигналов на входе, т. е. входной сигнал зависит от входных сигналов как в данный момент времени, так и в прошлые моменты. Следовательно, такие схемы должны обладать *памятью*.

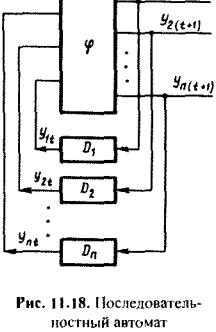


Рис. 11.18. Последовательностный автомат

Рассмотрим функциональную схему последовательностного автомата, описываемого уравнениями типа (11.11) и имеющего три входа (рис. 11.18). На входах и выходах автомата (рис. 11.18) действуют комбинации сигналов, представленные таблицей 11.8.

Пусть начальные значения равны

$$y_{1,0} = 1;\ y_{2,0} = 1;\ y_{3,0} = 1.$$

Используя таблицу состояний, можно построить алгоритм перехода указанного автомата из одного состояния в другое. По условию, в момент времени $t = 0$ на входе действуют сигналы (1, 1, 1), которые вызывают соответственно сигналы на выходе: $y_{1,1} = 1$; $y_{2,1} = 0$; $y_{3,1} = 1$. Через цепь обратной связи эти сигналы подаются на вход автомата и в свою очередь в такте t_1 вызывают новые выходные сигналы $y_{1,2} = 0$; $y_{2,2} = 1$; $y_{3,2} = 0$, поступающие на вход в такте t_2, и т. д. Проведя такой анализ, можно получить полную таблицу состояний заданного автомата в любые моменты времени (см. табл. 11.9).

Таблица 11.8

y_{1t}	y_{2t}	y_{3t}	$y_{1(t+1)}$	$y_{2(t+1)}$	$y_{3(t+1)}$
0	0	0	0	1	1
0	0	1	1	0	0
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	1	0
1	1	0	0	0	1
1	1	1	1	0	1

Если поменять начальные условия, то таблица состояний автомата изменится. Читателю предлагается самому убедиться в этом, приняв $y_{1,0}=1$; $y_{2,0}=1$; $y_{3,0}=0$. Надо помнить, что алгоритм функционирования автомата задается таблицей 11.8. Состояния автомата в процессе его функционирования могут повторяться с определенным периодом (для таблицы 11.9 он равен 6).

Таблица 11.9

t	$y_{1,t}$	$y_{2,t}$	$y_{3,t}$	$y_{1(t+1)}$	$y_{2(t+1)}$	$y_{3(t+1)}$
0	1	1	1	1	0	1
1	1	0	1	0	1	0
2	0	1	0	1	1	0
3	1	1	0	0	0	1
4	0	0	1	1	0	0
5	1	0	0	1	1	1
6	1	1	1	1	0	1
7	1	0	1	0	1	0
8	0	1	0	1	1	0
9	1	1	0	0	0	1

Имея таблицу состояний автомата, можно получить логическое описание устройства, с помощью которого можно реализовать этот автомат.

По таблице 11.8 запишем СНДФ для всех выходных переменных:

$$\begin{cases} y_{1(t+1)} = \bar{y}_{1t}\bar{y}_{2t}y_{3t} \vee \bar{y}_{1t}y_{2t}\bar{y}_{3t} \vee y_{1t}\bar{y}_{2t}\bar{y}_{3t} \vee y_{1t}y_{2t}y_{3t}; \\ y_{2(t+1)} = \bar{y}_{1t}\bar{y}_{2t}\bar{y}_{3t} \vee \bar{y}_{1t}y_{2t}\bar{y}_{3t} \vee y_{1t}\bar{y}_{2t}\bar{y}_{3t} \vee y_{1t}\bar{y}_{2t}y_{3t}; \\ y_{3(t+1)} = \bar{y}_{1t}\bar{y}_{2t}\bar{y}_{3t} \vee \bar{y}_{1t}y_{2t}y_{3t} \vee y_{1t}\bar{y}_{2t}\bar{y}_{3t} \vee y_{1t}y_{2t}\bar{y}_{3t} \vee y_{1t}y_{2t}y_{3t}. \end{cases}$$

Используя методы минимизации, получим после преобразований следующие формулы:

$$\begin{cases} y_{1(t+1)} = (\bar{y}_{1t}\bar{y}_{2t} \vee y_{1t}y_{2t})y_{3t} \vee (\bar{y}_{1t}y_{2t} \vee y_{1t}\bar{y}_{2t})\bar{y}_{3t}; \\ y_{2(t+1)} = \bar{y}_{1t}\bar{y}_{3t} \vee y_{1t}\bar{y}_{2t}; \\ y_{3(t+1)} = y_{1t}\bar{y}_{3t} \vee y_{2t}y_{3t} \vee \bar{y}_{2t}\bar{y}_{3t}. \end{cases} \qquad (11.12)$$

На основании логических формул можно построить функциональную схему логического автомата.

Пример 11.9. Провести анализ электронной схемы последовательностного автомата (рис. 11.19), описываемой системой уравнений

$$\begin{cases} y_{1(t+1)} = y_{2t} y_{1(t-1)} \vee \bar{y}_{2(t-1)}; \\ y_{2(t+1)} = y_{1(t-1)} \bar{y}_{2t} \bar{y}_{2(t-1)}. \end{cases}$$

Решение. Определяем таблицу состояний (табл. 11.10) в общем случае, так как начальные условия не заданы.

Таблица 11.10

y_{1t}	y_{2t}	$y_{1(t-1)}$	$y_{2(t-1)}$	$y_{1(t+1)}$	$y_{2(t+1)}$
0	0	0	0	1	0
0	0	0	1	0	1
0	0	1	0	1	1
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	1	1
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	0	1	0
1	1	1	1	1	0

Теперь в зависимости от различных обстоятельств можно задать начальные условия и получить конкретную таблицу состояний (табл. 11.11).

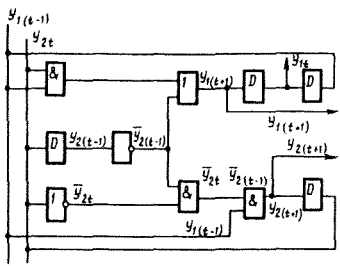


Рис. 11.19. Логическая схема последовательностного автомата (к примеру 11.9)

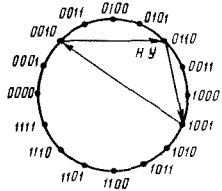


Рис. 11.20. Диаграмма переходов

Получился период повторения, равный трем. Диаграмма переходов для данного случая представлена на рис. 11.20.

Таблица 11.11

y_{1t}	y_{2t}	$y_{1(t-1)}$	$y_{2(t-1)}$	$y_{1(t+1)}$	$y_{2(t+1)}$
0	1	1	0	1	1
1	0	0	1	0	0
0	0	1	0	0	1
0	1	1	0	1	0
1	0	0	1	0	0
0	0	1	0	0	1
0	1	1	0	1	0
1	0	0	1	0	0
0	0	1	0	0	1

Ответ анализ представлен в таблицах 11.10 и 11.11.

11.8. Анализ последовательностных автоматов с помощью рекуррентных булевых функций

Для целей анализа введем понятие обобщенного последовательностного автомата, схема которого представлена на рис. 11.21. Такой автомат описывается следующей системой уравнений:

$$\begin{aligned}
&q_{1(t+1)} = \varphi_1(x_{1(t+1)}, \ldots, x_{n(t+1)}; q_{1t}, \ldots, q_{st}; q_{1(t-k)} \cdots q_{s(t-k)});\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&q_{s(t+1)} = \varphi_s(x_{1(t+1)}, \ldots, x_{n(t+1)}; q_{1t}, \ldots, q_{st}; q_{1(t-k)} \cdots q_{s(t-k)});\\
&z_{1(t+1)} = \psi_1(x_{1(t+1)}, \ldots, x_{n(t+1)}; q_{1t}, \ldots, q_{st}; q_{1(t-k)} \cdots q_{s(t-k)});\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&z_{s(t+1)} = \psi_s(x_{1(t+1)}, \ldots, x_{n(t+1)}; q_{1t}, \ldots, q_{st}; q_{1(t-k)} \cdots q_{s(t-k)}),
\end{aligned} \tag{11.13}$$

где $x_{i(t+1)}$ — входные переменные в момент $(t+1)$; q_{it} — внутреннее состояние схемы в момент времени t; $q_{i(t+1)}$ — внутреннее состояние в момент $(t+1)$; $z_{i(t+1)}$ — выходные переменные в момеит времени $(t+1)$.

Предположим, что некоторая схема описывается таблицей состояний (табл. 11.12).

В соответствии с основной теоремой выходная функция q_{t+1} записывается в виде уравнения

$$q_{t+1} = \bar{x}_{1(t+1)}\bar{x}_{2(t+1)}q_t \vee x_{1(t+1)}\bar{x}_{2(t+1)}\bar{q}_t \vee x_{1(t+1)}\bar{x}_{2(t+1)}q_t \vee x_{1(t+1)}x_{2(t+1)}\bar{q}_t,$$

которое после преобразований и минимизации определяется следующим образом:

$$q_{t+1} = \bar{x}_{2(t+1)} q_t \vee x_{1(t+1)} \bar{q}_t . \qquad (11.14)$$

Схемы, функционирование которых описывается уравнениями типа (11.14), назовем *триггерными схемами*. Они обладают двумя устойчивыми внутренними состояниями (q_t и $\bar{q}_t$), выходной сигнал в таких схемах зависит как от входного сигнала, так и от внутреннего состояния.

Таблица 11.12

$x_{1(t+1)}$	$x_{2(t+1)}$	q_t	q_{t+1}	$\bar{q}_{t+1}$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Таким образом, триггерные схемы являются частным случаем обобщенного последовательностного автомата.

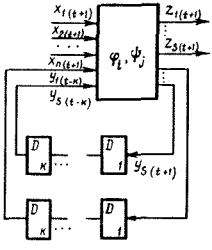


Рис. 11.21. Обобщенный логический автомат

Для системы уравнений (11.13) введем следующие условные обозначения:

$$X_{t+1} = \{x_{1(t+1)}, x_{2(t+1)}, \ldots, x_{n(t+1)}\};$$

$$Q_t = \{q_{1t}, q_{2t}, \ldots, q_{mt}\};$$

$$Q_{t+1} = \{q_{1(t+1)}, q_{2(t+1)}, \ldots, q_{m(t+1)}\};$$

$$Z_{t+1} = \{z_{1(t+1)}, z_{2(t+1)}, \ldots, z_{k(t+1)}\}.$$

Тогда уравнения (11.13) приобретают вид:

$$Q_{t+1} = F(X_{t+1}, Q_t);$$

$$Z_{t+1} = \Phi(X_{t+1}, Q_t)$$

и называются *каноническими уравнениями* автомата. Эти уравнения подтверждают, что в любом последовательностном автомате внутреннее состояние автомата в момент времени $(t+1)$ зависит от внутреннего состояния в момент t и от входных сигналов в момент $(t+1)$. Выходной сигнал

автомата в момент $(t+1)$ зависит также от внутреннего состояния автомата в предшествующий момент и от состояния входов в момент $(t+1)$.

Рассмотрим следующую важную теорему.

Любую логическую схему с памятью можно представить в виде совокупности логических схем И, ИЛИ, НЕ и триггерной схемы.

В самом деле, опираясь на следствие 1 из теоремы о разложении функции по k переменным, любое уравнение из системы (11.13) можно преобразовать следующим образом:

$$q_{s(t+1)} = q_{1t}\overbrace{\varphi_1(x_{1(t+1)},\ldots,x_{n(t+1)};1,q_{2t},\ldots,q_{mt})}^{\bar{u}_{2(t+1)}} \vee$$
$$\vee \bar{q}_{1t}\underbrace{\varphi_1(x_{1(t+1)},\ldots,x_{n(t+1)};0,q_{2t},\ldots,q_{mt})}_{u_{1(t+1)}}.$$

Осуществив замену, получим

$$q_{s(t+1)} = q_{1t}\bar{u}_{2(t+1)} \vee \bar{q}_{1t}u_{1(t+1)},$$

что является уравнением триггерной схемы с двумя входами, причем функции $u_{1(t+1)}$ и $\bar{u}_{2(t+1)}$ являются входными функциями триггера. Проведя подобное преобразование всех уравнений в системе (11.13), приходим к выводу, что теорема справедлива.

Пример 11.10. Уравнение $y_{1(t+1)} = x_{1(t+1)}y_{1t} \vee x_{2(t+1)}$ преобразовать так, чтобы его можно было реализовать с помощью триггера и других логических функций.

Р е ш е н и е. Так как триггер описывается уравнением $y_{t+1} = \bar{x}_{2,t+1}y_t \vee x_{1,t+1}\bar{y}_t$, то, введя обозначения $\bar{x}_{2,t+1} = y^1_{1,t+1}$, $x_{1,t+1} = y^2_{1,t+1}$, $y^1_{1,t+1} = y_{1,t+1}$ при $y_t = 1$, в результате подстановки $y_{1t} = 1$ в исходное уравнение получим $y^1_{1,t+1} = x_{1,t+1} \vee x_{2,t+1}$.

