Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

**Ю. А. Луцик, И. В. Лукьянова**

# АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

*Допущено Министерством образования
Республики Беларусь в качестве учебного пособия
для студентов учреждений высшего образования
по специальности «Вычислительные машины,
системы и сети»*

Минск БГУИР 2014

УДК 004.31 (075)
ББК 32.973.26-04я7
Л86

Р е ц е н з е н т ы :
кафедра математической кибернетики
Белорусского государственного университета
(протокол №1 от 30.08.2013);

заведующий кафедрой информационных систем и технологий учреждения образования «Белорусский государственный технологический университет», доктор технических наук, профессор П. П. Урбанович;

кафедра информатики и компьютерных систем учреждения образования «Белорусский государственный университет»
(протокол №6 от 11.01.2013);

декан факультета инновационной подготовки Института управленческих кадров Академии управления при Президенте Республики Беларусь, кандидат технических наук, доцент В. В. Лабоцкий;

кафедра программного обеспечения информационных технологий учреждения образования «Белорусский государственный университет информатики и радиоэлектроники» (протокол №21 от 01.04.2013)

**Луцик, Ю.А.**

Л86 Арифметические и логические основы вычислительной техники : −Учеб. пособие / Ю. А. Луцик, И. В. Лукьянова. – Минск : БГУИР, 2014. − 165с. : ил. 83.
ISBN 978-985-543-032-3.

Учебное пособие посвящено описанию способов представления числовой информации в ЭВМ, методов выполнения арифметических и логических операций в вычислительных машинах. Рассмотрены вопросы, связанные со способами контроля правильности функционирования вычислительного устройства и методами оптимизации устройств, выполняющих арифметические операции.

Пособие может быть использовано студентами всех форм обучения, магстрантами и аспирантами специальности «Вычислительные машины, системы и сети».

**УДК 004.31 (075)**
**ББК 32.973.26-04я7**

**ISBN 978-985-543-032-3**

**Введение**

В современном мире для подавляющего большинства людей актуальным, а для многих и необходимым стало умение пользоваться современными информационными технологиями, использовать компьютер в различных сферах деятельности. Более того, культура общения с компьютером становится частью общей культуры человека.

Предмет «Арифметические и логические основы вычислительных машин» является одним из первых специальных курсов, который формирует у студентов понимание основополагающих вопросов организации ЭВМ, принципы построения отдельных составных частей ЭВМ, их взаимосвязь. Этот предмет должен сформировать начальные знания для изучения последующих спецдисциплин, таких как «Схемотехника», «Периферийные устройства» и др., познакомить студентов с задачами разработки алгоритмов функционирования устройств ЭВМ.

Основная цель настоящего учебного пособия – помочь студенту, приступившему к изучению арифметики вычислительных машин, приобрести теоретические знания и практические навыки выполнения основных арифметических операций. Правильное понимание алгоритмов рассматриваемых операций подкрепляется знанием структурных и логических схем, реализующих эти алгоритмы и представляющих собой некоторые операционные устройства. В пособии уделяется внимание рассмотрению этих схемных решений. Подробно рассмотрен аппарат, основанный на правилах и законах булевой алгебры, ориентированный на упрощение (минимизацию) проектируемых логических схем. Кроме того, в пособии приводятся сведения об основных формах хранения и преобразования числовой информации, способах ее кодирования. Достаточное внимание уделено методам контроля правильности функционирования цифрового автомата, возможным ошибкам, возникающим при его работе, и способам их устранения. В заключительном разделе учебного пособия рассматриваются элементы теории конечных автоматов и их использование для построения управляющих устройств.

Рассматриваемый в пособии теоретический материал сопровождается большим количеством примеров, что упрощает и делает более понятным излагаемый материал.

Следует отметить, что в течение последних лет литература, освещающая арифметику вычислительных машин, не выпускалась. В пособии сделана попытка устранить этот информационный пробел. Материал базируется на работах, приведенных в списке литературы в конце учебного пособия.

Таким образом, основной задачей, решению которой посвящено пособие, является рассмотрение вопросов, которые помогут понять суть проблем, стоящих перед разработчиками при проектировании отдельных устройств ЭВМ, при разработке алгоритмов их функционирования и применении полученных в процессе обучения знаний. Это обеспечит возможность выполнения самостоятельного синтеза простых узлов и блоков дискретных устройств.

## 1. Информационные основы вычислительных машин

### 1.1. Основные понятия информатики

Определим некоторые основополагающие понятия, которые будем использовать при изложении основных вопросов курса. Одно из первых – определение самой **вычислительной машины**. В энциклопедии кибернетики она определена как «физическая система (устройство или комплекс устройств), предназначенная для механизации или автоматизации процесса алгоритмической обработки **информации** и вычислений».

**Информация** является объектом передачи и преобразования в вычислительных системах (и машинах). В этой связи о вычислительных машинах и системах говорят как об информационных системах. Сам термин «информация» происходит от латинского слова informatio – разъяснение, пояснение. Особенностью понятия «информация» является его универсальность. Оно используется во всех без исключения сферах человеческой деятельности.

Информация – категория нематериальная, для ее существования (передачи, обработки) необходимо связать ее с каким-то материальным носителем. Информация содержится в человеческой речи, текстах книг, журналов, газет, сообщениях радио и телевидения, показаниях приборов и т. д.

Изменение содержания информации с течением времени называется информационным процессом. Примерами информационных процессов являются создание новой информации, ее преобразование, уничтожение и передача.

Понятие «информация» обычно предполагает наличие двух объектов – источника и приемника информации. Информация, воплощенная и зафиксированная в материальной форме, называется сообщением. Сообщения могут быть непрерывными и дискретными. Непрерывное (аналоговое) сообщение представляется физической величиной (электрическим напряжением, током и т. д.), изменение которой во времени отображает протекание рассматриваемого процесса. Дискретная информация передается от источника к приемнику в форме сигналов, распространяющихся в определенной среде, при этом для передачи информации используют различные знаки или символы, например, естественного или искусственного (формального) языка, позволяющие выразить ее в форме сообщения.

Если говорить об информационных процессах в вычислительной машине, то все они непосредственно связаны с различными физическими носителями информационных сообщений, и все блоки и устройства являются физической средой, в которой и происходят эти информационные процессы.

Процесс решения задачи на вычислительной машине проходит через следующие этапы: ввод исходных данных в компьютер, переработка этой информации согласно заданному алгоритму, вывод результатов решения задачи (переработки информации).

Таким образом, в узком смысле этого слова информацию можно определить как *любые сведения, являющиеся объектом хранения, передачи и преобразования* [2].

Важным вопросом в изучении теории информации является установление меры, количества и качества информации. Эти меры, как правило, рассматривают в трех аспектах – структурном, статистическом и семантическом.

В **структурном** аспекте рассматривается измерение массивов информации путем простого подсчета информационных элементов или комбинаторным методом, без учета конкретных условий применения информационных систем.

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

**Семантический** подход позволяет выделить полезность или ценность информационного сообщения.

### 1.2. Структурная мера информации

В структурном аспекте информация представляется в виде сообщения, единицей которого является символ. Символы, собранные в группы, называются словами. При использовании структурной меры информации учитывается только дискретное строение сообщения, количество содержащихся в нем элементов информации, связей между ними. При структурном подходе можно определить геометрическую, комбинаторную и аддитивную меры информации.

Наиболее часто в нашем курсе будем использовать аддитивную меру информации (мера Хартли), при которой количество информации измеряется в двоичных единицах – битах. Здесь вводятся понятия глубины $q$ и длины $n$ числа.

*Глубина числа q – количество символов, принятых для представления информации.*

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

При заданной глубине и длине числа количество чисел, которое можно представить, $N = q^n$. Один бит информации соответствует одному элементарному событию, которое может произойти или не произойти.

В компьютерной технике наименьшей единицей измерения информации является 1 бит. Таким образом, объем информации, записанной двоичными знаками (0 и 1) в памяти компьютера или на внешнем носителе информации подсчитывается просто по количеству требуемых для такой записи двоичных символов. Например, восьмиразрядный двоичный код 11001011 имеет объем данных, равный 8 бит. Кроме минимальной единицы измерения данных «бит» широко используется единица измерения «байт», равная 8 бит. При работе с большими объемами информации для подсчета ее количества применяют более крупные единицы измерения, такие как килобайт (Кбайт), мегабайт (Мбайт), гигабайт (Гбайт), терабайт (Тбайт), петабайт (Пбайт) и т. д.:

1 Кбайт = 1024 байт = $2^{10}$ байт;

1 Мбайт = 1024 Кбайт = $2^{20}$ байт = 1 048 576 байт;

1 Гбайт = 1024 Мбайт = $2^{30}$ байт = 1 073 741 824 байт;

1 Тбайт = 1024 Гбайт = $2^{40}$ байт = 1 099 511 627 776 байт;

1 Пбайт = 1024 Тбайт = $2^{50}$ байт = 1 125 899 906 842 624 байт.

Следует обратить внимание, что в системе измерения двоичной (компьютерной) информации, в отличие от метрической системы, единицы с приставками «кило», «мега» и т. д. получаются путем умножения основной единицы не на $10^3 = 1000$, $10^6 = 1000\,000$ и т. д., а на $2^{10} = 1024$, $2^{20}$ и т. д.

### 1.3. Статистическая мера информации

Статистическая мера информации применяется в том случае, когда приходится иметь дело с явлениями, исход которых неоднозначен и зависит от факторов, которые мы не знаем или не можем учесть. Например, определение пола будущего ребенка, результат бросания игральной кости и пр. События, о которых нельзя сказать произойдут они или нет, пока не будет осуществлен эксперимент (опыт), называются случайными. Изучением таких событий занимается раздел математики, называемый теорией вероятности. Когда мы имеем дело со случайными событиями, имеется некоторая неопределенность.

Осуществление некоторого комплекса условий называется опытом, а интересующий нас исход этого опыта – благоприятным событием. Тогда, если $N$ – общее число опытов, а $N_A$ – количество благоприятных исходов случайного события $A$, то отношение $N/N_A$, называется относительной частотой появления события $A$. В случае, если все исходы опыта конечны и равновозможны, то их вероятность равна $P = 1/n$, где $n$ – число возможных исходов. Например, вероятность выпадения орла при бросании монеты – 1/2, вероятность вытянуть из урны красный шар (при условии, что там три шара – красный, синий, белый) – 1/3.

Численная величина, измеряющая неопределенность опыта, называется энтропией [1]. Обозначим ее $H$. Очевидно, что величины $H$ и $n$ (число возможных исходов опыта) связаны функциональной зависимостью: $H = f(n)$, т. е. мера неопределенности есть функция числа исходов.

Из определения следует, что энтропия – это числовая характеристика, отражающая ту степень неопределенности, которая исчезает после проведения опыта, в результате которого получаем определенную информацию. Энтропия опыта равна той информации, которую мы получаем в результате его осуществления. То есть информация $I$ – это содержание сообщения, понижающего неопределенность некоторого опыта с неоднозначным исходом; убыль связанной с ним энтропии является количественной мерой информации.

Значит, если $H1$ – начальная энтропия (до проведения опыта), $H2$ – энтропия после проведения опыта, то информация

$I = H1 - H2 = log_2 n_1 - log_2 n_2 = log_2 (n_1/n_2)$.

Значение $H$ будет равно единице при $n = 2$. В качестве единицы принимается количество информации, связанное с проведением опыта, состоящего в получении одного из двух равновероятных исходов (например, бросание монеты). Такая единица количества информации называется «бит».