Однако $y^2_{1,t+1} = y_{t+1}$ при $y_t = 0$. Значит, подставив $y_t = 0$ в исходное уравнение, получим $y_{1,t+1} = x_{2,t+1}$.

Ответ: $y_{1,t+1} = (x_{1,t+1} \vee x_{2,t+1})y_t \vee x_{2,t+1}\bar{y}_t$.

11.9. Разновидности триггерных схем

А. Триггерные схемы с раздельной установкой входов (*RS*-триггеры) (рис. 11.22).

Это наиболее распространенная и простая схема, которая может быть реализована с помощью различных элементов И—НЕ, ИЛИ—НЕ. Схема

имеет два входа S и R и два выхода Q и Q'. Работа такой схемы описывается следующими уравнениями:

$$Q = \overline{\overline{S} \cdot Q'};\ Q' = \overline{\overline{R} \cdot Q}.$$

Применив теорему де-Моргана, указанные уравнения можно преобразовать:

$$Q = S \vee \overline{Q}';\ Q' = R \vee \overline{Q}.$$

Рассмотрим следующие наиболее характерные ситуации, которые могут возникнуть при работе *RS*-триггера.

1. $S = R = 0$ (активных сигналов на входе нет).

Это нормальное состояние схемы и уравнения функционирования триггера показывают, что $Q = \overline{Q}'$. В самом деле, если $Q = 1$, то $Q' = 0$, или если $Q = 0$, то $Q' = 1$. Значит, есть два устойчивых состояния, в одном из которых система остается до прихода очередного входного сигнала. Однако уравнения этого не регистрируют.

2. $S = 1$, $R = 0$.

Из уравнения следует, что $Q = 1 \vee \overline{Q}' = 1$ и $Q' = 0 \vee \overline{Q}' = 0$. Следовательно, при таких входных условиях выход принимает значение 1. Такое состояние схемы назовем *единичным*.

3. $S = 0$, $R = 1$.

Это состояние противоположно состоянию 2. Назовем это состояние *нулевым*.

4. $S = R = 1$.

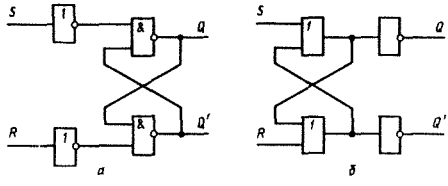


Рис. 11.22. Схема *RS*-триггера

Возникла самая трудная ситуация, так как получается $Q = Q' = 1$. Невозможность такого состояния обусловливает нестабильность рабо-

ты самой схемы. Опасность подобной ситуации состоит в том, что из-за различия временных задержек сигналов в различных цепях может произойти опережение одного из входных сигналов и схема ложно сработает. Такой режим называется «гоночным» состоянием (или состязанием) и вызывает отказ схем. Поэтому разработчик должен предусмотреть невозможность возникновения ситуации, когда $S = R = 1$, т. е. в нормально разработанной схеме такое состояние не должно возникать.

Б. Синхронный *RS*-триггер (рис. 11.23).

Рассмотренная ранее схема может быть использована только в асинхронных автоматах. Для применения *RS*-триггера в синхронных автоматах добавляют специальный тактовый вход, который заставляет схему срабатывать в точно определенные моменты времени. Если на тактовом входе триггера стоит 0, то в точках схемы *A* и *B* состояние не изменяется даже при изменении состояний входов *R* и *S*. Входы *R* и *S* влияют на состояние триггера только при состоянии тактового входа, равном 1. Работа синхронного триггера описывается таблицей 11.13.

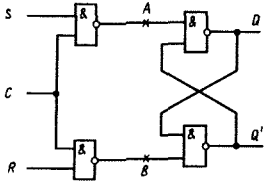


Рис. 11.23. Схема синхронного триггера

В таблице 11.13 символом t_n обозначен момент времени до появления синхроимпульса, а символом t_{n+1} — момент после появления синхроимпульса.

Таблица 11.13

Входы		Момент времени				Примечание
		t_n		t_{n+1}		
S	R	Q_n	Q'_n	Q_{n+1}	Q'_{n+1}	
0	0	0	1	0	1	
0	0	1	0	1	0	БЕЗ ИЗМЕНЕНИЙ
1	0	0	1	1	0	
1	0	1	0	1	0	УСТАНОВКА 1
0	1	0	1	0	1	
0	1	1	0	0	1	УСТАНОВКА 0
1	1	0	1	*	*	} НЕ СТАБИЛЬНО
1	1	1	0	*	*	

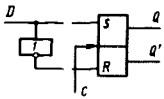


Рис. 11.24. Схема D-триггера

В. *D*-триггер (рис. 11.24).

Если к схеме *RS*-триггера подключить инвертор так, чтобы вход R синхронного *RS*-триггера был всегда инверсным по отношению к входу S, то такая схема будет иметь один вход D, а новая схема называться *D*-триггером (см. рис. 11.24). Поскольку всегда $R = \bar{S}$, гоночное состояние не может возникнуть. Действие *D*-триггера описывается таблицей 11.14.

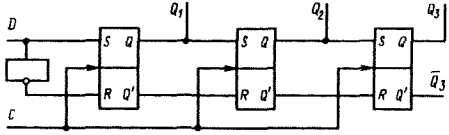


Рис. 11.25. Схема регистра на D-триггерах

D-триггер работает как элемент памяти емкостью 1 бит. Если несколько *D*-триггеров присоединены к одному источнику синхросигналов, то они могут хранить группу разрядов двоичного числа и такая группа триггеров называется *регистром*. Простой *D*-триггер называется в литературе защелкой (latch). Схема регистра показана на рис. 11.25. Группирование триггеров в регистры может осуществляться несколькими путями, например последовательным соединением *RS*- или *D*-триггеров.

Таблица 11.14

Вход D_n	t_n		t_{n+1}	
	Q_n	Q'_n	Q_{n+1}	Q'_{n+1}
0	0	1	0	1
0	1	0	0	1
1	0	1	1	0
1	1	0	1	0

Г. *JK*-триггер.

Это один из наиболее сложных типов триггерных схем, широко использующихся на практике (рис. 11.26). Фактически *JT*-триггер является универсальным триггером: он может работать как любой из описанных выше триггеров. Особенностью этого типа триггеров является то, что состояние $J = K = 1$ четко определено: в этом состоянии $Q = Q'$. Схема имеет

один тактовый и два управляющих входа. Действие схемы описывается таблицей 11.15.

Таблица 11.15

Входы		t_n		t_{n+1}	
J	K	Q_n	$\overline{Q}_n$	Q_{n+1}	$\overline{Q}_{n+1}$
0	0	0	1	0	1
0	0	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	1	0	1	1	0
1	1	1	0	0	1

Однако таблица 11.15 не полностью описывает работу схемы. Это подтверждается таблицей 11.16.

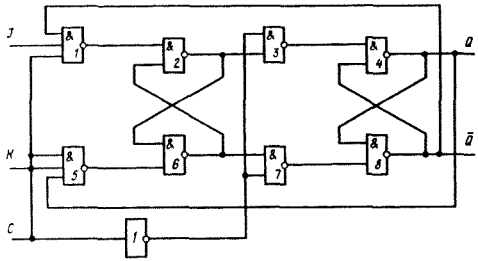


Рис. 11.26. Схема *JK*-триггера

Существуют два типа *JK*-триггеров: первый — *MS*-триггер с главным и подчиненным элементами, второй — с синхронизацией перепадов напряжения. Работа триггеров второго типа гораздо сложнее, и поэтому при проектировании триггеров чаще возникают ошибки. На рис. 11.26 представлена схема *JK*-триггера первого типа на элементах И—НЕ. Фактически это два синхронных *RS*-триггера, соединенных по типу главного и подчиненно-

го и охваченных обратной связью. Опишем работу триггера с помощью булевых выражений: выходной параметр — Q_{n+1}, входные переменные — J, K, Q_n. Такое описание может быть получено только для тех состояний, которые не имеют неопределенных действий:

	00	01	10	11
$\overline{Q}_n$	0	0	1	1
Q_n	1	0	0	1

С помощью карты Карно получим $Q_{n+1} = J \cdot \overline{Q}_n \vee \overline{K} \cdot Q_n$. Время переключения триггера зависит от его конструкции и существенно различно для триггеров первого и второго типов. Следовательно, при проектировании нужно стремиться к тому, чтобы использовать в составе одного устройства однотипные триггеры JK.

Таблица 11.16

Значение		Реакция на следующий импульс
J	K	
0	0	$Q_{n+1} = Q_n$
1	0	Установка $Q_{n+1} = 1$
0	1	Сброс $Q_{n+1} = 0$
1	1	Переключение $Q_{n+1} = \overline{Q}_n$

Задание для самоконтроля

1. В чем заключается смысл введения логического оператора схемы?

2. Найти минимальную форму не полностью определенной функции:

а) $f_1(x_1, x_2, x_3) = \bigvee_1(1^*, 2, 3^*, 4, 5, 7^*)$;

б) $f_2(x_1x_2x_3x_4) = \bigvee_1(0, 1^*, 2, 4, 6, 7^*, 8, 9, 10^*, 12, 14^*)$;

в) $f_3(x_1x_2x_3x_4) = \bigvee_1(1^*, 2, 5^*, 6^*, 8^*, 12, 15)$.

3. Найти методом каскадов набор минимальных функций для следующих уравнений.

а) $f_1(x_1x_2x_3) = AB + BC + AC$;
$f_2(x_1x_2x_3) = \overline{A}\,\overline{B}\,\overline{C} + \overline{ABC}$;
$f_3(x_1x_2x_3) = A + B + C$.

б) $f_1(x_1x_2x_3) = \overline{AB} + BC$;
$f_2(x_1x_2x_3) = AB\overline{C}$;
$f_3(x_1x_2x_3) = \overline{ABC} + A\overline{B}\,\overline{C}$

4. Для заданной системы уравнений:

$$y_{1(t+1)} = x_{1(t+1)}y_{1t} \vee x_{2(t+1)}y_{2t};$$

$$y_{2(t+1)} = x_{2(t+1)}y_{1t} \vee y_{2t};$$

$$z_{1(t+1)} = x_{1(t+1)}x_{2(t+1)}y_{2t}$$

построить электронную схему на элементах И, ИЛИ, НЕ и триггерах.

5. Используя элементы И, ИЛИ, НЕ и триггеры, синтезировать логическую схему, заданную системой уравнений:

$$y_{1,\,t+1} = x_{1,\,t+1}\bar{x}_{2,\,t+1}y_{1,\,t} \vee y_{2,\,t};$$

$$y_{2,\,t+1} = x_{2,\,t+1} \vee y_{1,\,t}.$$

6. Синтезировать логическую схему, заданную следующей таблицей состояний:

Таблица 11.17

$y_{1,t}$	$y_{2,t}$	$y_{2,t-1}$	$y_{1,t+1}$	$y_{2,t+1}$
0	0	0	1	1
0	0	1	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	1	0
1	1	1	0	0

12. МЕТОДЫ ОПИСАНИЯ И СИНТЕЗА ЦИФРОВЫХ АВТОМАТОВ

12.1. Основные понятия теории автоматов

Термин *автомат*, как правило, используется в двух аспектах. С одной стороны, автомат — устройство, выполняющее некоторые функции без непосредственного участия человека. В этом смысле мы говорим, что ЭВМ — автомат, так как после загрузки программы и исходных данных ЭВМ решает заданную задачу без участия человека. С другой стороны, термин «автомат» как математическое понятие обозначает математическую модель реальных технических автоматов [3, 4, 6]. В этом аспекте автомат представляется как «черный ящик», имеющий конечное число входов и выходов и некоторое множество внутренних состояний $Q = \{q_1(t), q_2(t), \ldots, q_n(t)\}$, в которые он под воздействием входных сигналов переходит скачкообразно, т. е. практически мгновенно, минуя промежуточное состояние. Конечно, это условие не выполняется в реальности, так как любой переходный процесс длится конечное время.

Автомат называется *конечным*, если множество его внутренних состояний и множество значений входных сигналов — конечные множества.

В практике часто используется понятие *цифрового автомата*, под которым понимают устройство, предназначенное для преобразования цифровой информации.

Входные сигналы в цифровых автоматах представляются в виде конечного множества мгновенных сигналов. Теоретически это означает, что входные сигналы не имеют длительности, хотя практически это не так. Такое допущение упрощает рассмотрение процессов, происходящих в автоматах, так как все события (состояния) должны относиться к фиксированному моменту времени t. Условно также принимается, что число выходных сигналов $y(t)$ конечно и они возникают в результате действия входных сигналов. При этом следует учитывать, что одновременно с появлением выходного сигнала происходит скачкообразный переход автомата из состояния $q_1(t)$ в состояние $q_2(t)$.

Цифровой автомат называется *правильным*, если выходиой сигиал $y(t)$ определяется только его состоянием $q(t-1)$ или $q(t)$ и не зависит от входных сигналов.

Пусть имеется цифровой автомат с одним входом (рис. 12.1). Математической моделью цифрового автомата является абстрактный автомат, заданный совокупностью шести объектов:

1) конечное множество X входных сигналов (*входной алфавит* автомата):

$$X = \{x_1(t), x_2(t), \ldots, x_n(t)\};$$

2) конечное множество Y выходных сигналов (*выходной алфавит* автомата):

$$Y = \{y_1(t), y_2(t), \ldots, y_k(t)\};$$

3) произвольное множество Q состояний автомата:

$$Q = \{q_1(t), q_2(t), \ldots, q_s(t)\};$$

4) начальное состояние автомата q_0, как элемент миожества Q:

$$q_0(t) \in Q;$$

5) функция $\delta(q, x)$ (*функция перехода* автомата из одного состояния в другое);

6) функция $\lambda(q, x)$ (*функция выходов* автомата).

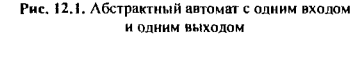


Рис. 12.1. Абстрактный автомат с одним входом и одним выходом

В начальный момент времени t_0 автомат иаходится в состояиии q_0. В каждый момент времени t автомат способен принять входиой сигиал $x(t)$ и выдать соответствующий выходной сигнал $y(t)$.

Через понятие «абстрактный автомат» реализуется некоторое отображение множества слов входного алфавита X в миожество слов выходиого алфавита Y.

Понятие *состояния автомата* используется для описания систем, выходы которых зависят не только от входных сигналов в даниый момент

времени, но и от некоторой предыстории, т. е. сигналов, которые поступили на входы системы ранее. Состояние автомата соответствует некоторой памяти о прошлом, позволяя устранить время как явную переменную и выразить выходные сигналы как функцию состояний и входных сигналов.

Работу абстрактного автомата следует рассматривать применительно к конкретным интервалам времени, так как каждому интервалу дискретности t будет соответствовать свой выходной сигнал $y(t)$. При этом предполагается, что выходной сигнал на одном из выходов автомата может появиться только после соответствующего этому же моменту времени входного сигнала с одновременным переходом из состояния $q(t-1)$ в состояние $q(t)$.

Время для цифрового автомата имеет также важное значение. Для решения задач анализа и синтеза цифровых автоматов обычно вводится *автоматное время*. Функционирование автомата рассматривается через дискретные интервалы времени конечной продолжительности (*интервал дискретности*).

Существуют два способа введения автоматного времени, по которым цифровые автоматы делят на два класса. В *синхронных автоматах* моменты времени, в которых фиксируются изменения состояний автомата, задаются специальным устройством — генератором синхросигналов, выдающим импульсы через равные промежутки времени (постоянный интервал дискретности). В *асинхронных автоматах* моменты перехода автомата из одного состояния в другое заранее не определены и зависят от каких-то событий. В таких автоматах интервал дискретности является переменным.

Общая теория автоматов при сделанных выше допущениях разбивается на две большие части — *абстрактную теорию автоматов* и *структурную теорию автоматов*. Различие между ними заключается в том, что в абстрактной теории мы отвлекаемся от структуры как самого автомата, так и его входных и выходных сигналов. Не интересуясь способом построения автомата, абстрактная теория изучает лишь те переходы, которые претерпевает автомат под воздействием входных сигналов, и те выходные сигналы, которые он при этом выдает. Абстрактная теория автоматов близка теории алгоритмов, является ее дальнейшей детализацией.

В противоположность абстрактной теории, в структурной теории автоматов рассматриваются прежде всего структуры как самого автомата, так и его входных и выходных сигналов. В структурной теории изучаются способы построения автоматов из элементарных автоматов, способы кодирования входных и выходных сигналов элементарными сигналами и т. п.

Автоматы классифицируют по двум наиболее распространенным признакам.

1. Объем памяти. *Памятью автомата* называют число его состояний.

Рассмотренные выше автоматы Поста (или Тьюринга) являются бесконечными автоматами, так как имеют неограниченную память на ленте. Конечными автоматами являются отдельные части ЭВМ или вся машина.

2. Механизм случайного выбора. В *детерминированных автоматах* поведение и структура автомата в каждый момент времени однозначно определены текущей входной информацией и состоянием автомата. В *вероятностных автоматах* они зависят от случайного выбора.

В теории автоматов установлено, что для осуществления различных преобразований информации совсем не обязательно каждый раз строить новые автоматы: в принципе это можно сделать на *универсальном автомате* с помощью программы и соответствующего кодирования.

В теории автоматов наиболее полно описаны синхронные автоматы. В зависимости от способа определения выходного сигнала в синхронных автоматах существуют две возможности:

1) выходной сигнал $y(t)$ однозначно определяется входным сигналом $x(t)$ и состоянием $q(t-1)$ автомата в предшествующий момент;

2) выходной сигнал $y(t)$ однозначно определяется входным сигналом $x(t)$ и состоянием $q(t)$ в данный момент времени. Следовательно, закон функционирования абстрактного автомата может быть задан следующим образом:

для автомата первого рода

$$\begin{cases} q(t) = \delta(q(t-1), x(t)), \\ y(t) = \lambda(q(t-1), x(t)), t = 1, 2, \ldots; \end{cases} \tag{12.1}$$

для автомата второго рода

$$\begin{cases} q(t) = \delta(q(t-1), x(t)), \\ y(t) = \lambda(q(t), x(t)), t = 1, 2, \ldots \end{cases} \tag{12.2}$$

Для дальнейшего анализа целесообразно рассмотреть вопрос о взаимоотношении автоматов первого и второго родов.

Предположим, что произвольный автомат S второго рода задан уравнениями (12.2). Для этого автомата построим новую функцию

$$\lambda_1(q_1, x) = \lambda(\delta(q(t-1), x(t)), x(t)).$$

На основании закона функционирования (12.2)

$$y(t) = \lambda(\delta(q(t-1), x(t)),\ x(t) = \lambda_1(q(t-1), x(t)). \tag{12.3}$$

Получили новый автомат R первого рода, заданный той же функцией перехода $\delta(q, x)$ и функцией выходов $\lambda_1(q, x)$.

Таким образом, для каждого автомата S второго рода существует эквивалентный ему абстрактный автомат R первого рода, функция выходов которого получается в результате подстановки функции переходов автомата S в его сдвинутую функцию выходов:

$$\lambda_1(q, x) = \lambda(\delta(q, x), x).$$

Для дальнейшего изложения примем, что произвольные автоматы первого рода [формула (12.1)] называются *автоматами Мили*, а частный случай автоматов второго рода, для которых сдвинутая функция выходов $\lambda(q, x)$ не зависит от второй переменной x, — *автоматами Мура*.

Закон функционирования автоматов Мура задается в виде:

$$\begin{cases} q(t) = \delta(q(t-1), x(t)), \\ y(t) = \lambda(q(t)). \end{cases} \qquad (12.4)$$

В отличие от автомата Мили, выходной сигнал в автомате Мура зависит только от текущего состояния автомата и в явном виде не зависит от входного сигнала.

Между моделями Мили и Мура существует соответствие, позволяющее преобразовать закон функционирования одного из них в другой или обратно. Такое преобразование порождает пару описаний законов функционирования, эквивалентных в том смысле, что им соответствует одинаковая зависимость между входной X и выходной Y последовательностями.

Совмещенная модель автомата (*С-автомат*). Абстрактный С-автомат — математическая модель дискретного устройства, для которого заданы следующие параметры:

$Q = \{q_1, \ldots, q_s\}$ — множество состояний;

$X = \{x_1(t) \ldots x_n(t)\}$ — входной алфавит;

$Y = \{y_1(t) \ldots y_k(t)\}$ — выходной алфавит типа 1;

$U = \{u_1(t) \ldots u_m(t)\}$ — выходной алфавит типа 2;

$\delta : Q \times X \to Q$ — функция переходов, реализующая отображение $D_\delta \subseteq Q \times X$ в Q;

$\lambda_1 : Q \times X \to Y$ — функция выходов, реализующая отображение $D_{\lambda_1} \subseteq Q \times X$ на Y;

$\lambda_2 : Q \to U$ — функция выходов, реализующая отображение $D_{\lambda_2} \subseteq Q$ на U;

$q_0 \in Q$ — начальное состояние автомата.

Абстрактный С-автомат можно представить в виде устройства с одним входом, на который поступают сигналы из входного алфавита X, и двумя выходами, на которых появляются сигналы из выходных алфавитов Y и U (рис. 12.2).

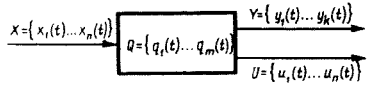


Рис. 12.2. Совмещенная модель автомата с одним входом и двумя выходами

Отличие С-автомата от моделей Мили и Мура состоит в том, что он одновременно реализует две функции выходов λ_1 и λ_2, каждая из которых характерна для этих моделей в отдельности. Этот автомат можно описать следующей системой уравнений:

$$\begin{cases} q(t+1) = \delta(q(t), x(t)); \\ y(t) = \lambda_1(q(t), x(t)); \\ u(t) = \lambda_2(q(t)). \end{cases} \qquad (12.5)$$

Выходной сигнал $u = \lambda_2(q_s)$ выделяется все время, пока автомат находится в состоянии q_s. Выходной сигнал $y_k = \lambda_1(q_s, x_n)$ выдается во время действия входного сигнала x_n при нахождении автомата в состоянии q_s. От С-автомата легко перейти к автоматам Мили или Мура (с учетом возможных сдвигов во времени на один такт), так же как возможна трансформация автомата Мили в автомат Мура, и наоборот.

12.2. Начальные языки описания цифровых автоматов

В зависимости от способа задания функцией переходов и выходов (δ и λ) в настоящее время выделяют два класса языков — начальные языки и стандартные, или автоматные языки. В начальных языках автомат описывается на поведенческом уровне, т. е. функции переходов и выходов обычно в явном виде не заданы. Поведение автомата описывается в терминах входных и выходных последовательностей, реализуемых операторов (отображений) или управляющих последовательностей сигналов, воздействующих на операционный автомат. В автоматных языках поведение

автомат задается путем явного задания функций переходов и выходов. Среди начальных языков следует выделить язык регулярных выражений алгебры событий, язык логических схем алгоритмов, язык граф-схем алгоритмов.

Рассмотрим кратко язык регулярных выражений алгебры событий. Для заданного конечного множества входных букв $X = \{x_1 \dots x_n\}$ регулярное выражение задается следующим образом. Для построения языка регулярных выражений используются три операции над событиями (две бинарные и одна унарная):

1) $A \cup B$ — объединение (дизъюнкция);

2) $A \cdot B$ — умножение (конкатенация);

3) $\{A\}$ — итерация (обозначается также A^*).

Выражение, построенное из букв алфавита X и из символов операций объединения, умножения и итерации с использованием соответствующим образом круглых скобок, называется регулярным выражением в алфавите X. Всякое регулярное выражение R определяет некоторое событие S (S получается в результате выполнения всех операций, входящих в выражение R). События, определяемые таким образом, называются регулярными событиями над алфавитом X. Другими словами, *регулярным событием* называется событие, полученное из элементарных событий (однобуквенных слов x_i), применением конечного числа раз операций дизъюнкции, умножения и итерации. Например, в алфавите из трех букв x, y, z регулярное выражение $x\{x \vee y \vee z\}(y \vee z)$ задает регулярное событие, состоящее из всех слов, которые начинаются буквой x и заканчиваются буквой y, или z. Регулярные события, и только они, представимы в конечных автоматах.

Пусть необходимо описать автомат, выдающий сигнал ω_1 всякий раз, когда происходит изменение входной буквы с x_1 на x_2. Другими словами, сигнал ω_1 должен выдаваться в ответ на любые входные последовательности, кончающиеся последовательностью x_1x_2. Фраза «любые входные последовательности» формализуется всеобщим, или универсальным, событием, состоящим из всех возможных слов в алфавите x_1x_2. Такое событие записывается как $\{x_1 \vee x_2\}$. Тогда событие S_1, в ответ на которое должен выдаваться сигнал ω_1, будет описываться регулярным выражением: $S_1/\omega_1 = \{x_1 \vee x_2\}x_1x_2$.

Рассмотрим кратко язык логических схем алгоритмов. В 1953 г. А. А. Ляпунов предложил записывать алгоритмы в виде конечной строчки, состоящей из символов операторов, логических условий и верхних и нижних стрелок, которым приписаны натуральные числа.

Порядок выполнения операций в автомате определяется *программой* (или *микропрограммой*), представляющей собой совокупность микроопераций и логических условий.