### 1.4. Семантическая мера информации

Информация, с которой имеют дело современные вычислительные машины, несет самое разное содержание, это может быть числовая, текстовая, графическая информация. Оценка содержания разнохарактерной информации очень сложна. К семантическим мерам информации можно отнести, например, содержательность, логическое количество, целесообразность и существенность информации.

### 1.5. Электронные цифровые вычислительные машины

Вид информации, которая подлежит обработке, влияет на структуру вычислительных машин. Устройства, которые используются для обработки дискретной информации, называются цифровыми или дискретными устройствами. Цифровые вычислительные машины (компьютеры) относятся к этому классу устройств.

Одной из основных задач курса является изучение основных принципов организации и построения алгоритмов функционирования цифровых вычислительных машин. История их возникновения и развития уходит совсем недалеко в прошлое. Первые такие устройства появились в середине прошлого века. Развитие вычислительной техники за прошедшие полвека продвинулось очень значительно, часто менялись технические детали устройства ЭВМ, однако фундаментальные принципы построения ЭВМ, напротив, используются в течение очень длительного времени. Джон фон Нейман с соавторами в 1946 году выдвинули основные принципы логического устройства ЭВМ, заложившие основы развития вычислительной техники на несколько десятилетий вперед, и предложили ее архитектуру, которая называется «фон-неймановской». Эта архитектура полностью воспроизводилась в течение первых двух поколений ЭВМ.

Вот основные рекомендации, предложенные фон Нейманом для разработчиков ЭВМ [3]:

1. Машины на электронных элементах должны работать не в десятичной, а в двоичной системе счисления.
2. Программа должна размещаться в одном из блоков машины − в *запоминающем устройстве* (ЗУ), обладающем достаточной емкостью и соответствующими скоростями выборки и записи команд программы.
3. Программа так же, как и числа, которыми оперирует машина, представляется в двоичном коде. Таким образом, по форме представления команды и числа однотипны. Это обстоятельство приводит к следующим важным последствиям:
   - промежуточные результаты вычислений, константы и другие числа могут размещаться в том же ЗУ, что и программа;
   - числовая форма записи программы позволяет машине производить операции над величинами, которыми закодированы команды программы.
4. Трудности физической реализации ЗУ, быстродействие которого со-

ответствовало бы скорости работы логических схем, требует иерархической организации памяти.

5. Арифметические устройства машины конструируются на основе схем, выполняющих операцию сложения. Создание специальных устройств для вычисления других операций нецелесообразно.

6. В машине используется параллельный принцип организации вычислительного процесса (операции над словами производятся одновременно по всем разрядам).

ЭВМ, построенная по принципам, определенным фон Нейманом, состоит из следующих основных блоков (рис. 1): запоминающего устройства (ЗУ), арифметико-логического устройства (АЛУ), устройства управления (УУ) и устройства ввода − вывода.


Рис. 1. Структура классической ЭВМ

Компьютеры, построенные в соответствии с этими рекомендациями, относят к типу фон-неймановских.

Все действия в ЭВМ выполняются под управлением сигналов, вырабатываемых УУ. Управляющие сигналы формируются на основе информации, содержащейся в выполняемой команде, и признаков результата, сформированных предыдущей командой (если выполняемая команда является, например, командой условного перехода). УУ помимо сигналов, определяющих те или иные действия в различных блоках ЭВМ (например, вид операции в АЛУ или сигнал считывания из ЗУ), формирует также адреса ячеек, по которым производится обращение к памяти для считывания команды, и операндов и записи результата выполнения команды.

УУ формирует адрес команды, которая должна быть выполнена в данном цикле, и выдает управляющий сигнал на чтение содержимого соответствующей ячейки ЗУ. Считанная команда передается в УУ. По информации, содержащей-

ся в адресных полях команды, УУ формирует адреса операндов и управляющие сигналы для их чтения из ЗУ и передачи в АЛУ. После считывания операндов УУ по коду операции, содержащемуся в команде, выдает в АЛУ сигналы на выполнение операции. Полученный результат записывается в ЗУ по адресу приемника результата под управлением сигналов записи. Признаки результата (знак, наличие переполнения, признак нуля и т. д.) поступают в УУ, где записываются в специальный регистр признаков. Эта информация может использоваться при выполнении следующих команд программы, например команд условного перехода.

В современных вычислительных устройствах основным исполнительным элементом является процессор или микропроцессор, который содержит в себе АЛУ, память, блок управления.

Еще одно базовое понятие курса – понятие **цифрового автомата**. Академик В. М. Глушков дает следующее определение цифрового автомата [12]: «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**». Таким образом, саму вычислительную машину (в настоящее время чаще используется термин компьютер) можно рассматривать как частный случай цифрового автомата. Понятие автомата можно использовать как модель для описания работы устройств, обрабатывающих цифровую информацию. Для формального описания цифрового автомата применяется аппарат алгебры логики. Эти вопросы также входят в число основных тем курса «Арифметические и логические основы вычислительной техники».

**Контрольные вопросы и задания**

1. Дайте определение вычислительной машины.
2. Определите понятие информации.
3. Перечислите, какие используются меры информации.
4. Назовите единицы измерения информации.
5. Перечислите принципы фон Неймана.
6. Перечислите основные блоки, составляюшие вычислительную машину согласно архитектуре фон Неймана.
7. Каковы основные функции АЛУ?
8. Для чего предназначено устройство управления?
9. Что хранится в запоминающем устройстве?
10. Что такое дискретное устройство?
11. Что такое цифровой автомат?

## 2. Арифметические основы вычислительной техники

### 2.1. Системы счисления

В ЭВМ информация представляется в виде чисел, записанных в той или иной системе счисления. Выбор системы счисления – один из важнейших вопросов. От правильности его решения зависят такие характеристики ЭВМ, как скорость вычислений, сложность алгоритмов реализации арифметических операций и др. Система счисления – совокупность цифр, приемов и правил для записи чисел цифровыми знаками.

Любая система счисления должна обеспечивать:

- возможность представления любого числа в рассматриваемом диапазоне величин;
- единственность этого представления;
- простоту выполнения операций над числами.

Используемые системы счисления можно разделить на три категории:

1) позиционные системы счисления;
2) непозиционные системы счисления;
3) смешанные системы счисления.

*Непозиционная* система счисления – система, для которой значение символа не зависит от его положения в числе. Примером может служить система счисления с одной цифрой «1». Для записи любого числа в ней необходимо написать количество единиц, равное числу. Другой пример – это римская система счисления.

*Смешанная* система зачастую относится к позиционным системам счисления, каждое число в ней представляется как линейная комбинация. Записью числа в смешанной системе счисления называется перечисление его цифр в порядке уменьшения индекса, начиная с первого ненулевого.

Наиболее известным примером смешанной системы счисления является представление времени в виде количества суток, часов, минут и секунд.

*Позиционной* системой счисления называется система записи любых по величине чисел, в которой значение цифры зависит от ее положения в числе, т. е. *веса*. Число цифр в позиционной системе счисления ограничено. Позиционная система счисления имеет алфавит и основание.

Совокупность различных цифр (символов), используемых в позиционной системе счисления для записи чисел, называется алфавитом системы счисления. Ниже приведены алфавиты некоторых позиционных систем счисления:

- десятичная система: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
- двоичная система: {0, 1};
- восьмеричная система: {0, 1, 2, 3, 4, 5, 6, 7};
- шестнадцатеричная система: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F} .

*Основание r* позиционной системы счисления – максимальное количество различных знаков или символов, используемых для изображения числа в данной системе счисления. Таким образом, основание может быть любым числом, кроме единицы и бесконечности. Последовательность степеней основания называют *базисом* позиционной системы счисления, каждая из которых задает

количественное значение или «вес» каждого разряда. Позиция цифры в числе называется *разрядом*. Ниже приведены базисы некоторых позиционных систем счисления:

- десятичная система: $10^0$, $10^1$, $10^2$, $10^3$, $10^4$, ..., $10^n$;
- двоичная система: $2^0$, $2^1$, $2^2$, $2^3$, $2^4$, ..., $2^n$;
- восьмеричная система: $8^0$, $8^1$, $8^2$, $8^3$, $8^4$, ..., $8^n$.

В общем случае любое число в системе счисления с основанием $r$ может быть записано в общем виде:

$$A = a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + ... + a_1 \cdot r^1 + a_0 \cdot r^0 + a_{-1} \cdot r^{-1} + ... + a_{-(m-1)} \cdot r^{-(m-1)} + a_{-m} \cdot r^{-m}, \quad (1)$$

или

$$A_i = \sum_{i=-m}^{n} a_i r^i , \quad (2)$$

где любая разрядная цифра $a_i \in \{0, \ldots, r-1\}$, а каждая степень $r^i$ в такой записи называется весовым коэффициентом разряда; $n + 1$ и $m$ соответственно число разрядов целой и дробной части числа.

Старшинство разрядов и соответствующих им цифр определяется значением показателя $i$ (номером разряда). Запись числа в форме (1) назовем записью числа в развернутой форме. Свернутой формой записи чисел называется запись чисел в виде последовательности его $r$-ичных цифр, перечисляемых по возрастанию старшинства разрядов слева направо:

$A = a_n a_{n-1} \ldots a_0 \ldots a_{-(m-1)} a_{-m}$.

Например, число 214,13 представляется в развернутом виде в десятичной системе счисления:

$214{,}13_{10} = 2 \cdot 10^2 + 1 \cdot 10^1 + 4 \cdot 10^0 + 1 \cdot 10^{-1} + 3 \cdot 10^{-2}$.

Наиболее употребляемыми в настоящее время позиционными системами являются:

- двоичная (в дискретной математике, информатике, программировании);
- троичная;
- восьмеричная (используется в программировании, информатике);
- десятичная (используется в быту повсеместно);
- двенадцатеричная (счёт дюжинами);
- тринадцатеричная;
- шестнадцатеричная (используется в программировании, информатике);
- шестидесятеричная (единицы измерения времени, измерение углов, координат, долготы и широты).

В позиционных системах, чем больше основание системы, тем меньшее количество разрядов требуется при записи числа.

Вес разряда $p_i$ числа выражается соотношением

$p_i = r^i / r^0 = r^i$,

где $i$ – номер разряда при отсчете справа налево.

Если в $i$-м разряде накопилось значение единиц, равное или большее $r$, то должна происходить передача единицы в старший $i + 1$ разряд. При сложении такая передача информации называется переносом. При вычитании передача

единицы из $i + 1$ разряда в $i$-й – заем.

*Длина числа* – количество позиций (разрядов) в записи числа. В технической реализации под длиной числа понимается длина разрядной сетки.

*Диапазон представления чисел* в заданной системе счисления – интервал числовой оси, заключенный между максимальным и минимальным числами, представленными при заданной длине разрядной сетки.

В вычислительной технике для представления данных и выполнения арифметических операций над ними удобно использовать двоичную, восьмеричную и шестнадцатеричную системы счисления. Ниже коротко рассмотрим каждую из них.