Под *микрооперацией* обычно понимается элементарный процесс переработки информации в одной из частей автомата, происходящий за время такта работы автомата. При этом устройство управления вырабатывает управляющие сигналы, которые обозначим символом $V = \{v_1, v_2, \ldots, v_s\}$. Ход выполнения микроопераций может нарушаться в зависимости от условий, задаваемых множеством $Z = \{z_1 \ldots z_k\}$.

Запись алгоритма, выполненная с учетом вышеизложенного, называется *логической схемой алгоритма* (ЛСА).

Логические схемы алгоритмов удовлетворяют следующим условиям:

1) содержат один начальный ($v_н$) и один конечный оператор ($v_к$);

2) перед оператором $v_н$ и после оператора $v_к$ стрелок быть не должно;

3) вслед за каждым логическим условием всегда стоит верхняя стрелка;

4) не существует двух одинаковых (с одинаковыми цифрами) нижних стрелок;

5) для каждой нижней стрелки должна быть по крайней мере одна соответствующая ей (с одинаковой цифрой) верхняя стрелка;

6) для каждой верхней стрелки должна быть точно одна соответствующая ей (с одинаковой цифрой) нижняя стрелка.

Описание поведения автомата с помощью ЛСА поясним на следующем примере:

$$v_н z_1 \overset{1}{\uparrow} v_1 \overset{1}{\downarrow} z_2 \overset{2}{\uparrow} v_2 v_3 z_3 \overset{2}{\uparrow} v_4 \overset{2}{\downarrow} v_5 v_к . \qquad (12.6)$$

Эта ЛСА имеет операторы начала и конца ($v_н$ и $v_к$), пять операторов ($v_1 \ldots v_5$) и три логических условия (z_1, z_2, z_3). Начальному оператору соответствует некоторое начальное состояние автомата, при котором никакие микрооперации не выполняются. Если в начальном состоянии на первый вход устройства придет сигнал, равный единице ($z_1 = 1$), то устройство перейдет в новое состояние, в котором выполняется оператор v_1 — первый справа после логического условия z_1, а затем проверяется логическое условие z_2. Если же $z_1 = 0$, то выходим по верхней стрелке $\overset{1}{\uparrow}$ и входим по нижней стрелке с той же цифрой на проверку логического условия z_2, минуя выполнение оператора v_1. Если $z_2 = 0$, то переходим к выполнению опера-

тора y_5. Если же $z_2 = 1$, то выполняются операторы y_2 и y_3 и проверяется оператор z_3; если $z_3 = 0$, то оператор y_4 не выполняется, происходит переход к оператору y_5. После выполнения оператора y_5 происходит переход к конечному оператору, т. е. работа дискретного устройства заканчивается.

Другой разновидностью языка, позволяющей описывать логические схемы алгоритмов, является язык *граф-схем алгоритмов* (ГСА). Граф-схема алгоритма — ориентированный связный граф, содержащий одну начальную вершину, произвольное число условных и операторных вершин и одну конечную вершину.

Конечная, операторная и условная вершины имеют по одному входу, начальная вершина входов не имеет. Начальная и операторная вершины имеют по одному выходу, конечная вершина выходов не имеет, условная вершина имеет два выхода, помеченных символами 1 и 0. Граф-схема алгоритма удовлетворяет следующим условиям:

1) входы и выходы вершин соединяются друг с другом с помощью дуг, направленных всегда от выхода к входу;

2) каждый выход соединен только с одним входом;

3) любой вход соединяется по крайней мере с одним выходом;

4) любая вершина графа лежит по крайней мере на одном пути из начальной к конечной вершине;

5) в каждой условной вершине записывается один из элементов множества логических условий $Z = \{z_1 \dots z_l\}$, разрешается в различных условных вершинах запись одинаковых элементов множества Z;

6) в каждой операторной вершине записывается один из элементов множества операторов $Y = \{y_1 \dots y_t\}$, разрешается в различных операторных вершинах запись одинаковых элементов множества Y.

На рис. 12.3 представлена граф-схема алгоритма, описанная выражением (12.6). Язык ГСА используется очень часто при описании алгоритмов функционирования как самого цифрового автомата, так и программ, выполняющих то или иное действие.

При проектировании различных устройств ЭВМ обычно используются *содержательные граф-схемы алгоритмов*, которые описывают не только формальные элементы, но и логические условия и микрооперации в содержательных терминах. Основные правила выполнения содержательных граф-схем алгоритмов и программ, которые студенты разрабатывают в процессе выполнения курсовой работы, представлены в соответствующих ГОСТах. В качестве иллюстрации на рис. 12.4 представлена содержательная граф-схема алгоритма операции сложения чисел, представленных в форме с фиксированной запятой.

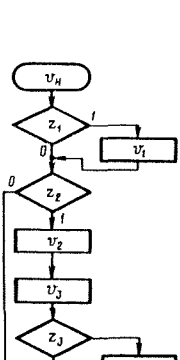


Рис. 12.3. Граф-схема алгоритма

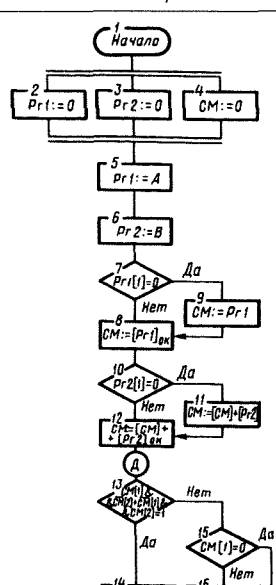


Рис. 12.4. Содержательная граф-схема алгоритма операции сложения чисел с фиксированной занятой

12.3. Автоматные языки для задания автоматных отображений

Среди автоматных языков наиболее распространены таблицы, графы и матрицы переходов и выходов.

Таблица переходов (выходов) представляет собой таблицу с двойным входом, строки которой нумерованы входными буквами, а столбцы — состояния-

ми. На пересечении указывается состояние, в которое переходит автомат (в таблице переходов) или выходной сигнал, выдаваемый им (в таблице выходов).

Описание работы автомата Мили таблицами переходов и выходов иллюстрируется на примере автомата S_1:

$\delta: Q \times X \to Q^*$

	q_1	q_2	q_3
x_1	q_2	q_3	q_2
x_2	q_3	q_2	q_1

$\lambda: Q \times X \to Y^*$

	q_1	q_2	q_3
x_1	y_1	y_3	y_3
x_2	y_2	y_1	y_1

На пересечении столбца q_m и строки x_l в таблице переходов ставится состояние $q_s = \delta(q_m, x_l)$, в которое автомат переходит из состояния q_m под действием сигнала x_l, а в таблице выходов — соответствующий этому переходу выходной сигнал $y_g = \lambda(q_m, x_l)$.

Иногда при задании автоматов Мили используют одну совмещенную таблицу переходов и выходов, в которой на пересечении столбца q_m и строки x_l записываются в виде q_s / y_g следующее состояние и выдаваемый выходной сигнал. Так, для автомата S_1 имеем следующую совмещенную таблицу:

$\delta: Q \times X \to Q$, $\lambda: Q \times X \to Y$:

	q_1	q_2	q_3
x_1	q_2/y_1	q_3/y_3	q_2/y_3
x_2	q_3/y_2	q_2/y_1	q_1/y_1

Так как в автомате Мура выходной сигнал зависит только от состояния, автомат Мура задается одной отмеченной таблицей переходов, в которой каждому ее столбцу приписан кроме состояния q_m еще и выходной сигнал $y_g = \lambda(q_m)$, соответствующий этому состоянию. Пример табличного описания автомата Мура S_2:

$\delta: Q \times X \to Q$, $\lambda: Q \to Y$:

	y_1	y_1	y_3	y_2	y_3
	q_1	q_2	q_3	q_4	q_5
x_1	q_2	q_5	q_5	q_3	q_3
x_2	q_4	q_2	q_2	q_1	q_1

Для частичных автоматов, у которых функции δ или λ определены не для всех пар $(q_m, x_l) \in Q \times X$, на месте неопределенных состояний и выходных сигналов ставится прочерк.

Часто автомат задают с помощью *графа автомата*. Этот язык удобен и нагляден. Граф автомата — ориентированный граф, вершины которого соответствуют состояниям, а дуги — переходам между ними. Две вершины графа автомата q_m и q_s (исходное состояние и состояние перехода) соединяются дугой, направленной от q_m к q_s, если в автомате имеется переход из q_m в q_s, т. е. если $q_s = \delta(q_m, x_l)$ при некотором $x \in X$. Дуге (q_m, q_s) графа автомата приписывается входной сигнал x_l и выходной сигнал $y_k = \lambda(q_m, x_l)$. При описании автомата Мура в виде графа выходной сигнал y_k записывается внутри вершины q_m или рядом с ней. На рис. 12.5, *а*, *б* приведены графы автоматов S_1 и S_2, описанных ранее табличным способом.

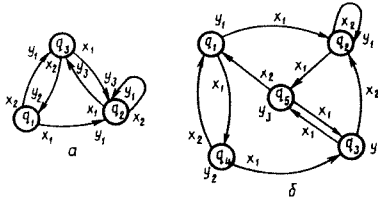


Рис. 12.5. Графы автоматов

Любой автомат может быть задан с помощью графа, но не всякий граф в алфавитах Q, X, Y задает автомат. В графе автомата не должно существовать двух дуг с одинаковыми входными сигналами, выходящих из одной и той же вершины (условие однозначности).

Иногда применяется способ задания автомата с помощью матрицы переходов и выходов, которая представляет собой таблицу с двумя входами. Строки и столбцы таблицы отмечены состояниями. Если существует переход из состояния q_m под действием входного сигнала x в состояние q_s с выдачей входного сигнала y_l, то на пересечении строки q_m и столбца q_s записывается пара x_l / y_l.

Для автомата Мура используется матрица, столбцы которой отмечены выходными сигналами y_l, а на пересечении ее строк и столбцов указываются только входные сигналы x_l.

Ниже приведены матрицы переходов и выходов для рассмотренных ранее автоматов S_1 и S_2:

	q_1	q_2	q_3
q_1		x_1/y_1	q_2/y_2
q_2		x_2/y_1	x_1/y_3
q_3	x_2/y_1	x_1/y_3	

	y_1	y_1	y_3	y_2	y_3
	q_1	q_2	q_3	q_4	q_5
q_1		x_1		x_2	
q_2		x_2			x_1
q_3		x_2			x_1
q_4	x_2		x_1		
q_5	x_2		x_1		

Рассмотрим пример составления графа автомата.

Пусть задано некоторое устройство в виде таблицы 12.1.

Таблица 12.1

Входы			Выходы		
x_1	x_2	x_3	y_1	y_2	y_3
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	1	1	1	0
1	0	0	1	1	1*
1	0	1	—	—	
1	1	0	—	—	
1	1	1	—		

В таблице 12.1 три последние набора не имеют выходного значения, т. е. выходы не определены.

Рассмотрим последовательно входные наборы (рис. 12.6).

1) $x_1x_2x_3 = 000$. Этому набору соответствует начальное состояние q_0, которое не должно изменяться, что отображается дугой, выходящей и входящей в q_0.

2) $x_1x_2x_1 = 001$. Состояние устройства меняется на q_1. Значит, проводится дуга от состояния q_0 к состоянию q_1 и помечается набором 001.

3) $x_1x_2x_1 = 010$. Новое состояние обозначим q_2, для чего соединим q_0 и q_2 дугой и обозначим ее набором 010.

4) $x_1x_2x_1 = 100$. Переход в состояние q_3, которое также отличается от всех предыдущих. Соединим q_0 и q_3 дугой, отмеченной набором 011.

5) $x_1x_2x_1 = 100$. В таблице этому входному набору соответствуют два набора: либо 111, либо 000, что помечено звездочкой *. Для определенности можно уточнить, что выбран набор 000, т. е. возврат в начальное со-

стояние. Значит, надо отметить дугу, входящую и выходящую из q_0 еще набором 100.

Аналогичным образом поступаем с остальными внутренними состояниями q_1, q_2, q_3, в результате чего получаем окончательный граф переходов для рассматриваемого автомата (рис. 12.6).

На основании графа автомата можно составить таблицу переходов (табл. 12.2) или таблицу выходов.

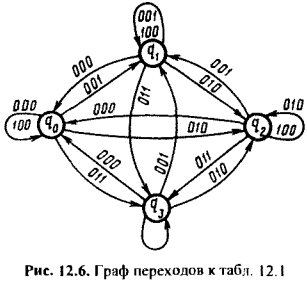


Рис. 12.6. Граф переходов к табл. 12.1

Состояние автомата, вершина графа для которого имеет только исходящие дуги, но не имеет входящих дуг, называется *переходящим*. В такое состояние попасть нельзя, из него можно только выйти.

Состояние автомата называется *тупиковым*, если соответствующая вершина графа не содержит исходящих дуг, но имеет хотя бы одну входящую дугу.

Изолированным состоянием называется такое состояние, которому соответствует вершина графа, не имеющая как входящих, так и исходящих дуг.

Таким образом, с помощью графов абстрактная модель автомата записывается в виде некоторого пространственного изображения, которое помогает при решении задач анализа или синтеза.

Любое разбиение множества состояний автомата на ряд подмножеств, объединяющих некоторые состояния, приводит к понятию *подавтомата*.

Таблица 12.2

Состояния	Входы				
	000	001	010	011	100
q_0	q_0	q_1	q_2	q_3	q_0
q_1	q_0	q_1	q_2	q_3	q_1
q_2	q_0	q_1	q_2	q_3	q_2
q_3	q_0	q_1	q_2	q_3	q_3

12.4. Соединение автоматов

Существуют несколько основных способов соединения автоматов.

Параллельное соединение автоматов S_1 и S_2 представлено на рис. 12.7, *а*. Имеется общий входной алфавит X и некоторое устройство φ, объединяющее выходы автоматов φ: $Y_1 \times Y_2 \to Y$.

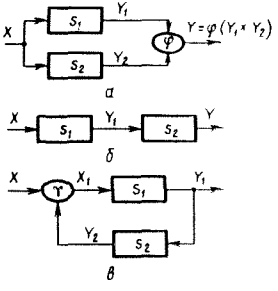


Рис. 12.7. Соединение автоматов

Автоматы заданы:

$$S_1 = \{Q_1, X, Y_1, \delta_1, \lambda_1\},$$

$$S_2 = \{Q_2, X, Y_2, \delta_2, \lambda_2\}.$$

В результате объединения получаем новый автомат $S = \{Q, X, Y, \delta, \lambda\}$, для которого заданы следующие параметры:

множество состояний $Q = Q_1 \times Q_2$ образуется из всевозможных пар состояний автоматов S_1 и S_2, т. е. $Q = \{q_m = (q_{m1}, q_{m2}) \mid q_{m1} \in Q_1, q_{m2} \in Q_2\}$;

входной алфавит X;

выходной алфавит $Y = \varphi(Y_1 \times Y_2)$;

функция переходов определяется правилом $\delta(q_m, x_j) = (\delta_1(q_{m1}, x_j), \delta_2(q_{m2}, x_j))$, где $x_j \in X$;

функция выходов определяется правилом $\lambda(q_{m1}, x_j) = \varphi(\lambda_1(q_{m1}, x_j), \lambda_2(q_{m2}, x_j))$.

Пусть два автомата, соединяемых параллельно, заданы следующими таблицами: таблица 12.3 — $S_1 = \{B, X, Y_1, \delta_1, \lambda_1\}$, таблица 12.4 — $S_2 = \{C, X, Y_2, \delta_1, \lambda_1\}$.

Таблица 12.3

	b_1	b_2	b_3
x_1	b_1/y_1'	b_2/y_2'	b_3/y_2'
x_2	b_3/y_1'	b_1/y_1'	b_2/y_1'

Таблица 12.4

	c_1	c_2
x_1	c_1/y_1''	c_2/y_2''
x_2	c_2/y_2''	c_1/y_1''

Функция φ преобразования выходов в объединенной схеме задана в таблице 12.5.

Получаем следующий результирующий автомат $S = \{Q, X, Y, \delta, \lambda\}$, для которого

$$Q = B \times C = \{(b_1, c_1), (b_1, c_2), (b_2, c_1), (b_2, c_2), (b_3, c_1), (b_3, c_2)\} = \{q_1 \dots q_6\};$$

$$X = \{x_1, x_2\};$$

$$Y = \{y_1, y_2, y_3\}.$$

Функция переходов $\delta = Q \times X \to Q$ задается таблицей 12.6.

Таблица 12.5

	y'_1	y'_2
y''_1	y_1	y_2
y''_2	y_2	y_3

Здесь, например, $\delta(q_3, x_1) = \delta((b_2, c_1), x_1) = (\delta_1(b_1, x_1), \delta_2(c_1, x_1)) = (b_1, c_1) = q_1$.

Таблица 12.6

	q_1	q_2	q_3	q_4	q_5	q_6
x_1	b_1c_1	b_1c_2	b_1c_1	b_1c_2	b_2c_1	b_2c_2
x_2	b_2c_2	b_2c_1	b_2c_2	b_2c_1	b_2c_2	b_2c_1

Функция выходов λ задается таблицей 12.7.

Таблица 12.7

	q_1	q_2	q_3	q_4	q_5	q_6
x_1	y_1	y_2	y_2	y_3	y_2	y_3
x_2	y_2	y_1	y_2	y_1	y_2	y_1

Здесь, например, $\lambda(q_3, x_1) = \lambda((b_2, c_1), x_1) = \varphi(\lambda_1(b_2, x_1), \lambda_2(c_1, x_1)) = \varphi(y'_2, y''_1) = y_2$.

Последовательное соединение двух автоматов представлено на рис. 12.7, *б*. В этом случае первый автомат S_1, второй автомат S_2, т. е. выходы первого автомата являются входом второго. Результирующим автоматом последовательного соединения S_1 и S_2 будет автомат $S = \{Q, X, Y, \delta, \lambda\}$, для которого

$$Q = Q_1 \times Q_2, \text{ или } Q = \{q_m = (q_{m1}, q_{m2}) \mid q_{m1} \in Q_1, q_{m2} \in Q_2\};$$
$$X = X;$$
$$Y = Y;$$

функция переходов:

$$\delta(q_{m1}, x_j) = (\delta_1(q_{m1}, x_j), \delta_2(q_{m2}, \lambda_2(q_{m1}, x_j))), \text{ или}$$
$$\delta(Q \times X) = (\delta_1(Q_1 \times X), \delta_2(Q_2 \times \lambda_1(Q_1 \times X)));$$

функция выходов λ:

$$\lambda(q_m, x_j) = \lambda_2(q_{m2}, \lambda_1(q_{m1}, x_j)), \text{ или } \lambda(Q \times X) = \lambda_2(Q_2 \times \lambda_1(Q_1 \times X)).$$

В качестве примера рассмотрим те же автоматы S_1 и S_2, заданные таблицами 12.8, 12.9.

Таблица 12.8

	b_1	b_2	b_3
x_1	b_1/y_1'	b_1/y_2'	b_2/y_2'
x_2	b_3/y_1'	b_1/y_1'	b_2/y_1'

Таблица 12.9

	c_1	c_2
u_1	c_1/y_1	c_2/y_2
u_2	c_2/y_2	c_1/y_1

Результирующим автоматом последовательного соединения автоматов S_1 и S_2 будет автомат S, для которого

$$Q = \{(b_1, c_1), (b_1, c_2), (b_2, c_1), (b_2, c_2), (b_3, c_1), (b_3, c_2)\} = \{q_1 \dots q_6\};$$

$$X = \{x_1, x_2\};$$

$$Y = \{y_1'', y_2''\}.$$

Функция переходов δ определяется таблицей 12.10.

Таблица 12.10

	q_1	q_2	q_3	q_4	q_5	q_6
x_1	b_1c_1	b_1c_2	b_2c_2	b_2c_1	b_3c_2	b_3c_1
x_2	b_3c_1	b_3c_2	b_1c_1	b_1c_2	b_2c_1	b_2c_2

Здесь, например,

$$\delta(q_3, x_1) = \delta((b_2, c_1), x_1) = (\delta_1(b_2, x_1), \delta_2(c_1, \lambda_1 \times (b_2, x_1))) =$$
$$= (b_1, \delta_2(c_1, y_2')) = (b_1, c_2) = q_2.$$

Функция выходов λ определяется табл. 12.11.

Таблица 12.11

	q_1	q_2	q_3	q_4	q_5	q_6
x_1	y_1	y_2	y_2	y_1	y_2	y_1
x_2	y_1	y_2	y_1	y_2	y_1	y_2

Здесь, например, $\lambda(q_3, x_1) = \lambda((b_2, c_1), x_1) = \lambda_2(c_1, \lambda_1 \times (b_2, x_1)) =$
$= \lambda_2(c_1, y_2') = y_2$.

Соединение автоматов с обратной связью представлено на рис. 12.7, *в*. В этом случае автоматы $S_1 = \{Q_1, X_1, Y_1, \delta_1, \lambda_1\}$ и $S_2 = \{Q_2, Y_1, Y_2, \delta_2, \lambda_2\}$. Имеется некоторый функциональный преобразователь γ, являющийся автоматом без памяти, который реализует отображение γ: $X \times Y_2 \to X_1$. В этом случае, по крайней мере, один из автоматов S_1 или S_2 должен быть автоматом Мура. Пусть S_2 – автомат Мура, у которого $Y_2 = \lambda_2(Q_2)$. Тогда

результирующим автоматом такого соединения с обратной связью будет новый автомат $S = \{Q, X, Y, \delta, \lambda\}$, для которого

$$Q = Q_1 \times Q_2 = \{q_m = (q_{m1}, q_{m2}) \mid q_{m1} \in Q_1, q_{m2} \in Q_2\};$$
$$X = X;$$
$$Y = Y_1;$$

функция переходов:

$$\delta(q_m, x_j) = (\delta_1(q_{m1}, \gamma(x_j, \lambda_2(q_{m2}))), \delta_2(q_{m2}, \lambda_1(q_{m1}, \gamma(x_j, \lambda_2(q_{m2})))),$$

или $\delta(Q \times Z) = \delta_1(Q_1 \times \gamma(X \times \lambda_2(Q_2))), \delta_2(Q_2 \times \lambda_1(Q_1 \times \gamma(X \times \lambda_2(Q_2))))$;

функция выходов:

$$\lambda(q_m, x_j) = \lambda_1(q_{m1}, \gamma(x_j, \lambda_2(q_{m2}))),$$

или $\lambda(Q \times X) = \lambda_1(Q_1 \times \gamma(X \times \lambda_2(Q_2)))$.

В качестве примера рассмотрим два автомата: $S_1 = \{B, P, Y_1, \delta_1, \lambda_1\}$ и $S_2 = \{C, Y, V, \delta_2, \lambda_2\}$, которые заданы таблицами 12.12 и 12.13.

Таблица 12.12

	b_1	b_2	b_3
p_1	b_3/y_3	b_2/y_2	b_3/y_1
p_2	b_2/y_3	b_1/y_1	b_1/y_2

Таблица 12.13

	v_1	v_2
	c_1	c_2
y_1	c_1	c_2
y_2	c_2	c_2
y_3	c_1	c_1

Функциональный преобразователь γ задан таблицей 12.14

Таблица 12.14

	x_1	x_2	x_3
v_1	p_1	p_1	p_1
v_2	p_2	p_2	p_1

Результирующим автоматом соединения с обратной связью будет автомат $S = \{Q, X, Y, \delta, \lambda\}$, для которого

$$Q = B \times C = \{(b_1, c_1), (b_1, c_2), (b_2, c_1), (b_2, c_2), (b_3, c_1), (b_3, c_2)\} = \{q_1, q_2 \ldots q_6\};$$
$$X = \{x_1, x_2, x_3\};$$
$$Y = \{y_1, y_2, y_3\}.$$

Функция переходов $\delta: Q \times X \to Q$ определяется следующим образом (табл. 12.15):

Таблица 12.15

	q_1	q_2	q_3	q_4	q_5	q_6
x_1	b_1c_1	b_2c_1	b_2c_2	b_1c_2	b_1c_1	b_1c_2
x_2	b_1c_1	b_2c_1	b_2c_2	b_1c_2	b_1c_1	b_1c_2
x_3	b_1c_1	b_1c_2	b_2c_2	b_2c_2	b_1c_1	b_1c_2

Здесь, например,

$$\delta(q_1, x_1) = \delta((b_1, c_1), x_1) = (\delta_1(b_2, \gamma(x_1, \lambda_2(c_1)))),$$

$$\delta_2(c_1, \lambda_1(b_2, \gamma(x_1, \lambda_2(c_1)))) = (\delta_1(b_2, \gamma(x_1, v_1))), \delta_2(c_1, \lambda_1(b_2, \gamma(x_1, v_1))) =$$

$$= (\delta_1(b_2, \gamma), \delta_2(c_1, \lambda_1(b_1, p_1))) = (b_2, \delta_2(c_1, x_2)) = (b_2, c_2) = q_3.$$

Функция выходов $\lambda : Q \times X \to Y$ задается таблицей 12.16.

Таблица 12.16

	q_1	q_2	q_3	q_4	q_5	q_6
x_1	y_1	y_3	y_2	y_1	y_1	y_2
x_2	y_1	y_3	y_2	y_1	y_1	y_2
x_3	y_1	y_1	y_2	y_2	y_1	y_1

В этом случае, например,

$$\lambda(q_1, x_1) = \lambda((b_1, c_1), x_1) = \lambda_1(b_2, \gamma(x_1, \lambda_2(c_1))) =$$

$$= \lambda_1(b_1, \gamma(x_1, v_1)) = \lambda_1(b_2, p_1) = y_2.$$

12.5. Синтез управляющего автомата

По функциональному назначению основные устройства ЭВМ можно условно разделить на две категории: *операционные устройства* (ОУ) и *управляющие устройства* (УУ). Отдельные части операционного устройства функционируют в определенной последовательности в зависимости от алгоритма выполняемой операции. Управляющее устройство по сигналу операции вырабатывает необходимые сигналы, по которым запускается выполнение заданной микрооперации. Совокупность микроопераций, объединенных алгоритмом операции, составляет микропрограмму операции, которая, в свою очередь, является связующим звеном между командой (кодом операции) и операционным устройством (аппаратными средствами), предназначенным для преобразования информации.