### 2.2. Двоичная система счисления

Для записи числа в двоичной системе счисления используются две цифры: 0 и 1. Основание системы (число 2) записывается как $10_2$ (согласно формуле (1) $2_{10} = 1 \cdot 2^1 + 0 \cdot 2^0$). Используя данную систему, любое число можно выразить последовательностью высоких и низких потенциалов или группой элементов, способных запоминать одно из двух (0,1) значений. Арифметические операции в двоичной системе счисления выполняются по тем же правилам, что и в десятичной системе счисления.

| Сложение | Вычитание | Умножение |
|---|---|---|
| $0 + 0 = 0$ | $0 - 0 = 0$ | $0 \cdot 0 = 0$ |
| $0 + 1 = 1$ | $1 - 0 = 1$ | $0 \cdot 1 = 0$ |
| $1 + 0 = 1$ | $1 - 1 = 0$ | $1 \cdot 0 = 0$ |
| $1 + 1 = 10$ | $10 - 1 = 1$ | $1 \cdot 1 = 1$ |

Рассмотрим несколько примеров, демонстрирующих выполнение арифметических операций:

Примеры:

$$\begin{array}{r} 01010110 \\ +\ 10010111 \\ \hline 11101101 \end{array} \qquad \begin{array}{r} 10101101 \\ -\ 10010110 \\ \hline 00010111 \end{array} \qquad \begin{array}{r} 10011 \\ \times\ \ 1011 \\ \hline 10011 \\ 10011 \\ 10011 \\ \hline 11010001 \end{array}$$

### 2.3. Восьмеричная система счисления

В восьмеричной системе счисления используется восемь цифр: 0, 1, 2, …, 7, а основание (число 8) записывается как $10_8$ (согласно (1) $8_{10} = 1 \cdot 8^1 + 0 \cdot 8^0$). Рассмотрим выполнение операций в восьмеричной системе счисления. При их выполнении используются правила, представленные в таблицах сложения и умножения восьмеричных цифр.

Сложение

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 10 |
| 2 | 3 | 4 | 5 | 6 | 7 | 10 | 11 |
| 3 | 4 | 5 | 6 | 7 | 10 | 11 | 12 |
| 4 | 5 | 6 | 7 | 10 | 11 | 12 | 13 |
| 5 | 6 | 7 | 10 | 11 | 12 | 13 | 14 |
| 6 | 7 | 10 | 11 | 12 | 13 | 14 | 15 |
| 7 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |

Умножение

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 2 | 4 | 6 | 10 | 12 | 14 | 16 |
| 3 | 3 | 6 | 11 | 14 | 17 | 22 | 25 |
| 4 | 4 | 10 | 14 | 20 | 24 | 30 | 34 |
| 5 | 5 | 12 | 17 | 24 | 31 | 36 | 43 |
| 6 | 6 | 14 | 22 | 30 | 36 | 44 | 52 |
| 7 | 7 | 16 | 25 | 34 | 43 | 52 | 61 |

Примеры:

$$\begin{array}{r} 35047326 \\ +\ 21764254 \\ \hline 57033602 \end{array} \qquad \begin{array}{r} 43670154 \\ -\ 17352326 \\ \hline 24315626 \end{array} \qquad \begin{array}{r} 20314 \\ \times\ \quad 264 \\ \hline 101460 \\ 142310 \\ 40630 \\ \hline 5607560 \end{array}$$

**2.4. Шестнадцатеричная система счисления**

В шестнадцатеричной системе счисления используются шестнадцать символов: 0, 1, 2, …, 9, *A*, *B*, *C*, *D*, *E*, *F*. Основание (число 16) записывается как $10_{16}$ (согласно формулы (1) $16_{10} = 1 \cdot 16^1 + 0 \cdot 16^0$).

Сложение

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | *A* | *B* | *C* | *D* | *E* | *F* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | *A* | *B* | *C* | *D* | *E* | *F* | 10 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | *A* | *B* | *C* | *D* | *E* | *F* | 10 | 11 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 | *A* | *B* | *C* | *D* | *E* | *F* | 10 | 11 | 12 |
| 4 | 5 | 6 | 7 | 8 | 9 | *A* | *B* | *C* | *D* | *E* | *F* | 10 | 11 | 12 | 13 |
| 5 | 6 | 7 | 8 | 9 | *A* | *B* | *C* | *D* | *E* | *F* | 10 | 11 | 12 | 13 | 14 |
| 6 | 7 | 8 | 9 | *A* | *B* | *C* | *D* | *E* | *F* | 10 | 11 | 12 | 13 | 14 | 15 |
| 7 | 8 | 9 | *A* | *B* | *C* | *D* | *E* | *F* | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 8 | 9 | *A* | *B* | *C* | *D* | *E* | *F* | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 9 | *A* | *B* | *C* | *D* | *E* | *F* | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| *A* | *B* | *C* | *D* | *E* | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| *B* | *C* | *D* | *E* | *F* | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1*A* |
| *C* | *D* | *E* | *F* | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1*A* | 1*B* |
| *D* | *E* | *F* | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1*A* | 1*B* | 1*C* |
| *E* | *F* | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1*A* | 1*B* | 1*C* | 1*D* |
| *F* | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1*A* | 1*B* | 1*C* | 1*D* | 1*E* |

Умножение

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | *A* | *B* | *C* | *D* | *E* | *F* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | *A* | *B* | *C* | *D* | *E* | *F* |
| 2 | 2 | 4 | 6 | 8 | *A* | *C* | *E* | 10 | 12 | 14 | 16 | 18 | 1*A* | 1*C* | 1*E* |
| 3 | 3 | 6 | 9 | *C* | *F* | 12 | 15 | 18 | 1*B* | 1*E* | 21 | 24 | 27 | 2*A* | 2*D* |
| 4 | 4 | 8 | *C* | 10 | 14 | 18 | 1*C* | 20 | 24 | 28 | 2*C* | 30 | 34 | 38 | 3*C* |
| 5 | 5 | *A* | *F* | 14 | 19 | 1*E* | 23 | 28 | 2*D* | 32 | 37 | 3*C* | 41 | 46 | 4*B* |
| 6 | 6 | *C* | 12 | 18 | 1*E* | 24 | 2*A* | 30 | 36 | 3*C* | 42 | 48 | 4*E* | 54 | 5*A* |
| 7 | 7 | *E* | 15 | 1*C* | 23 | 2*A* | 31 | 38 | 3*F* | 46 | 4*D* | 54 | 5*B* | 62 | 69 |
| 8 | 8 | 10 | 18 | 20 | 28 | 30 | 38 | 40 | 48 | 50 | 58 | 60 | 68 | 70 | 78 |
| 9 | 9 | 12 | 1*B* | 24 | 2*D* | 36 | 3*F* | 48 | 51 | 5*A* | 63 | 6*C* | 75 | 7*E* | 87 |
| *A* | *A* | 14 | 1*E* | 28 | 32 | 3*C* | 46 | 50 | 5*A* | 64 | 6*E* | 78 | 82 | 8*C* | 96 |
| *B* | *B* | 16 | 21 | 2*C* | 37 | 42 | 4*D* | 58 | 63 | 6*E* | 79 | 84 | 8*F* | 9*A* | *A*5 |
| *C* | *C* | 18 | 24 | 30 | 3*C* | 48 | 54 | 60 | 6*C* | 78 | 84 | 90 | 9*C* | *A*8 | 84 |
| *D* | *D* | 1*A* | 27 | 34 | 41 | 4*E* | 5*B* | 68 | 75 | 82 | 8*F* | 9*C* | *A*9 | *B*6 | *C*3 |
| *E* | *E* | 1*C* | 2*A* | 38 | 46 | 54 | 62 | 70 | 7*E* | 8*C* | 9*A* | *A*8 | *B*6 | *C*4 | *D*2 |
| *F* | *F* | 1*E* | 2*D* | 3*C* | 4*B* | 5*A* | 69 | 78 | 87 | 96 | *A*5 | *B*4 | *C*3 | *D*2 | *E*1 |

Примеры:

$$\begin{array}{r} 9A4DBE6 \\ +\ A2864AF \\ \hline 13CD4095 \end{array} \qquad \begin{array}{r} 4A6B0F52 \\ -\ 2F392F2F \\ \hline 1B31E023 \end{array} \qquad \begin{array}{r} 1B2A3 \\ \times\ \quad 3C2 \\ \hline 36546 \\ 145FA4 \\ 517E9 \\ \hline 6614886 \end{array}$$

### 2.5. Критерии выбора системы счисления

Сформулируем основные требования, которым должна удовлетворять система счисления, используемая для выполнения операций в ЭВМ [9].

1. **Простота технической реализации**. Для хранения чисел в той или иной системе счисления используются *n*-позиционные запоминающие элементы. Элемент будет тем проще, чем меньше состояний требуется для запоминания цифры числа, т. е. чем меньше основание системы счисления. Двухпозиционными элементами, имеющими два состояния, являются, например:

- электромеханическое реле (контакты замкнуты – 1, разомкнуты – 0);
- конденсатор (заряжен – 1, разряжен – 0);
- полупроводниковый элемент (если открыт, то хранит 0, иначе − 1) и др.

Трехпозиционные элементы более редки. Например, конденсатор для запоминания цифр: 0 – разряжен, 1– заряжен в одном направлении, 2 – в другом. Таким образом, реализация *n*-позиционных элементов более сложна, чем двухпозиционных.

2. **Наибольшая помехоустойчивость кодирования цифр**. Положим, что при технической реализации любой системы счисления диапазон изменения электрического значения наибольшей и наименьшей цифры одинаков. Очевидно преимущество систем с меньшим основанием, так как представление соседних цифр в этих системах отличается друг от друга больше, чем для систем с большим основанием.

Таким образом при наложении помехи на основной сигнал, соответствующий некоторой цифре, наиболее вероятна ошибка в устройствах, для которых используется система счисления с наибольшим основанием (рис. 2). Следовательно, при увеличении основания помеха может привести к искажению числа.

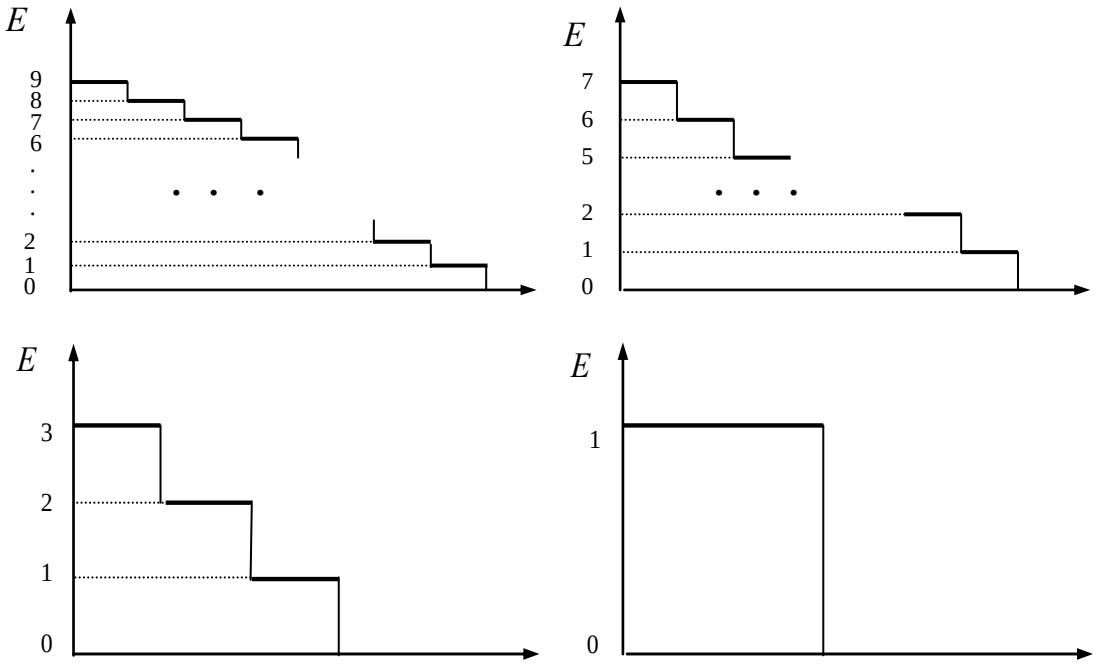



Рис. 2. Влияние помех на значение цифры

3. **Минимум оборудования**. Пусть $r$ − количество цифр в числе, $n$ − количество разрядов в каждом числе, тогда $D = r \cdot n$ − количество цифроразрядов на одно число. Надо найти такую систему счисления, которая имеет минимальное количество цифроразрядов при заданном количестве чисел $N$:

$$N = r^n,$$

$$n = \log_r N,$$

$$D = r \cdot \log_r N = \frac{r}{\log_N r}.$$

Будем считать, что основание системы счисления может принимать любые значения, а не только целочисленные, изменяясь непрерывно, а не дискретно. Соответственно количество цифроразрядов может быть также величиной непрерывной, связанной с основанием системы счисления логарифмической зависимостью:

$$D(r) = \frac{r}{\log_N r}.$$

Это позволит свести задачу нахождения $D(r)_{\min}$ к исследованию функции на экстремум

$$\frac{dD}{dr} = \frac{\log_N r - r\frac{1}{r}\log_N e}{(\log_N r)^2} = 0,$$

следовательно, $r_{\text{опт}} = e \cong 2{,}718$. Так как основание системы счисления должно быть целым числом, то основанием, наиболее близким к $e$, является основание $r = 3$. Но для реализации этого нужен элемент с тремя стабильными состояниями.

Выясним, насколько каждое из целочисленных оснований $r_i$ уступает $r_{\text{опт}}$. Для этого оценим каждое основание $r_i$ не абсолютной величиной $D_i$, а его относительным значением, исходя из выражения

$$D_{i(\text{отн})} = \frac{D_i}{D_{\min}} = \frac{r_i \log_{r_i} N}{D_{\min}},$$

где $D_{\min} = r_{\text{опт}} \log_{r_{\text{опт}}} N = e \cdot \ln N.$

Следовательно, получим

$$D(r_i) = \frac{r_i}{e \ln r_i}.$$

Выполнив расчеты для некоторых оснований, получим следующие результаты:

| $r_i$ | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ... |
|---|---|---|---|---|---|---|---|---|
| $D_{i\,(\text{отн})}$ | 1,062 | 1,004 | 1,062 | 1,143 | 1,232 | 1,300 | 1,416 | ... |

4. **Простота арифметических действий**. Чем меньше цифр в системе счисления, тем проще арифметические действия над ними. Таблицы для выполнения четырех арифметических операций будут усложняться с увеличением основания системы счисления. Это можно принять за косвенное до-

казательство выдвинутого положения.

5. **Наибольшее быстродействие**. Как будет показано далее, операции вычитания, умножения и деления могут быть выполнены посредством операции алгебраического сложения. Алгебраическое сложение чисел часто сводится к их арифметическому сложению. Таким образом, взяв его за базовое, вычислим соотношение времени, необходимого на сложение в некоторой $r_i$ системе счисления

$$T_{\text{сл}} = n\, t_{\text{пер}} = t_{\text{пер}} \log_{r_i} N,$$

где $t_{\text{пер}}$ – время затрачиваемое на формирование и выполнение переноса.

Для упрощения оценки быстродействия перейдем от абсолютной оценки величин времени сложения к относительной

$$\delta = \frac{T_{\text{сл max}}}{T_{\text{сл}}}.$$

Максимальное время сложения получается в случае если выбрана система счисления с основанием 2. Это обусловлено тем, что число в системе счисления с меньшим основанием записывается большим количеством цифроразрядов.

$$T_{\text{сл max}} = T_{\text{сл (2)}} = t_{\text{пер}} \log_2 N,$$

$$\delta = \frac{t_{\text{пер}} \log_2 N}{t_{\text{пер}} \log_{r_i} N} = \log_2 r_i,$$

эта относительная оценка не зависит от $N$ и $t_{\text{пер}}$. Выполнив вычисления для различных $r$, получим следующие результаты:

| $r_i$ | 2 | 3 | 4 | 5 | ... |
|---|---|---|---|---|---|
| $\delta$ | 1,00 | 1,58 | 2,00 | 2,32 | ... . |

6. **Простота аппарата для выполнения анализа и синтеза цифровых устройств**. Математическим аппаратом, позволяющим относительно просто и экономно строить цифровые схемы, является алгебра логики. Наибольшее распространение и законченность вследствие своей простоты получила двузначная логика.

7. **Удобство работы с ЭВМ**. Наиболее удобной системой счисления для выполнения операций человеком является десятичная система счисления. Но в ЭВМ для выполнения арифметических операций числа из десятичной системы счисления требуется переводить во внутреннюю систему счисления.

Для системы счисления с основанием, большим 10, появляются новые цифры. Таким образом, система счисления должна иметь минимальное число цифр, так как в этом случае можно пользоваться младшими цифрами десятичной системы счисления.

8. **Возможность представления любого числа в рассматриваемом диапазоне величин**.

Обеспечивает любая позиционная система счисления.

9. **Единственность представления числа**.

### 2.6. Перевод чисел из одной системы счисления в другую

Так как числа, над которыми производятся арифметические операции, могут быть представлены в различных позиционных системах счисления, то для выполнения действий над ними требуется привести их к одной системе счисления. Следует помнить, что при переводе числа из одной системы счисления в другую количественное значение числа не изменяется, а меняется только форма записи числа. Все методы перевода чисел можно подразделить на две группы: перевода целых и дробных чисел.

Необходимо отметить, что для перевода неправильных дробей отдельно выделяется целая и дробная части числа и с использованием соответствующих методов выполняется их перевод. Результаты записываются в виде новой неправильной дроби.

#### 2.6.1. Перевод целых чисел

*Метод подбора степеней основания*. В соответствии с (2) целые числа в системах счисления с основаниями $r_1$ и $r_2$ могут быть представлены:

$$A_{r_1} = \sum_{i=0}^{n} a_i r_1^{\,i} = \sum_{j=0}^{k} b_j r_2^{\,j} = A_{r_2} .$$

В общем случае перевод числа из системы счисления с основанием $r_1$ в систему счисления с основанием $r_2$ можно представить как задачу определения коэффициентов $b_i$ нового ряда, изображающего число в системе счисления с основанием $r_2$. Основная трудность в выборе максимальной степени основания $r_2$, которая еще содержится в числе $A_{r_1}$. Все действия должны выполняться по правилам $r_1$-арифметики (т. е. исходной системы счисления). После нахождения максимальной степени и соответствующего ей коэффициента необходимо найти коэффициенты для всех остальных (младших) степеней.

*Примеры.* Перевести десятичное число ($A_{10}$) 37 в двоичную ($A_2$) систему счисления.

$$37_{10} = 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 100101_2 .$$

Перевести восьмеричное число ($A_8$) 703 в десятичную ($A_{10}$) систему счисления.

$$703 = 7 \cdot 8^2 + 0 \cdot 8^1 + 3 \cdot 8^0 = 451_{10} .$$

Перевести шестнадцатеричное ($A_{16}$) *B2E* в десятичную ($A_{10}$) систему счисления.

$$B2E = 11 \cdot 16^2 + 2 \cdot 16^1 + 14 \cdot 16^0 = 2862_{10} .$$

*Метод деления на основание системы счисления*. На основании формулы (1) число $A_{r_1}$ в системе счисления с основанием $r_2$ запишется в виде

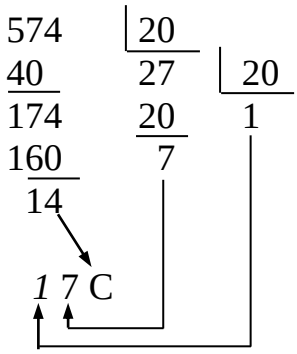
$$A_{r_2} = a_n r_2^n + a_{n-1} r_2^{n-1} + \ldots + a_1 r_2^1 + a_0 r_2^0 .$$

Переписав это выражение по схеме Горнера, получим

$$A_{r_2} = (\ldots(a_n r_2 + a_{n-1}) r_2 + \ldots + a_1) r_2 + a_0 .$$

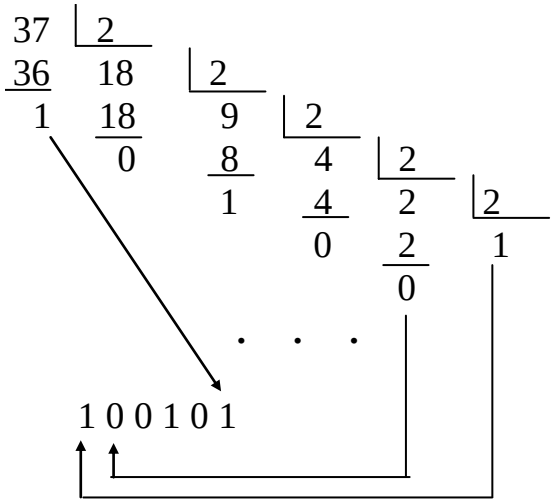
Разделив правую часть на $r_2$, получим первый остаток $a_0$ и целое частное $(\ldots(a_n r_2 + a_{n-1}) r_2 + \ldots + a_1)$. Разделив целое частное на $r_2$, получим следующий остаток $a_1$ и новое целое частное. Последовательно выполнив деление $n + 1$ раз, получим последнее целое частное $a_n < r_2$, являющееся старшей цифрой числа.

*Примеры*. Перевести восьмеричное число ($A_8$) 574 в шестнадцатеричную ($A_{16}$) систему счисления. Деление выполняется в восьмеричной системе счисления.

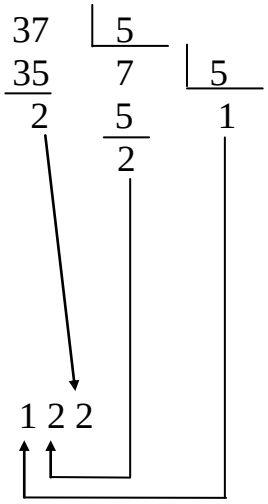



Ответ: $A_{16} = 17C$.

*Примеры*. Перевести десятичное число ($A_{10}$) 37 в двоичную ($A_2$) и пятиричную ($A_5$) системы счисления. Деление в десятичной системе счисления.

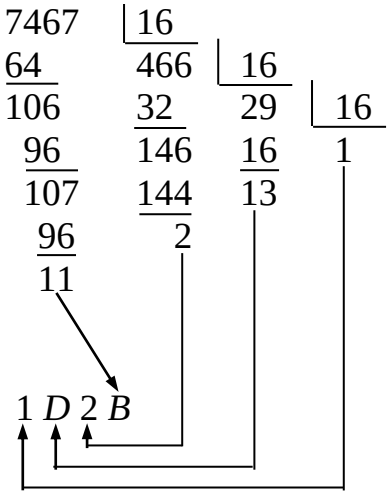



Ответ: $A_2 = 100101$.




Ответ: $A_5 = 122$.