Управляющее устройство состоит из отдельных логических схем, вырабатывающих управляющие сигналы в заданной последовательности. Такое управляющее устройство можно рассматривать как управляющий автомат типа Мура или Мили.

Чтобы построить схему управляющего автомата, нужно задать микропрограмму работы операционного устройства. Микропрограмма может быть задана в виде граф-схемы* (рис. 12.8, *а*). Микропрограмма выполняется при начальном условии $H = 1$. Условия определяют последовательность выполнения микроопераций. Посмотрим, как связать граф-схему микропрограммы с автоматом Мура или Мили.

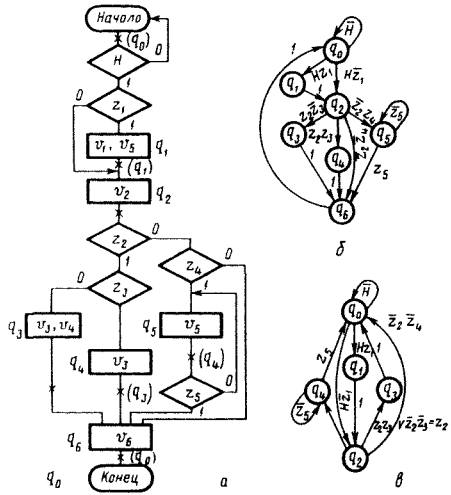


Рис. 12.8. Граф-схема операционного автомата (*а*), граф-переходов автомата Мура (*б*), граф-переходов автомата Мили (*в*)

Для автомата Мура выходной сигнал зависит только от внутреннего состояния, т. е. в нашем случае $y = F(Q)$. Поэтому каждая операторная

* *В. Я. Хартов.* Методические указания к лабораторным работам. Ч. 2. МВТУ, 1987. С. 14–16.

вершина граф-схемы алгоритма должна быть отмечена символом исходного состояния автомата q_i. На рис. 12.8, *а* слева от операторов дана отметка граф-схемы, интерпретируемой как автомат Мура. По этим отметкам можно построить граф переходов автомата Мура (рис. 12.8, *б*), где вершинами являются состояния автомата, а дугами — условия переходов из одного состояния в другое. Таким образом, функции выходов автомата Мура

$$y_j = \bigvee_i q_i,$$

где q_i - состояние автомата, обеспечивающее выработку сигнала y_j.

Переход автомата из одного состояния в другое при отсутствии логических условий (безусловный переход) происходит под действием синхросигнала. Условный переход происходит в том случае, если соответствующее условие $z_i = 1$.

Для построения автомата Мили следует помнить, что выходной сигнал зависит как от внутреннего состояния, так и от входного сигнала (т. е. условия z_i): $y = F(Q, Z)$. Разметка граф-схемы для автомата Мили делается иначе, чем для автомата Мура. Символом q_0 отмечаются вход первой вершины графа, следующей за начальной, и вход конечной вершины. Выходы других операторных вершин отмечаются символом q_i. На рис. 12.8, *а* автомат Мили отмечен символами, взятыми в скобки. Граф переходов автомата Мили показан на рис. 12.8, *в*. В автоматах Мили функции выходов, по которым вырабатываются сигналы микроопераций, определяются по формуле

$$y_j = \bigvee_i q_i \left(\bigvee_k z_k\right),$$

где q_i — состояние автомата, сопровождающегося выработкой сигнала y_j; z_k — логическое условие, определяющее выработку сигнала y_j при переходе автомата из состояния q_i.

В случае безусловного перехода автомата из состояния q_i сигнал y_j определяется только значением q_i.

После построения автомата Мура или Мили функционирование управляющего автомата представляют в виде таблиц переходов и выходов. Для этого проводят кодирование состояний автомата двоичными кодами, определяют тип и количество триггеров. Затем по таблице переходов устанавливают значения сигналов на входах триггеров, при которых осуществляются переходы; определяют функции возбуждения

триггеров и проводят их минимизацию (упрощение). По найденным выражениям строится схема управляющего автомата на выбранных элементах.

12.6. Логическое проектирование управляющего автомата

Рассмотрим методику синтеза управляющего автомата на конкретном примере. Предположим, что требуется создать логическую схему для управления цифровым замком. Срабатывание реле замка происходит в результате поочередного набора комбинации цифр (трех из десяти кнопок) в строгой последовательности. Возвращение автомата в исходное состояние происходит при срабатывании двери в момент ее приоткрывания. Синтез автомата проводится в несколько этапов*.

1. Анализ условия задачи. Очевидно, схема должна содержать не только комбинационную, но и запоминающую часть, так как автомат должен «помнить» предшествующее состояние и в зависимости от него реагировать по-разному на один и тот же входной сигнал. Кроме того, следует принимать в расчет то, что при подаче питания на электронную схему автомат может установиться в любое состояние (в том числе и в состояние «сработал» или некое нештатное состояние, из которого он может и не выйти). Поэтому для обеспечения работоспособности придется предусмотреть сброс автомата в начальное состояние при включении питания, а также при возможных сбоях в функционировании. Переход в начальное состояние *обязателен* и в случае открывания двери (кем-либо из выходящих), в процессе набора кода индивидуумом, желающим проникнуть за охраняемую дверь.

Условимся, что автомат должен выдавать «сигнал тревоги» при вводе неверной цифры, что вполне может оказаться попыткой подобрать код.

Для разработки алгоритма функционирования автомата совершенно необходимо *определить множества входных сигналов, выходных сигналов и множество условий перехода автомата из одного состояния в другое.*

Множество входных сигналов $\{x_i\}$:

x_1 — сигнал введенной правильной 1-й цифры кода;

x_2 — сигнал введенной правильной 2-й цифры кода;

x_3 — сигнал введенной правильной 3-й цифры кода;

x_4 — сигнал, означающий «дверь открыта».

---

* Пример подготовлен В. Гуренко.

Входными сигналами могут быть сигналы, поступающие от любой из 10 кнопок. Для простоты принимаем, что автомат реагирует избирательно лишь на перечисленные сигналы.

Множество выходных сигналов $\{f_i\}$:

f_1 — сигнал, приводящий в состояние «открыто» реле замка (при $f_1 = 0$ считаем, что автомат не разрешает замку открываться);

f_2 — «сигнал тревоги», он может возбуждать схему запуска звукового генератора (попросту говоря, включать сигнализацию).

Множество логических условий переходов $\{z_i\}$:

z_1 — прошел ли сигнал x_1?

z_2 — прошел ли сигнал x_2?

z_3 — прошел ли сигнал x_3?

z_4 — открыта ли дверь?

Ввиду очевидной связи между условиями и входными сигналами будем сами условия обозначать так же, как и входные сигналы $\{x_i\}$.

2. Разработка алгоритма функционирования автомата.

Алгоритм функционирования автомата вполне понятен: переход в состояние открытого замка происходит в том и только в том случае, когда введены верные цифры кода.

Введем *операторы*, которые потребуются для записи алгоритма:

v_1 — ожидать ввода сигнала 1-й цифры кода;

v_2 — ожидать ввода сигнала 2-й цифры кода;

v_3 — ожидать ввода сигнала 3-й цифры кода;

v_4 — выдать выходной сигнал $f_1 = 1$;

v_5 — выдать выходной «сигнал тревоги» $f_2 = 1$.

В дальнейшем станет ясно, зачем операторы $v_1, \ldots, v_3$ определяются столь странной, на первый взгляд, формулировкой.

Ввиду простоты алгоритма перейдем сразу к состоянию его граф-схемы (ГСА) без записи микропрограммы (рис. 12.9).

Операторы $v_1, \ldots, v_3$ указывают на «зависание» автомата, т. е. ожидание ввода очередной информации (рис. 12.9 — граф-схема алгоритма функционирования электронного замка).

Примечание. ГСА не обращает переход автомата в начальное состояние из неопределенного состояния.

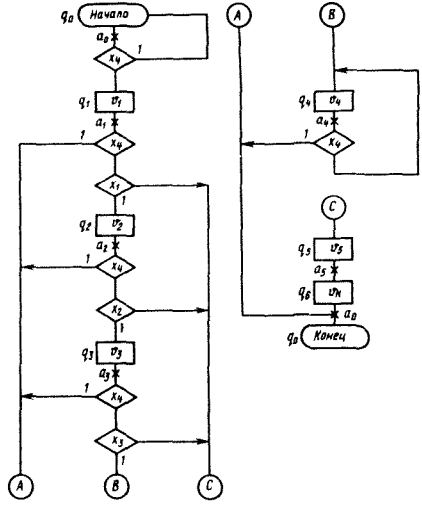


Рис. 12.9. Граф-схема алгоритма функционирования электронного замка

3. *Синтез цифрового автомата.*

Синтез проводится по одному из двух вариантов.

Синтез автомата Мура.

В соответствии с тем, что выходной сигнал в автомате Мура зависит только от состояния, но не зависит от входного сигнала, выполним отметку ГСА по следующим правилам:

— символом q_0 отмечаем вершины «Начало» и «Конец»;

— отмечаем все операторные вершины символами q_1, q_2 и т. д.;

— различные вершины отмечаем различными символами.

Символы q_i обозначают состояние автомата.

Следуя от вершины к вершине всеми возможными путями, получаем граф переходов автомата Мура (рис. 12.10).

Состояние q_6 является любым нештатным (неопределенным), в которое попадает автомат при сбое или выключении питания. Если сбой наступил после ввода правильной первой цифры кода, то на попытку ввести правильную

вторую цифру автомат либо не будет реагировать, оставаясь в состоянии q_6, либо будет воспринимать ее как неправильную первую цифру (при этом сработает сигнализация), если из состояния q_6 автомат перешел в состояние q_0 по дуге x_4. Индивидуум же, набирающий код, остался бы при этом в полном неведении, отчего автомат столь неделикатно обошелся с ним. Поэтому предусматриваем $q_6 \to q_2$ по условию $x_1 x_2 \bar{x}_4$ (аналогично для $q_6 \to q_1$: $x_1 \bar{x}_2 \bar{x}_4$).

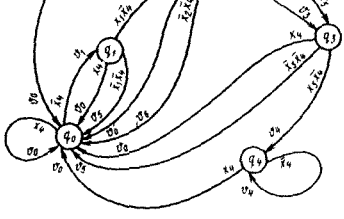


Рис. 12.10. Граф переходов автомата Мура

Сброс автомата в начальное состояние при включении питания можно обеспечить следующим образом. Заметим, что из двух вершин графа (кроме q_5) выходят дуги к вершине q_0 по одному и тому же условию x_4 («дверь открыта»). Так нельзя ли обеспечить $x_4 = 1$ при включении питания даже в том случае, когда дверь заперта? Сделаем так: пусть при подаче питания срабатывает одновибратор, выход которого соединим с одним из входов дизъюнктора, а на другой вход подадим непосредственно сигнал D с контакта двери. Выход схемы и даст требуемый управляющий сигнал x_4.

Имея граф переходов автомата, переходим к составлению таблицы переходов (выходов), предварительно определив число триггеров N_1 («память» автомата) и закодировав состояния

$$N_1 = [\log_2 N_q]^>,$$

где N_q — число состояний, а скобки $[\,]^>$ означают ближайшее целое сверху (в нашем случае $N_1 = 3$):

q_0 000

q_1 001

q_2 010

q_3 011

q_4 100

q_5 101

q_6 110 или 111.

Память реализуем на асинхронных JK-триггерах (табл. 12.17), потому что применять триггеры с одним информационным входом здесь нецелесообразно, так как была бы нужна еще и схема синхронизации. Прочерк означает безразличное состояние. По составленной таблице записываем СНДФ для функций J_i, K_i ($i = \overline{1,3}$). Выход f_2 реализуем так: поскольку $f_2 = 1$, лишь тогда, когда автомат находится в состоянии q_5 (101), конъюнкция $Q_1\overline{Q}_2Q_3$ как раз и даст f_2.