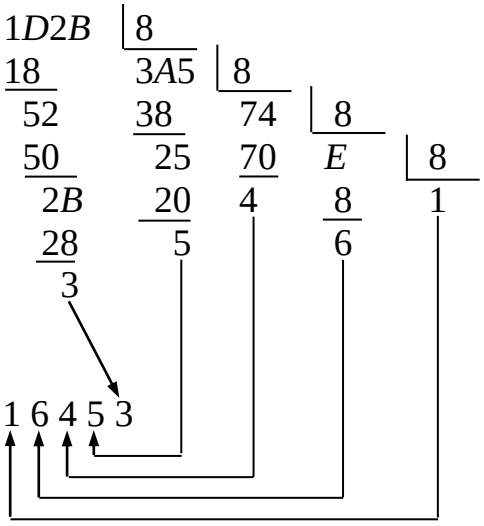
*Пример.* Перевести десятичное число ($A_{10}$) 7467 в шестнадцатеричную ($A_{16}$) систему счисления. Деление выполняется в десятичной системе счисления.



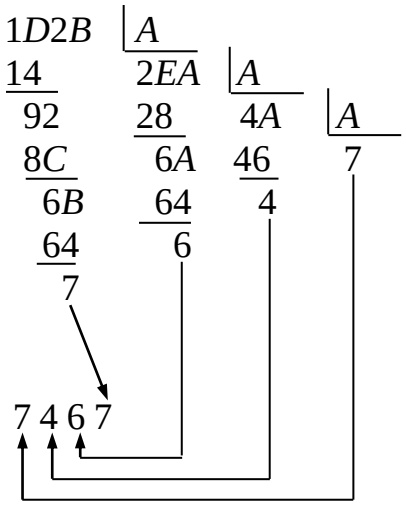

Ответ: $A_{16} = 1D2B$.

*Примеры.* Перевести шеснадцатеричное число ($A_{16}$) 1*D*2*B* в восьмерич-

ную ($A_8$) и десятичную ($A_{10}$) систему счисления. Деление выполняется в шестнадцатеричной системе счисления.




Ответ: $A_8$ = 16453.




Ответ: $A_{10}$ = 7467.

### 2.6.2. Перевод правильных дробей

Конечной исходной десятичной дроби в новой системе счисления может соответствовать бесконечная (иногда периодическая) дробь. В этом случае количество знаков в представлении дроби в новой системе счисления берется в зависимости от требуемой точности.

*Метод подбора величин, обратных степеням основания*

В этом методе необходимо определять величину обратную степени основания умноженную на коэффициент $k$ (1…$r$ − 1), которая содержится в исходном числе. В этом случае в число в новой системе счисления вносится цифра $k$ , исходное число уменьшается на величину, обратную степени основания умноженную на коэффициент $k$, операция повторяется вновь для поиска следующей цифры.

*Пример*. Перевести десятичное число ($A_{10}$) 0,716 в двоичную ($A_2$) систему счисления. В примере $k$ принимает значения из множества {0,1}.

$$r^{-1}=\frac{1}{r}=\frac{1}{2}=0,5;\quad r^{-2}=\frac{1}{r^2}=\frac{1}{4}=0,25;\quad r^{-3}=\frac{1}{r^3}=\frac{1}{8}=0,125;\quad r^{-4}=\frac{1}{r^4}=\frac{1}{16}=0,0625...$$

| | |
|---|---|
| 0,7160 | $A_{10}$ (исходное число) |
| 0,5 | $r^{-1} \cdot 1$ ($k$ = 1 т. к. $A_{10} \geq r^{-1} \cdot k$) |
| **0**,2160 | первый остаток |
| 0,0 | $r^{-2} \cdot 0$ ($k$ = 0 т. к. первый остаток $\leq r^{-2} \cdot k$) |
| 0,2160 | второй остаток |
| 0,125 | $r^{-3} \cdot 1$ ($k$ = 1 т. к. второй остаток $\geq r^{-3} \cdot k$) |
| **0**,0910 | третий остаток |
| 0,0625 | $r^{-4} \cdot 1$ ($k$ = 1 т. к. третий остаток $\geq r^{-4} \cdot k$) |
| . . . | ответ: $A_2$ = 0,1011… |

Количество разрядов после запятой зависит от точности, с которой требу-

ется представить число.

*Метод умножения на основание $r_2$ новой системы счисления*. Используя формулу (1), дробное число $A_{r_2}$ в системе счисления с основанием $r_2$ запишется в виде

$$A_{r_2} = a_{-1}\, r_2^{-1} + ... + a_{-m}\, r_2^{-m}.$$

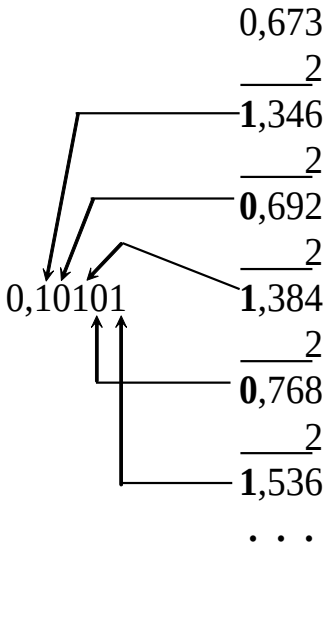
Переписав это выражение по схеме Горнера, получим

$$A_{r_2} = r_2^{-1}\,(a_{-1} + r_2^{-1}\,(a_{-2} + ... + a_{-m}\, r_2^{-1})...).$$

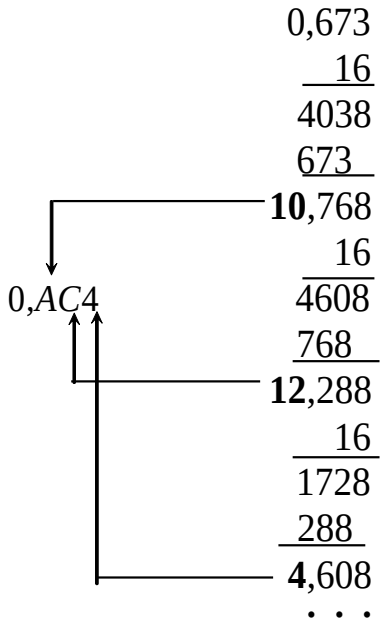
Умножив правую часть на $r_2$, получим новую неправильную дробь, целая часть которой есть $a_{-1}$ (старшая цифра числа $A_{r_2}$ ). Продолжим процесс умножения дробной части на $r_2$ $m - 1$ раз, получим цифры $a_{-2}, a_{-3}, \ldots$ числа $A_{r2}$.

Процесс умножения может быть прекращен, если во всех разрядах после очередного умножения получены нули либо достигнута требуемая точность.

*Пример*. Перевести десятичное число ($A_{10}$) 0,673 в двоичную ($A_2$) и шестнадцатеричную ($A_{16}$) систему счисления. Умножение выполняется в десятичной системе счисления.

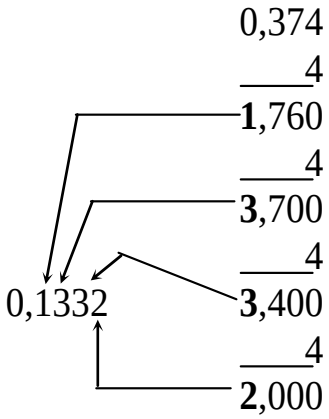



Ответ: $A_2 = 0{,}10101\ldots$

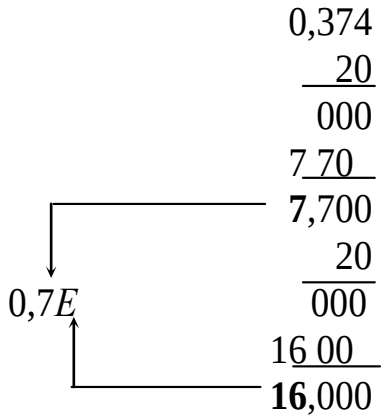



Ответ: $A_{16} = 0{,}AC4\ldots$

*Пример*. Перевести восьмеричное число ($A_8$) 0,374 в четверичную ($A_4$) и шестнадцатеричную ($A_{16}$) систему счисления. Умножение выполняется в восьмеричной системе счисления.




Ответ: $A_4 = 0{,}1332$.




Ответ: $A_{16} = 0{,}7E$.

### 2.6.3. Перевод чисел из одной системы счисления в другую, основание которой кратно степени 2

К таким системам относятся двоичная, четверичная, восьмеричная и другие системы счисления.

$$A_8 = \sum_{i=0}^{n} a_{m_i} 8^i = \sum_{i=0}^{n} (b_{2i} 2^2 + b_{1i} 2^1 + b_{0i}) 2^{3i} = A_2 \cdot$$

Ограничимся тремя восьмеричными разрядами, придавая $i$ значения 0, 1, 2.

$$\underbrace{(b_{20} 2^2 + b_{10} 2 + b_{00})}_{i=0} + \qquad \text{(младшая восьмеричная цифра с весом } 8^0 \text{)}$$

$$+ \underbrace{(b_{21} 2^5 + b_{11} 2^4 + b_{01} 2^3)}_{i=1} +$$

$$+ \underbrace{(b_{22} 2^8 + b_{12} 2^7 + b_{02} 2^6)}_{i=2} \qquad \text{(старшая восьмеричная цифра с весом } 8^2 \text{)}$$

При переводе числа из двоичной системы счисления в восьмеричную необходимо разделить его разряды на триады, начиная с младших разрядов, и каждую триаду заменить восьмеричной цифрой.

Общий метод перевода числа из двоичной системы счисления в систему с счисления с основанием $2^n$.

**Метод**. Для записи двоичного числа в системе с основанием $2^n$ достаточно данное двоичное число разбить на группы цифр от запятой по $n$ в каждой группе. Затем каждую группу цифр следует рассматривать как $n$-разрядное двоичное число и записать его эквивалент в системе счисления с основанием $2^n$.

*Пример*. Перевести двоичное число в восьмеричную систему счисления.

$$111011101{,}11101 = \underset{7}{\underline{111}}\ \underset{3}{\underline{011}}\ \underset{5}{\underline{101}}\ ,\ \underset{7}{\underline{111}}\ \underset{2}{\underline{010}} = 735{,}72$$

В этом примере мы столкнулись с ситуацией, когда после запятой у двоичного числа количество цифр не кратно трем. В этом случае мы имеем право дописать столько нулей, сколько нам необходимо. У целых чисел незначащие нули дописываются слева от самого старшего разряда, у дробей – справа от самого младшего разряда.

*Пример*. Перевести двоичное число в шеснадцатеричную систему счисления.

$$111011101{,}11101 = \underset{1}{\underline{0001}}\ \underset{D}{\underline{1101}}\ \underset{D}{\underline{1101}}\ ,\ \underset{E}{\underline{1110}}\ \underset{8}{\underline{1000}} = 1DD{,}E8$$

**Метод**. Для замены числа, записанного в системе с основанием $2^n$, на эквивалентное ему число в двоичной системе счисления, достаточно каждую цифру исходного числа заменить $n$-разрядным двоичным числом.

*Пример*. Перевести четверичное число в двоичную систему счисления.

$$3021{,}322 = \underset{11}{\underline{3}}\ \underset{00}{\underline{0}}\ \underset{10}{\underline{2}}\ \underset{01}{\underline{1}}\ ,\ \underset{11}{\underline{3}}\ \underset{10}{\underline{2}}\ \underset{10}{\underline{2}} = 11001001{,}11101$$

Обратите внимание, что запятая, отделяющая дробную часть от целой, остается на месте. В данном примере, каждая четверичная цифра заменяется

соответствующей двоичной диадой. Полученные диады и образуют число в двоичной системе счисления.