Преобразование СНДФ для функций:

$$J_1 = \overline{Q}_1Q_2Q_3\bar{x}_1\bar{x}_4 + Q_1Q_2\overline{Q}_3\bar{x}_2\bar{x}_4 + \overline{Q}_1Q_2Q_3x_3\bar{x}_4 + \overline{Q}_1Q_2Q_3\bar{x}_3\bar{x}_4 = \overline{Q}_1Q_3\bar{x}_1\bar{x}_4 +$$
$$+ Q_1Q_2Q_3x_4 + \overline{Q}_1Q_2\overline{Q}_3\bar{x}_2\bar{x}_4 = \overline{Q}_1Q_3\bar{x}_1\bar{x}_4 + \underbrace{\overline{Q}_1Q_2Q_3\bar{x}_4}_{A} + \underbrace{\overline{Q}_1\overline{Q}_2\bar{x}_2\bar{x}_4}_{B};$$

$$K_1 = x_4 + Q_1\overline{Q}_2Q_3 + (Q_1Q_2\overline{Q}_3 + Q_1\overline{Q}_2Q_3)\cdot x_1\bar{x}_4 + (Q_1Q_2\overline{Q}_3 +$$
$$+ Q_1\overline{Q}_2Q_3)\cdot x_1x_2\bar{x}_4 = x_4 + Q_1\overline{Q}_2Q_3 + Q_1Q_2x_1\bar{x}_4 + Q_1Q_2x_1x_2\bar{x}_4 =$$
$$= x_4 + \underbrace{Q_1\overline{Q}_2Q_3}_{C} + \underbrace{Q_1Q_2x_1\bar{x}_4}_{D};$$

$$J_2 = \overline{Q}_1\overline{Q}_2Q_3\bar{x}_1\bar{x}_2 = E\cdot x_1;$$

$$K_2 = x_4 + \overline{Q}_1Q_2\overline{Q}_3\bar{x}_2\bar{x}_4 + \overline{Q}_1Q_2Q_3x_3\bar{x}_4 + \overline{Q}_1Q_2Q_3\bar{x}_3\bar{x}_4 =$$
$$= x_4 + \overline{Q}_1Q_2\bar{x}_2\bar{x}_4 + \overline{Q}_1Q_2Q_3\bar{x}_4 = x_4 + B + A;$$

$$J_3 = \overline{Q}_1\overline{Q}_2\overline{Q}_3\bar{x}_1 + \overline{Q}_1Q_2\overline{Q}_3x_2\bar{x}_4 + \overline{Q}_1Q_2\overline{Q}_3\bar{x}_2\bar{x}_4 + Q_1Q_2x_1x_2\bar{x}_4 =$$
$$= \overline{Q}_1\overline{Q}_3(\overline{Q}_2\bar{x}_1 + Q_2\bar{x}_2\bar{x}_4) + \overline{Q}_1Q_2\overline{Q}_3\bar{x}_1 + Q_1Q_2x_1x_2\bar{x} = \overline{Q}_1\overline{Q}_3\bar{x}_4 + D\cdot x_2;$$

$$K_3 = x_4 + \overline{Q}_1\overline{Q}_2Q_3x_1\bar{x}_4 + \overline{Q}_1Q_2Q_3x_3\bar{x}_4 + Q_1\overline{Q}_2Q_3 + Q_1Q_2x_1\bar{x}_4 =$$
$$= x_4 + E\cdot x_1 + A\cdot x_3 + C + D;$$

$$f_1 = Q_1\overline{Q}_2\overline{Q}_3 \text{ [соответствует состоянию } q_4 \text{ (100)]}.$$

Примечание. Мы синтезировали автомат Мура, и поэтому в y_1 и y_2 включить входные сигналы из вне

Таблица 12.17

Текущее состояние				Условие перехода	Входные сигналы комбинационной схемы x_i				Последующее состояние				Выходные сигналы комбинационной схемы $J_i K_i$						Выходной сигнал
имя	код								имя	код									
	Q_1^t	Q_2^t	Q_3^t		x_1	x_2	x_3	x_4		Q_1^{t+1}	Q_2^{t+1}	Q_3^{t+1}	J_1	K_1	J_2	K_2	J_3	K_3	f_1
Все, кроме q_5	×	×	×	x_4	—	—	—	1	q_0	0	0	0	0	1	0	1	0	1	0
q_0	0	0	0	$\bar{x}_4$	—	—	—	0	q_1	0	0	1	0	0	0	0	1	0	0
q_1	0	0	1	$x_1\bar{x}_4$	1	—	—	0	q_2	0	1	0	0	0	1	0	0	1	0
q_1	0	0	1	$\bar{x}_1\bar{x}_4$	0	—	—	0	q_5	1	0	1	1	0	0	0	0	0	0
q_2	0	1	0	$x_2\bar{x}_4$	—	1	—	0	q_3	0	1	1	0	0	0	0	1	0	0
q_2	0	1	0	$\bar{x}_2\bar{x}_4$	—	0	—	0	q_5	1	0	1	1	0	0	1	1	0	0
q_3	0	1	1	$x_3\bar{x}_4$	—	—	1	0	q_4	1	0	0	1	0	0	1	0	1	1
q_3	0	1	1	$\bar{x}_3\bar{x}_4$	—	—	0	0	q_5	1	0	1	1	0	0	1	0	0	0
q_4	1	0	0	$\bar{x}_4$	—	—	—	0	q_4	1	0	0	0	0	0	0	0	0	1
q_5	1	0	1	1	—	—	—	—	q_0	0	0	0	0	1	0	0	0	1	0
q_6	$1_{(1)}$	$1_{(1)}$	$0_{(1)}$	$x_1\bar{x}_4$	1	—	—	0	q_2	0	1	0	0	1	0	0	0	1	0
q_6	$1_{(1)}$	$1_{(1)}$	$0_{(1)}$	$x_1x_2\bar{x}_4$	1	1	—	0	q_3	0	1	1	0	1	0	0	1	0	0

4. Построение логической схемы автомата.

Полученные зависимости позволяют построить логическую схему автомата Мура в базе И—ИЛИ—НЕ. Если заданы другие элементы, то необходимо осуществить соответствующие преобразования и перейти в заданную систему элементов (рис. 12.11).

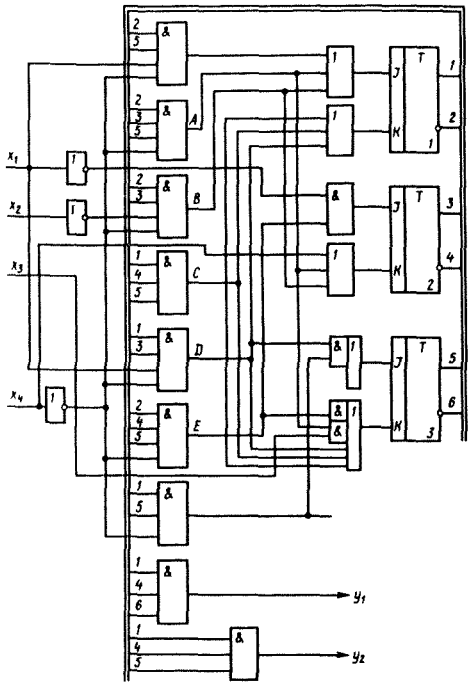


Рис. 12.11. Логическая схема электронного замка

Синтез автомата МИЛИ

Состояния выходов автомата Мили зависят как от состояния автомата, так и от входных сигналов. В соответствии с этим выполняем отметку граф-схемы алгоритма по следующим правилам:

— символом q_0 отмечаем вход вершины, следующей за начальной, и вход конечной вершины;

— символами q_0 ($i \neq 0$) отмечаем входы всех вершин, следующих за операторными;

— входы различных вершин (кроме конечной) отмечаются разными символами.

Составляем граф переходов автомата Мили (рис. 12.12). Состояние q_5 аналогично состоянию q_6 автомата Мура. Из графа видно, что выходной сигнал y_1 (оператор v_4), вырабатываемый в состоянии q_4, зависит только от состояния q_4, а сигнал y_2 (оператор v_5) зависит как от состояния q_0, так и от входных сигналов. Число триггеров $N_1 = 3$, кодирование то же, что и в автомате Мура ($q_0 = 000$; ...; $q_5 = 101 \vee 110 \vee 111$).

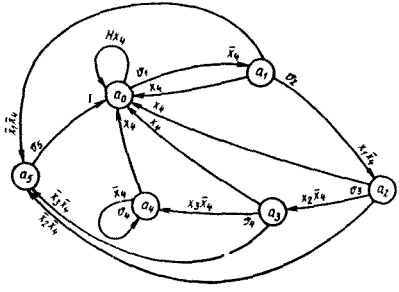


Рис. 12.12. Граф переходов автомата Мили

По таблице 12.18 записываем основные уравнения для всех параметров.

Уравнения автомата Мили:

$$J_1 = \bar{Q}_1 Q_2 Q_3 x_3 \bar{x}_4 = E \cdot Q_2 x_3;$$

Таблица 12.18

Текущее состояние t				Условие перехода	Входные сигналы комбинационной схемы				Последующее состояние $t+1$				Выходные сигналы комбинационной схемы						Выходной сигнал
имя	код								имя	код									
	Q_1	Q_2	Q_3		x_1	x_2	x_3	x_4		Q_1	Q_2	Q_3	J_1	K_1	J_2	K_2	J_3	K_3	f_2
Все	—	—	—	x_4	—	—	—	1	q_0	0	0	0	0	1	0	1	0	1	0
q_0	0	0	0	$\bar{x}_4$	—	—	—	0	q_1	0	0	1	0	0	0	0	1	0	0
q_1	0	0	1	$\bar{x}_1\bar{x}_4$	0	—	—	0	q_0	0	0	0	0	0	0	0	0	1	1
q_1	0	0	1	$x_1\bar{x}_4$	1	—	—	0	q_2	0	1	0	0	1	0	0	0	1	0
q_2	0	1	0	$\bar{x}_2\bar{x}_4$	—	0	—	0	q_0	0	0	0	0	0	0	1	0	0	1
q_2	0	1	0	$x_2\bar{x}_4$	—	1	—	0	q_3	0	1	1	0	0	0	0	1	0	0
q_3	0	1	1	$\bar{x}_3\bar{x}_4$	—	—	0	0	q_0	0	0	0	0	0	0	1	0	1	1
q_3	0	1	1	$x_3\bar{x}_4$	—	—	1	0	q_4	1	0	0	0	0	0	1	0	1	0
q_4	1	0	0	$\bar{x}_4$	—	—	—	0	q_4	1	0	0	0	0	0	0	0	0	0
q_5	1	0	1	$x_1\bar{x}_4$	1	—	—	0	q_2	0	1	0	0	1	1	0	0	1	0
q_5	1	0	1	$x_1x_2\bar{x}_4$	1	1	—	0	q_3	0	1	1	0	1	1	0	1	0	0

$$K_1 = x_4 + (Q_1\overline{Q}_2Q_3 + Q_1Q_2\overline{Q}_3 + Q_1Q_2Q_3)x_1\bar{x}_4 + (Q_1\overline{Q}_2Q_3 + Q_1Q_2\overline{Q}_3 +$$
$$+ Q_1Q_2Q_3)x_1x_2\bar{x}_4 = x_4(Q_1Q_2 + Q_1Q_3)x_1\bar{x}_4 = x_4 + \underbrace{Q_1Q_2x_1\bar{x}_4}_{B} + \underbrace{Q_1Q_3x_1\bar{x}_4}_{C};$$

$$J_2 = \overline{Q}_1\overline{Q}_2Q_3x_1\bar{x}_4 + Q_1Q_2x_1\bar{x}_4 + Q_1Q_3x_1\bar{x}_4 =$$
$$= Q_1Q_2x_1\bar{x}_4 + \underbrace{Q_1Q_3x_1\bar{x}_4}_{H} + \overline{Q}_2\underbrace{Q_3x_1\bar{x}_4}_{H} = B + C + \overline{Q}_2 \cdot H;$$

$$K_2 = x_4 + \overline{Q}_1Q_2\overline{Q}_3\bar{x}_2\bar{x}_1 + \overline{Q}_1Q_2Q_3\bar{x}_1\bar{x}_4 + \overline{Q}_1Q_2Q_3x_2\bar{x}_4 =$$
$$= x_4 + \overline{Q}_1Q_2Q_3\bar{x}_4 + \underbrace{\overline{Q}_1Q_2\bar{x}_2\bar{x}_4}_{D} = x_4 + \overline{Q}_1Q_2 \cdot E + D;$$

$$J_3 = \overline{Q}_1\overline{Q}_2\overline{Q}_3\bar{x}_4 + \overline{Q}_1Q_2\overline{Q}_3x_2\bar{x}_4 + (Q_1Q_2 + Q_1Q_3)x_1x_2\bar{x}_4 =$$
$$= \overline{Q}_1\overline{Q}_2\overline{Q}_3\bar{x}_4 + \overline{Q}_1\overline{Q}_3x_2\bar{x}_4 + Q_1Q_2x_1x_2\bar{x}_4 + Q_1Q_2x_1x_2\bar{x}_4 + Q_1Q_3x_1x_2\bar{x}_4 =$$
$$= B \cdot x_2 + C \cdot x_2 + \overline{Q}_1\overline{Q}_3\overline{Q}_3\bar{x}_4 + \overline{Q}_1\overline{Q}_3x_2\bar{x}_4;$$

$$K_3 = x_4 + \overline{Q}_1\overline{Q}_2Q_3\bar{x}_1\bar{x}_4 + \overline{Q}_1\overline{Q}_2Q_3x_1\bar{x}_4 + Q_1Q_2Q_3(x_1 + \bar{x}_1)\bar{x}_4 + Q_1Q_2x_1\bar{x}_4 +$$
$$+ Q_1Q_3x_1\bar{x}_4 = x_4 + \overline{Q}_1Q_3\bar{x}_1 + Q_3\bar{x}_1\bar{x}_4 + Q_1Q_2x_1\bar{x}_4 = x_4 + E + H + B;$$
$$f_2 = \overline{Q}_1\overline{Q}_2Q_3\bar{x}_1\bar{x}_4 + \overline{Q}_1Q_2\overline{Q}_3\bar{x}_2\bar{x}_4 + \overline{Q}_1Q_2Q_3\bar{x}_1\bar{x}_4 =$$
$$= \overline{Q}_1\overline{Q}_2Q_3\bar{x}_1\bar{x}_4 + D \cdot \overline{Q}_3 + Q_2\bar{x}_3 \cdot E.$$

Логическая схема автомата составляется так же, как и в случае автомата Мура.

Пример 12.1. Провести синтез автомата, управляющего работой n-разрядного сумматора (рис. 12.13).

Р е ш е н и е. Проектирование осуществляется в несколько этапов:

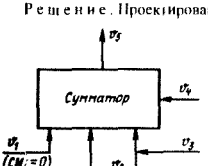


Рис. 12.13. Схема сумматора, для которой создается управляющий автомат

1) разработка алгоритма операции;

2) построение таблицы переходов–выходов микропрограммного автомата Мили (или Мура) для разработанного алгоритма выполнения операции;

3) кодирование внутренних состояний (заданным способом);

4) построение таблицы функций логических преобразований микропрограммного автомата (ЛП ПМА) для заданного типа триггеров, реализующих автомат;

5) минимизация логических функций;

6) построение логической схемы автомата в заданном базисе.

Итак, наш сумматор имеет две входные n-разрядные шины для приема чисел B и A, n-разрядную шину для выдачи результата.

На сумматор (рис. 12.14) подаются сигналы: v_1 — установка сумматора в $\varnothing$, v_2 — прием и поразрядное сложение A с содержимым сумматора; v_3 — прием и поразрядное сложение B с содержимым сумматора; v_4 — выработка переносов и сложение их с содержимым сумматора; v_5 — выдача результата. С другой стороны, сумматор выполняет определенные операции, которые должны быть реализованы в виде микропрограммы:

Операция	Микропрограмма	Условие
Сброс СМ	$MП_0 = H \cdot v_1 \; v_k$	z_0
Загрузка А	$MП_1 = H \; v_1 \cdot v_k$	z_1
Загрузка В	$MП_2 = H \cdot v_1 \; v_1 \cdot v_k$	z_2
$C = A \oplus B$	$MП_3 = H \cdot v_1 \cdot v_2 \cdot v_3 \cdot v_k$	z_3
$C = A + B$	$MП_4 = H \cdot v_1 \cdot v_2 \cdot v_3 \cdot v_4 \cdot v_k$	z_4
Результат	$MП_5 = H \cdot v_5 \cdot v_k$	z_5

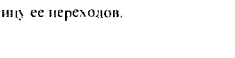
Здесь H — начальное логическое условие, инициирующее выполнение заданной операции (при $H = 1$), v_k — пустая микрооперация, сигнализирующая внешнему устройству, что операция завершилась.

На основе этой микропрограммы можно построить граф-схему микропрограммы (см. рис. 12.14). После этого проведем граф-схемы по следующим правилам:

Для автомата Мили.

Этап 1. Отметим входы вершин, следующих за операторными вершинами, причем: а) вход вершины, следующей за начальной, так же как и вход вершины, a_1, б) входы всех вершин, следующих за операторными, должны быть отмечены, в) если вход отмечается, то только одним символом, г) входы разных вершин, кроме конечной, отмечаются различными символами.

Этап 2. а) если идти от отметки a_m к a_n в направлении ориентации дуг ГСА, то каждую из них пометим следующим образом:



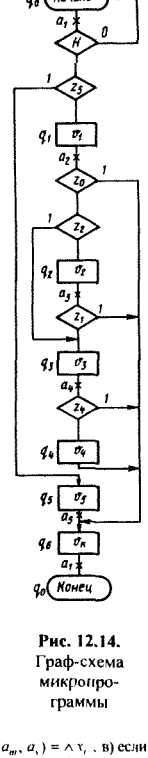


Рис. 12.14. Граф-схема микропрограммы

б) каждому пути (слову) можно поставить в соответствие конъюнкцию $x(a_m, a_s) = \wedge_i x_i$, в) если путей несколько, то ставим в соответствие дизъюнкцию $x(a_m, a_s) = \vee_n x_m$, а в случае пустого множества переменных — символ 1; г) выходной сигнал $\bar{v}_0$ (пустой оператор) ставим в соответствие пути $a_m \to a_1$. В итоге получаем граф автомата Мили (рис. 12.15).

Составим таблицу ее переходов.

Таблица 12.19

a_m (исходное состояние)	a_s (состояние перехода)	$x(a_m, a_s)$ (входной сигнал)	$y(a_m, a_s)$ (выходной сигнал)
a_1	a_1	H	y_0
	a_2	$H\ z_5$	y_1
	a_5	$H\ z_5$	y_5
a_2	a_3	$z_0\ z_2$	y_2
	a_4	$z_0\ z_2$	y_3
	a_5	z_0	y_0
a_3	a_4	z_1	y_3
	a_5	z_1	y_0
a_4	a_5	z_4	y_0
	a_5	z_4	y_4
a_5	a_1	1	y_4

Для автомата Мура

Этап 1. а) символом q_0 отметим начальную и конечную вершины. б) в отличие от автомата Мили отмечаем не входы вершин, а сами операторные вершины. в) различные операторные вершины должны быть помечены разными символами; г) помечены должны быть все операторные вершины

Рис. 12.15. Граф переходов для автомата Мили

Этап 2. Строим граф автомата Мура (рис. 12.16). Ему соответствует следующая таблица переходов (табл. 12.20)

Теперь воспользуемся каноническим методом структурного синтеза, используя D-триггеры (рис. 12.17). Сколько их понадобится? Чтобы закодировать семь состояний, хватит трех триггеров

Таблица 12.20

Исходное состояние	Состояние перехода	Условие перехода	Состояние триггера		
			Q_1	Q_2	Q_3
q_0	q_0	$\overline{H}$	0	0	0
	q_1	$H \cdot \bar{z}_5$	0	0	1
	q_5	$H \cdot z_5$	1	0	1
q_1	q_2	$\bar{z}_0 \cdot \bar{z}_2$	0	1	0
	q_3	$\bar{z}_0 \cdot z_2$	0	1	1
	q_0	z_0	1	1	0
q_2	q_3	$\bar{z}_1$	0	1	1
	q_0	z_1	1	1	0
q_3	q_4	$\bar{z}_4$	1	0	0
	q_0	z_4	1	1	0
q_4	q_0	1	1	1	0
q_5	q_0	1	1	1	0
q_6	q_0	0	0	0	0

Если в качестве памяти использовать D-триггеры, то логические уравнения для сигналов на входах триггеров имеют вид

$$\begin{cases} D_1 = q_0 H z_5 + q_1 \bar{z}_0 z_2 + q_2 \bar{z}_1 + q_0 H \bar{z}_5; \\ D_2 = q_1 \bar{z}_0 \bar{z}_2 + q_1 \bar{z}_0 z_2 + q_1 z_0 + q_2 \bar{z}_1 + \\ q_3 \bar{z}_4 + q_3 z_4 + q_4 + q_5; \\ D_3 = q_0 H z_5 + q_1 z_0 + q_1 z_1 + q_3 \bar{z}_4 + \\ + q_3 z_4 + q_4 + q_5. \end{cases}$$

После минимизации преобразуем эти уравнения, используя дистрибутивный закон

$$D_1 = q_0 H + q_1 \bar{z}_0 z_2 + q_2 \bar{z}_1;$$

$$D_2 = q_1 \bar{z}_0 + q_1 z_0 + q_3 z_4 + q_4 + q_5 =$$

$$= q_1 + q_3 + q_4 z_4 + q_4 + q_5;$$

$$D_3 = q_0 H z_5 + q_1 z_0 + q_2 z_1 + q_3 + q_4 + q_5.$$

Полученные уравнения можно реализовать с помощью логической схемы в базисе И, ИЛИ, НЕ и D-триггеров (рис. 12.17).

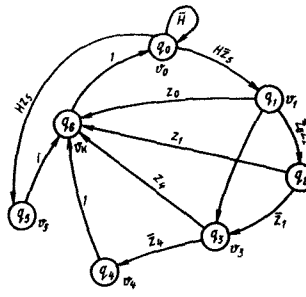


Рис. 12.16. Граф-переходов для автомата Мура

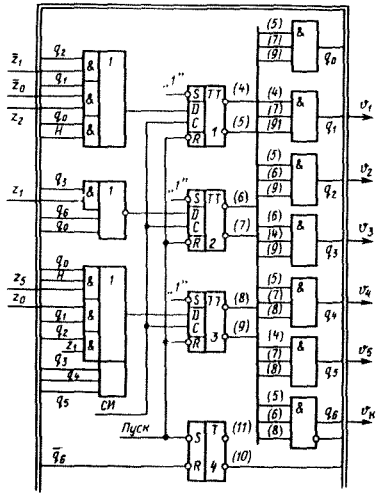


Рис. 12.17. Логическая схема управляющего автомата Мура

Ответ Схема синтезированного автомата представлена на рис. 12.17

Задание для самоконтроля

1. Почему необходимо задавать начальное состояние автомата?
2. В чем состоит отличие автомата Мура от автомата Мили?
3. Что такое программа автомата?
4. Чем отличается операционный автомат от управляющего автомата?
5. Чем отличается граф автомата от содержательного графа автомата?
6. Что такое разметка граф-схемы автомата?

Список литературы

1 *Гаврилов Г. В.* Теория алгоритмов. — М.: Статистика, 1973.

2 *Балашов Е. П., Пузанков Д. В.* Микропроцессоры и микропроцессорные системы. — М.: Энергия, 1981.

3 *Баранов С. И.* Синтез микропрограммных автоматов. — М.: Энергия, 1974.

4 *Гаврилов М. А.* Теория релейных и конечных автоматов. — М.: Наука, 1983.

5 *Гивоне Д., Рессер Р.* Микропроцессоры и микрокомпьютеры /Пер. с англ. — М.: Мир, 1983.

6 *Глушков В. М.* Синтез цифровых автоматов. — М.: Наука, 1962.

7 *Калан Б. М.* Электронные вычислительные машины и системы. — М.: Энергоатомиздат, 1985.

8 *Лорьер Ж.-Л.* Системы искусственного интеллекта. — М.: Мир, 1991.

9 Микропроцессоры. В 3 кн. /Под ред. *Л. Н. Преснухина.* — М.: Высшая школа, 1984.

10 *Дж. фон-Нейман.* Теория самонастраивающихся автоматов. — М.: Мир, 1971.

11 *Питерсон У., Уэлдон Э.* Коды, исправляющие ошибки /Пер. с англ. — М.: Мир, 1976.

12 *Поспелов Д. А.* Арифметические основы вычислительных машин дискретного действия. — М.: Энергия, 1970.

13 *Савельев А. Я.* Арифметические и логические основы цифровых автоматов. — М.: Высшая школа, 1980.

14 *Савельев А. Я., Сазонов Б. А., Лукьянов С. Э.* Персональный компьютер для всех: в 4 кн. — М.: Высшая школа, 1991.

15 *Соломатин Н. М.* Информационные семантические системы. — М.: Высшая школа, 1989.

16 *Сяо Д., Керр Д., Мэдник С.* Защита ЭВМ. — М.: Мир, 1982.

17 *Темников Ф. Е., Афонин В. А., Дмитриев В. И.* Теоретические основы информационной техники. — М.: Высшая школа, 1979.

18 *Уокерли Д.* Архитектура и программирование микроЭВМ /Пер. с англ. — М.: Мир, 1984.

19 *Фистер М.* Логическое проектирование цифровых вычислительных машин. Киев, 1964.

20 *Хоровиц П., Хилл У.* Искусство схемотехники: в 2 т. — М.: Мир, 1983.

21 *Хоффман Л. Д.* Современные методы защиты информации. — М.: Советское радио, 1980.

22 *Шоломов Л. А.* Основы теории дискретных логических и вычислительных устройств. — М., 1980.

23 Электронные вычислительные машины: в 8 кн. /Под ред. *Савельева А. Я.* — М.: Высшая школа, 1987.

24 Энциклопедия кибернетики. Т. 1, 2. Киев, 1975.

25 *Яблонский С. В.* Функции алгебры логики и классы Поста. — М.: Наука, 1966.