*Пример*. Перевести шеснадцатеричное число в двоичную систему счисления.

$$F093,A8 = \underset{1111}{F}\ \underset{0000}{0}\ \underset{1001}{9}\ \underset{0011}{3},\ \underset{1010}{A}\ \underset{1000}{8} = 1111000010010011,10101$$

В данном примере каждая шеснадцатеричная цифра исходного числа заменяется тетрадой, совокупность которых и образует новое число.

Приведем еще несколько примеров перевода чисел.

*Пример.* $A_8 = 45$ – исходное число в восьмеричной системе счисления

$A_2 = \underline{100}\ \underline{101}$ – каждая цифра исходного числа заменена соответствующей двоичной триадой

*Пример.* $A_2 = \underline{0010}\ \underline{0101}$ – исходное число в двоичной системе счисления (разбитое на тетрады)

$A_{16} = 2\ 5$ – каждой тетраде разрядов поставлена в соответствие шеснадцатеричная цифра

*Пример.* $A_{16} = 7\ B, E\ 6$ – исходное число в шеснадцатеричной системе счисления

$A_2 = \underline{0111}\ \underline{1011},\underline{1110}\ \underline{0110}$ – каждая цифра исходного числа заменена двоичной тетрадой

$\underline{001}\ \underline{111}\ \underline{011},\underline{111}\ \underline{001}\ \underline{100}$ – число $A_2$ разбито на триады

$A_8 = 1\ 7\ 3, 7\ 1\ 4$ – каждой триаде поставлена в соответствие восьмеричная цифра

### 2.7. Кодирование чисел

**Кодирование знака числа**. Кодирование чисел позволяет заменить операцию арифметического вычитания операцией алгебраического сложения с помощью двоичного сумматора. Для кодирования знака числа используется специальный двоичный разряд, называемый *знаковым*. При этом знак плюс кодируется двоичной цифрой 0, а минус – цифрой 1 (для системы счисления с основанием $r$ – цифрой $r-1$). Для машинного представления чисел используют три основных вида кодов: прямой, обратный и дополнительный. Общая схема кода числа: *код знака . код числа.*

**Прямой код числа**. При этом способе кодирования чисел кодируется только знак числа, а значащая часть остается без изменения.

$$[A]_{\text{пр}} = \begin{cases} 0, A, & \text{если } A \ge 0 \\ r-1, |A|, & \text{если } A < 0 \end{cases}$$ – для правильных дробей,

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

*Пример:* $A = +0,1101$ $\quad A = -0,1101$

$[A]_{\text{пр}} = 0,1101$ $\quad [A]_{\text{пр}} = 1,1101$

$$[A]_{\text{пр}}=\begin{cases}0.A, \text{ если } A\geq 0\\ r-1.|A|, \text{ если } A<0\end{cases}$$ – для целых чисел.

*Пример.* $A = +1101$ $A = -1101$

$[A]_{\text{пр}} = 0.1101$ $[A]_{\text{пр}} = 1.1101$

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

Недостатком прямого кода является сложность выполнения операции сложения чисел с разными знаками.

Для арифметических операций над числами в прямом коде используется сумматор прямого кода. В этом сумматоре отсутствует цепь поразрядного переноса между старшим значащим и знаковым разрядами, т. е. на этом сумматоре невозможно выполнение операции алгебраического сложения.

**Дополнительный код числа**. Число $A'$ называется *дополнением* к числу $A$, если выполняется соотношение: $A + A' = r^n$ для целых чисел или $A + A' = r^0$ для дробных чисел, где $n$ – количество цифр в записи числа $A$.

*Пример*. Записать дополнение числа $A_{10} = 378$

$n = 3$ (в числе 3 цифры)

$A'_{10} = 10^3 - |A| = 1000 - 378 = 622$

378

621 – все разряды дополняются до младшей цифры системы счисления

1 – младший разряд дополняется до основания системы счисления

1000

*Пример*. Записать дополнение числа $A_2 = 1011$.

$n = 4$ (в числе 4 цифры)

$A'_2 = 2^4 - |A| = 10000 - 1011 = 0101$, или $A'_2 = 0101$.

**Замена операции вычитания операцией сложения**. В ЭВМ достаточно сложно выполнить операцию вычитания $(A - B)$. Для этого требуется:

1) сравнить числа и выявить наибольшее из них по абсолютной величине;
2) наибольшее число разместить на входах вычитающего устройства;
3) выполнить операцию вычитания;
4) присвоить значению разности знак наибольшего по абсолютной величине числа.

Для сложения чисел в дополнительных кодах требуется сумматор и неважно, какой знак имеют подаваемые на его входы слагаемые $A$ и $B$ и в каком порядке они поступают.

*Пример.* Пусть необходимо сложить числа 487 и −348.

$A = 487$ $A = 487$

$B = -348$ $B' = 652$

$A + B = 139$ $A + B' = 1139$

$A + (10^3 - B) = A - B + 10^3$ ($10^3$ игнорируется).

*Пример*. Выполнить операцию сложения над теми же числами, но изменив знаки слагаемых, т. е. сложить 348 и −487.

$A = \ \ 348$ $\qquad A = 348$

$B = -487$ $\qquad B' = 513$

$A + B = -139$ $\qquad A + B' = 861$ – дополнение числа 139.

Дополнительный код отрицательных чисел является математическим дополнением абсолютной величины числа до основания $r$ системы счисления для дробных чисел и до $r^{n+1}$ для целых чисел.

$|A| + [A]_{\text{доп}} = r$ – для дробных чисел, $\qquad |A| + [A]_{\text{доп}} = r^{n+1}$ – для целых чисел, где $n$ – количество цифр в записи числа $A$.

Положительные числа в дополнительном коде не меняют своего изображения. Правило преобразования числа в дополнительный код можно записать:

$$[A]_{\text{доп}} = \begin{cases} 0, A, & \text{если } A \geq 0 \\ r - |A|, & \text{если } A < 0 \end{cases}$$ – для правильных дробей;

$$[A]_{\text{доп}} = \begin{cases} 0.\ A, & \text{если } A \geq 0 \\ r^{n+1} - |A|, & \text{если } A < 0 \end{cases}$$ – для целых чисел.

Рассмотрим несколько примеров сложения чисел в дополнительных кодах.

$A = \ \ 0{,}1001$ $\quad [A]_{\text{доп}} = \ 0{,}1001$ $\quad A = -0{,}1001$ $\quad [A]_{\text{доп}} = 1{,}0111$

$B = -0{,}0100$ $\quad [B]_{\text{доп}} = \underline{\ 1{,}1100}$ $\quad B = \ \ 0{,}0100$ $\quad [B]_{\text{доп}} = \underline{0{,}0100}$

$[A + B]_{\text{доп}} = 10{,}0101$ $\qquad [A + B]_{\text{доп}} = 1{,}1011$

**Теорема**. Сумма дополнительных кодов чисел есть дополнительный код результата.

Доказательство теоремы приведено в [1].

Теорема справедлива для всех случаев, в которых не возникает переполнения разрядной сетки, что позволяет складывать машинные представления чисел по правилам двоичной арифметики, не разделяя знаковую и значащую части числа. Для выполнения арифметических операций над числами в дополнительном коде используется *двоичный сумматор дополнительного кода*, характерной особенностью которого является наличие поразрядного переноса из старшего значащего в знаковый разряд.

**Обратный код числа**. Обратный код двоичного числа является инверсным изображением числа, в котором все разряды исходного числа принимают инверсное (обратное) значение. Правила преобразования чисел в обратный код аналитически можно определить следующим образом:

$$[A]_{\text{обр}} = \begin{cases} 0, A, & \text{если } A \geq 0 \\ r - |A| - r^{-n}, & \text{если } A < 0 \end{cases}$$ – для правильных дробей;

$$[A]_{\text{обр}} = \begin{cases} 0.\ A, & \text{если } A \geq 0 \\ r^{n+1} - |A| \ -1, & \text{если } A < 0 \end{cases}$$ – для целых чисел.

Выполнение арифметических операций над числами в обратном коде осуществляется на *сумматоре обратного кода.* Этот код имеет малозначительный недостаток: требует наличия в сумматоре цепи циклического переноса из

знакового разряда в младший значащий. Это может привести к увеличению времени выполнения арифметических операций. Ниже приведены несколько примеров выполнения арифметических операций над числами, записанными в обратном коде.

*Примеры.* Выполнить сложение чисел $A$ и $B$ в обратном коде.

$A = 0{,}1001$ $\quad [A]_{обр} = \ 0{,}1001$ $\quad A = -0{,}1001$ $\quad [A]_{обр} = 1{,}0110$

$B = -0{,}0100$ $\quad [B]_{обр} = \underline{\ 1{,}1011}$ $\quad B = 0{,}0100$ $\quad [B]_{обр} = \underline{\ 0{,}0100}$

$10{,}0100$ $\quad [A + B]_{обр} = 1{,}1010$

$\underline{\longrightarrow 1}$

$[A + B]_{обр} = 0{,}0101$

**Теорема**. Сумма обратных кодов чисел есть обратный код результата.

Доказательство теоремы приведено в [1].

### 2.8. Переполнение разрядной сетки

При выполнении некоторых арифметических операций может возникнуть ситуация, когда старшие разряды результата операции не помещаются в отведенной для него области памяти. Эта ситуация называется *переполнением разрядной сетки формата числа*. Причиной переполнения может служить, например, суммирование двух чисел с одинаковыми знаками, которые в сумме дают величину, большую или равную единице (при сложении правильных дробей), или величине $r^n$ (при сложении целых чисел). При выполнении операций над числами с разными знаками переполнение не возникает.

*Пример.* $A = +0{,}101$ $\quad [A]_{доп} = 0{,}101$

$B = +0{,}110$ $\quad [B]_{доп} = \underline{0{,}110}$

$[A + B]_{доп} = 1{,}011$

В результате сложения двух положительных чисел получено отрицательное число, что является ошибкой. Результат неверен также и по величине.

Для обнаружения переполнения можно использовать следующие признаки:

- знаки слагаемых не совпадают со знаком суммы;
- есть перенос только в знаковый ($П_1$) или только из знакового ($П_2$) разряда.

Если при сложении чисел с фиксированной запятой возникло переполнение, то вырабатывается сигнал переполнения разрядной сетки и вычисления прекращаются.

Следует отметить, что при сложении чисел в дополнительном коде возможен случай, когда переполнение не фиксируется. Это происходит тогда, когда сумма модулей двух отрицательных чисел равна удвоенному весу единицы старшего разряда числа.

*Пример.* $A = -0{,}101$ $\quad [A]_{доп} = 1{,}011$

$B = -0{,}011$ $\quad [B]_{доп} = \underline{1{,}101}$

$[A + B]_{доп} = 1{,}000$

### 2.9. Модифицированные коды

Для обнаружения переполнения разрядной сетки можно использовать модифицированные коды. Модифицированные коды отличаются от обычных кодов тем, что знак числа кодируется не одним, а двумя разрядами (Зн1 и Зн2). При выполнении алгебраического сложения или вычитания два знаковых разряда участвуют в операции как равноправные цифровые разряды. После выполнения операции содержимое знаковых разрядов определяет знак результата (левый знаковый разряд) и наличие переполнения (несовпадение знаковых разрядов): комбинация 01 фиксирует переполнение при сложении положительных чисел (положительное переполнение), а 10 – отрицательных (отрицательное переполнение).

*Пример*. Положительное переполнение

$A$= +0,101 $\quad [A]^{\text{мод}}_{\text{доп}} = 00{,}101$

$B$= +0,110 $\quad [B]^{\text{мод}}_{\text{доп}} = \underline{00{,}110}$

$$[A]^{\text{мод}}_{\text{доп}} + [B]^{\text{мод}}_{\text{доп}} = 01{,}011$$

отрицательное переполнение

$A = -0{,}101 \quad [A]^{\text{мод}}_{\text{доп}} = 11{,}011$

$B = -0{,}110 \quad [B]^{\text{мод}}_{\text{доп}} = \underline{11{,}010}$

$$[A]^{\text{мод}}_{\text{доп}} + [B]^{\text{мод}}_{\text{доп}} = 10{,}101$$

Функция переполнения имеет вид: $f = Зн1 \cdot \overline{Зн2} \vee \overline{Зн1} \cdot Зн2 = Зн1 \oplus Зн2$. Операция v (логическое ИЛИ) будет рассмотрена в четвертой главе пособия.

Логическая схема формирования сигнала при возникновении переполнения приведена на рис. 3.

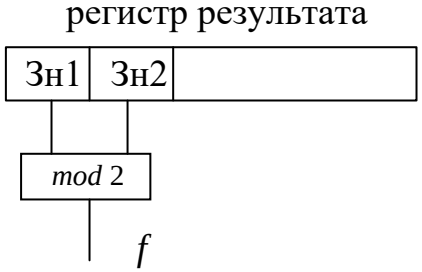



Рис. 3. Схема выявления переполнения

### 2.10. Машинные формы представления чисел

Существуют два основных способа представления данных в ЭВМ: с фиксированной и плавающей запятой (точкой).

**Представление чисел в форме с фиксированной запятой**. Для сокращения длины разрядной сетки и упрощения обработки данных положение запятой может быть зафиксировано. При этом в слове данных сохраняются только

две структурные компоненты: поле знака и поле цифр.

| ± | целая часть | дробная часть |
|---|---|---|

Определим диапазон представления чисел для этого формата.

$$A_{\max} = (2^k - 1) + (1 - 2^{-l}),$$

где $k$ – число разрядов целой, а $l$ – дробной части числа $(k + l = n)$.

В зависимости от размеров целой и дробной частей возможно следующее:

1) $k = 0, \quad l = n \quad A_{\max} = 1 - 2^{-n}$

| ± | 1 | 1 | . . . | 1 | 1 |
|---|---|---|---|---|---|

2) $k = n, \quad l = 0 \quad A_{\max} = 2^n - 1$

| ± | 1 | 1 | . . . | 1 | 1 |
|---|---|---|---|---|---|

3) $k = 0, \quad l = n \quad A_{\min} = 2^{-n}$

| ± | 0 | 0 | . . . | 0 | 1 |
|---|---|---|---|---|---|

4) $k = n, \quad l = 0 \quad A_{\min} = 1$

| ± | 0 | 0 | . . . | 0 | 1 |
|---|---|---|---|---|---|

Целые числа в ЭВМ представляются в формате с фиксированной запятой. Возможны четыре формата представления целых чисел:

- целое число;
- короткое целое число;
- длинное целое число;
- упакованное десятичное число.

Целое число занимает 2 или 4 байта. Его формат полностью соответствует используемому центральному процессору. Для представления отрицательных чисел используется дополнительный код. Короткое целое и длинное целое имеют аналогичные форматы, но занимают, соответственно, 4 и 8 байт.

Упакованное десятичное число занимает 10 байт. Это число содержит 18 десятичных цифр, расположенных по две в каждом байте. Знак упакованного десятичного числа находится в старшем бите самого левого байта. Остальные биты старшего байта должны быть равны нулю.

Приведем возможные варианты представления целых чисел (в дополнительном коде):

| | |
|---|---|
| 0 00...00 | нуль |
| 0 00...01 | наименьшее положительное число |
| 0 11...11 | наибольшее положительное число |
| 1 00...01 | наименьшее отрицательное число |
| 1 11...11 | наибольшее отрицательное число |
| 1 00...00 | неопределенность |

К достоинствам использования чисел с фиксированной запятой относится простота выполнения арифметических операций. К недостаткам – то, что ограничение длины разрядной сетки приводит к ограничению диапазона хранимых чисел и потере точности их представления в случае дробных чисел. Поэтому на практике широко используется и другая форма представления чисел.

**Представление чисел в форме с плавающей запятой.** Прежде чем говорить о форматах вещественных чисел, используемых в ЭВМ, вспомним о числах с плавающей запятой, встречающихся в научных расчетах. Общий вид этих чисел можно записать следующим образом:

$$A = \pm m_A r^{\pm p_A},$$

где $r$ – основание системы счисления, в которой записано число; $m_A$ – мантисса, $p_A$ – порядок числа $A$.

Порядок (с учетом знака) показывает, на сколько разрядов и в какую сторону сдвинута запятая при замене формы записи числа с естественной на нормальную.

Например, $A_{10} = 239{,}745 = 0{,}239745 \cdot 10^3 = 239745 \cdot 10^{-3}$.

Наиболее распространено и удобно для представления ограничение вида: $r^{-1} \le |m_A| < 1$.

Форма представления чисел, для которых справедливо данное ограничение, называется нормализованной. Так как абсолютное значение мантиссы в этом случае лежит в диапазоне от $r^{-1}$ до $1 - r^{-n}$, где $n$ – число разрядов мантиссы без знака, то положение разрядов числа в его машинном изображении непостоянно. Отсюда и название этой формы представления чисел – с плавающей запятой. Однако реализация в ЭВМ нормализованных чисел имеет некоторые особенности. Число с плавающей запятой называется нормализованным, если целая часть мантиссы числа состоит из одной, не равной нулю, цифры.

В чем преимущества использования нормализованных чисел? В том, что для фиксированной разрядной сетки числа (т.е. для фиксированного количества цифр в числе) нормализованные числа имеют наибольшую точность. Кроме того, нормализованное представление исключает неоднозначность – каждое число с плавающей запятой может быть представлено различными (ненормализованными) способами:

$$123.5678 \cdot 10^5 = 12.35678 \cdot 10^6 = 1.235678 \cdot 10^7 = 0.1235678 \cdot 10^8.$$

Формат машинного изображения чисел с плавающей запятой должен включать знаковые поля (мантиссы и порядка), поле мантиссы и поле порядка числа и иметь следующий вид:

| ± | мантисса $m_A$ | ± | порядок $p_A$ |
|---|---|---|---|

Для данного формата разрядной сетки можно записать следующий диапазон представления чисел:

| ± | , | 1 | 1 | . | . | . | 1 | ± | 1 | . | . | . | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

$$A_{\max} = (1 - 2^{-n})\, 2^{(2^q - 1)}$$

| ± | , | 1 | 0 | . | . | . | 0 | ± | 1 | . | . | . | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

$$A_{\min} = 2^{-2^q}$$

В языках высокого уровня используется следующее представление чисел с плавающей запятой:

(знак)(мантисса)$E$(знак)(порядок)

Например, $-5.35E-2$ означает число $-5.35 \cdot 10^{-2}$. Такое представление называется научной нотацией.

В зависимости от типа данных числа с плавающей запятой в памяти ЭВМ хранятся в одном из следующих трех форматов:

– одинарной точности;

– двойной точности;
– расширенной точности.

Эти числа занимают в памяти, соответственно, 4, 8 или 10 байт. Для упрощения операций над порядками применяют представление чисел с плавающей запятой со смещенным порядком: $p' = p + N$, где $N$ – целое положительное число (смещение). Обычно $N = 2^k - 1$, где $k$– число двоичных разрядов в поле цифр несмещенного порядка. В этом случае поле знака порядка избыточно, так как $p'$ всегда положительно. Такие смещенные порядки называют *характеристиками*. Поле характеристики – это степень числа 2, на которую умножается мантисса, плюс смещение, равное 127 для одинарной точности, 1023 – для двойной точности и 16383 – для расширенной точности.


В любом из форматов старший (знаковый) бит определяет знак вещественного числа:

– 0 – положительное число;
– 1 – отрицательное число.

Все равные по абсолютному значению положительные и отрицательные числа отличаются только этим битом. В остальном числа с разным знаком полностью симметричны. Для представления отрицательных чисел здесь не используется дополнительный код, как это сделано для чисел с фиксированной запятой. В поле мантиссы содержится мантисса нормализованного числа. Для двоичного представления чисел, если целая часть мантиссы числа равна единице, то число с плавающей запятой называется нормализованным.

Так как для нормализованного двоичного числа целая часть всегда равна единице, то эту единицу можно не хранить. В форматах одинарной и двойной точности целая часть мантиссы не хранится. Таким образом, экономится один бит памяти.

Все операции с числами выполняются в формате с расширенной точностью. В этом формате хранится и «лишний» бит целой части нормализованного числа. Основная причина использования для вычислений расширенной точности – предохранение программы от возможной потери точности вычислений, связанной с большими различиями в порядках чисел, участвующих в арифметических операциях.

Для того, чтобы определить абсолютное значение числа с плавающей запятой, можно воспользоваться следующими формулами:

– одинарная точность: $1.(\text{цифры мантиссы}) \cdot 2^{(\text{характеристика} - 127)}$;
– двойная точность: $1.(\text{цифры мантиссы}) \cdot 2^{(\text{характеристика} - 1023)}$;
– расширенная точность: $1.(\text{цифры мантиссы}) \cdot 2^{(\text{характеристика} - 16383)}$.

Приведем конкретный пример машинного представления числа, например, −0,875 в формате с одинарной точностью, которое в двоичном виде выглядит следующим образом:

1 01111110 11000000000000000000000

Для этого числа знаковый бит равен 1 (отрицательное число), расширенный порядок (характеристика) равен 126, мантисса – 11 (в двоичной системе счисления). Значение этого числа вычисляется:

$1.11 \cdot 2^{(126-127)} = -1.75 \cdot 2^{-1} = -0{,}875$

Машинное представление числа −0,875 в разрядных сетках с двойной и расширенной точностью соответственно имеет следующий вид:

1 01111111110 1100000000000000000 . . . 00000000 (64 бита)

1 011111111111110 1110000000000000 . . . 00000000 (80 бит)

В стандарте IEEE крайние значения порядка (характеристики) зарезервированы и не используются для представления обычных чисел. Рассмотрим некоторые особые случаи представления вещественных чисел.

*Нуль* может иметь положительный или отрицательный знаки, которые игнорируются в операциях сравнения. Таким образом, имеется два нуля – положительный и отрицательный.

*Наименьшее положительное число* – это число, которое имеет нулевой знаковый бит, значение порядка, равное единице, и значение мантиссы, равное нулю. В зависимости от представления наименьшее положительное число имеет следующие значения: $1{,}17 \cdot 10^{-38}$ (одинарная точность), $2.23 \cdot 10^{-308}$ (двойная точность), $3.37 \cdot 10^{-4932}$ (расширенная точность).

*Наибольшее отрицательное число* полностью совпадает с наименьшим положительным числом, но имеет бит знака, установленный в единицу.

*Наибольшее положительное число* – это число, которое имеет нулевой знаковый бит, поле порядка, в котором все биты кроме самого младшего, равны единице, и содержит единицы во всех разрядах мантиссы. В зависимости от представления наибольшее положительное число имеет следующие значения: $3.37 \cdot 10^{38}$ (одинарная точность), $1.67 \cdot 10^{308}$ (двойная точность), $1.2 \cdot 10^{4932}$ (расширенная точность).

*Наименьшее отрицательное число* полностью совпадает с наибольшим положительным числом, но имеет бит знака, установленный в единицу.

*Положительная и отрицательная бесконечность* – это число содержит все единицы в поле порядка и все нули в поле мантиссы. В зависимости от состояния знакового бита может быть положительная и отрицательная бесконечности. Бесконечность может получиться, например, как результат деления конечного числа на нуль.

*Нечисло* содержит все единицы в поле порядка и любое значение в поле мантиссы. Нечисло может возникнуть в результате выполнения неправильной операции при замаскированных особых случаях.

*Неопределенность* содержит в поле порядка все единицы, а в поле мантиссы – число 100...0 (для одинарной и двойной точности) или 1100...0 (для расширенной точности, так как в этом формате хранится старший бит мантиссы).

Отмеченные особые случаи могут быть наглядно представлены следующим образом:

| | | | |
|---|---|---|---|
| 0 | 00...00 | 0...0 | положительный нуль |
| 1 | 00...00 | 0...0 | отрицательный нуль |
| 0 | 00...01 | 0...0 | наименьшее положительное число |
| 1 | 00...01 | 0...0 | наибольшее отрицательное число |
| 0 | 11...10 | 1...1 | наибольшее положительное число |
| 1 | 11...10 | 1...1 | наименьшее отрицательное число |
| 0 | 11...11 | 0...0 | положительная бесконечность |
| 1 | 11...11 | 0...0 | отрицательная бесконечность |
| 1 | 11...11 | *x...x* | нечисло |
| 1 | 11...11 | 10...0 | неопределенность |

При выполнении арифметических операций над числами с плавающей запятой может получаться результат, выходящий за пределы диапазона представления чисел, при этом выход за правую границу диапазона принято называть переполнением порядка (получение очень большого числа), а выход за левую границу – исчезновением порядка (потерей порядка) – получение очень малого числа, близкого к нулю.

### 2.11. Погрешность выполнения арифметических операций

Выбор длины разрядной сетки и формы представления чисел тесно связан с точностью получаемых при арифметических операциях результатов [1]. При выполнении операций над числами с фиксированной запятой можно считать, что результат точен (при условии отсутствия переполнения).

При выполнении операций над числами, представленными в форме с плавающей запятой, требуется выравнивание порядков. Это может приводить к потере некоторых разрядов мантиссы.

Рассмотрим арифметические операции над операндами, заданными с абсолютными погрешностями: $A = [A] + \Delta A$ и $B = [B] + \Delta B$, здесь $A$ и $B$ – точное значение числа, $[A]$ и $[B]$ – приближенное их значение, $\Delta A$ и $\Delta B$ – абсолютная погрешность. Абсолютной погрешностью приближения называют обычно некоторую величину, определенную как:

$$\Delta A = |A - [A]|.$$

В результате выполнения арифметических операций получаем:

$$A + B = [A] + [B] + (\Delta A + \Delta B),$$

где абсолютная погрешность суммы $\Delta(A + B) = \Delta A + \Delta B$.

$$A - B = [A] - [B] + (\Delta A - \Delta B),$$

где абсолютная погрешность разности $\Delta(A - B) = \Delta A - \Delta B$.

$$A \cdot B = [A][B] + [A]\Delta B + [B]\Delta A + \Delta A \Delta B.$$

Произведением $\Delta A \Delta B$ можно пренебречь, следовательно,

$$A \cdot B = [A][B] + [A] \cdot \Delta B + [B] \cdot \Delta A,$$

т. е. абсолютная погрешность произведения $\Delta(A \cdot B) \approx [A]\Delta B + [B]\Delta A$.

При выполнении операции деления

$$\frac{A}{B} = \frac{[A]+\Delta A}{[B]+\Delta B} = \frac{[A]+\Delta A}{[B]}\left(\frac{1}{1+\Delta B/[B]}\right)$$

абсолютная погрешность частного $\Delta(A/B) = \Delta A/[B] - [A]\Delta B/([B])^2$.

Отношение абсолютной погрешности к приближенному значению называется относительной погрешностью:

$$\delta A = \frac{\Delta A}{[A]}.$$

Для сложения

$$\delta_{A+B} = \frac{[A]\delta A + [B]\delta B}{[A] + [B]}.$$

Для умножения

$$\delta_{A \cdot B} = \frac{\Delta A}{[A]} + \frac{\Delta B}{[B]} = \delta A + \delta B.$$

Для деления

$$\delta_{A/B} = \frac{\Delta A}{[A]} - \frac{\Delta B}{[B]} = \delta A - \delta B.$$

### 2.12. Округление

Рассмотрим округление только дробных чисел, целые не округляются. Так как в ЭВМ используются числа с конечным числом разрядов, а также не редко выполняются операции приведения данных одной размерности к данным другой, то операция округления выполняется часто.

В общем виде число с плавающей запятой, размещенное в разрядной сетке, имеет вид $A_r = \pm m_a r^{\pm k}$. Если для записи мантиссы используются только $n$ разрядов, то число может быть представлено в виде двух частей: $A_r = [m_a] r^n + [A_0] r^{k-n}$, где $[A_0] r^{k-n} = A_0$ – часть числа, не вошедшая в разрядную сетку размерностью $k$.

В зависимости от того, как учитывается $A_0$ при записи числа $A$ в $n$-разрядную сетку, можно выделить несколько способов округления чисел [1].

1. *Отбрасывание* $A_0$. При этом возникает относительная погрешность $\delta_{\text{окр}} = |A_0| r^{k-n} / \left(|m_A| r^n\right)$, так как $r^{-1} \le |m_A| < 1$, $0 \le |A_0| < 1$, то $\delta_{\text{окр}} = r^{-(n-1)}$.

2. *Симметричное округление*. При этом производится анализ величины $A_0$:

$$[A] = \begin{cases} [m_A] r^n, & \text{если } |A_0| < r^{-1}; \\ [m_A] r^n + r^{k-n}, & \text{если } |A_0| \ge r^{-1}. \end{cases}$$

При условии $|A_0| \geq r^{-1}$ единица добавляется к младшему разряду мантиссы. Данный способ округления наиболее часто используется на практике.

3. *Округление по дополнению*. В этом случае для округления используется $(n + 1)$-й разряд. Если в нем находится единица, то она передается в $n$-й разряд, иначе разряды начиная с $(n + 1)$-го отбрасываются.

4. *Случайное округление*. Генератор случайных чисел формирует нулевое или единичное значение, посылаемое в младший разряд мантиссы.

Оценка точности вычислений зависит как от вида выполняемых операций, так и от последовательности их следования друг за другом.

### 2.13. Нормализация чисел

Нормализация – процесс, относящийся к числам, записанным в форме с плавающей запятой [1].

Число называется нормализованным, если его мантисса удовлетворяет условию $r^{-1} \leq |M_A| < 1$. В процессе выполнения арифметических операций мантисса результата может не удовлетворять этому условию. Возникает денормализация мантиссы. Возможны два случая денормализации – вправо (например, при вычитании или делении) и влево (при сложении или умножении). В этом случае необходимо выполнить действия для приведения его мантиссы в соответствии с указанным условием, т. е. произвести нормализацию.

Например, число $A$ = 0,00101…1 – денормализованное (признак нарушения нормализации вправо). Для нормализации число нужно сдвинуть в сторону, противоположную направлению нарушения нормализации. Таким образом, в примере мантиссу числа $A$ необходимо сдвинуть влево на два разряда. При этом порядок необходимо уменьшить на два. Таким образом, действия, выполняемые над мантиссами и порядками, различны. Следовательно должны быть два устройства для раздельной обработки мантисс и порядков.

Различают два вида сдвигов: простой и модифицированный.

Простой сдвиг – сдвиг, выполняемый по правилу:

| Исходная комбинация | Сдвиг влево | Сдвиг вправо |
|---|---|---|
| $0,a_1a_2\ldots a_n$ | $a_1,a_2\ldots a_n0$ | $0,0a_1a_2\ldots a_{n-1}$ |
| $1,a_1a_2\ldots a_n$ | $a_1,a_2\ldots a_n\alpha$ | $0,1a_1a_2\ldots a_{n-1}$ |

Модифицированный сдвиг − сдвиг, при котором в сдвигаемый разряд заносится значение, совпадающее со значением знакового разряда.

| Исходная комбинация | Сдвиг влево | Сдвиг вправо |
|---|---|---|
| $00,a_1a_2\ldots a_n$ | $0a_1,a_2\ldots a_n0$ | $00,0a_1a_2\ldots a_{n-1}$ |
| $01,a_1a_2\ldots a_n$ | $1a_1,a_2\ldots a_n0$ | $00,1a_1a_2\ldots a_{n-1}$ |
| $10,a_1a_2\ldots a_n$ | $0a_1,a_2\ldots a_n\alpha$ | $11,0a_1a_2\ldots a_{n-1}$ |
| $11,a_1a_2\ldots a_n$ | $1a_1,a_2\ldots a_n\alpha$ | $11,1a_1a_2\ldots a_{n-1}$ |

Величина α определятся в зависимости от вида кодирования числа. Для чисел в прямом и дополнительном кодах α = 0, а для обратного кода α = 1.

Нарушение нормализации вправо может быть более глубоким при вычитании, например, одного числа из другого, если они близки по величине.

### 2.14. Последовательное и параллельное сложение чисел

Параллельный способ передачи информации является более быстродействующим по сравнению с последовательным, но менее экономичным (для $n$-разрядного числа требуется вместо одного проводника $n$ проводников).

В соответствии со способом приема/передачи информации устройства, обрабатывающие эту информацию, могут быть либо параллельного, либо последовательного действия. В литературе описано множество устройств выполняющих сложение двоичных чисел [10]. Рассмотрим самые простые из них.

Двоичный сумматор – устройство, предназначенное для выполнения арифметического сложения чисел в двоичном коде. Простейший случай – это суммирование двух одноразрядных чисел.

**Последовательный сумматор.** Для последовательного выполнения операции сложения (разряд за разрядом) используется один полный одноразрядный сумматор (рис. 4). Он имеет два входа ($a$ и $b$) на каждый из которых последовательно, начиная с младшего, подаются разряды слагаемых (по одному за такт). С выхода снимается значение соответствующего разряда суммы. Последнее определяется не только текущими значениями разрядов слагаемых, но и переносом из предыдущего разряда $C_{in}$.

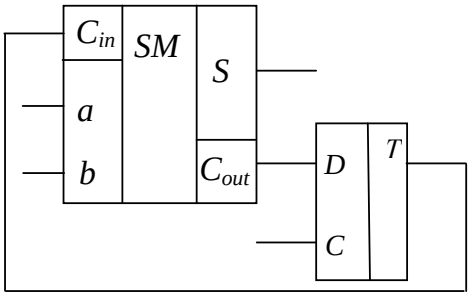



Рис. 4. Схема последовательного суммирования

Время, затрачиваемое на сложение $n$-разрядных чисел, в этом случае равно $T_{\text{сл(посл)}} \cong n\Delta t$, где $\Delta t$ – время формирования сигналов переноса и суммы.

В этой схеме на входы $a$ и $b$ последовательно подаются попарно разряды слагаемых $a_i$ и $b_i$. $C_{in}$ и $C_{out}$ – соответственно перенос из предыдущего (младшего) разряда и перенос в следующий (старший) разряд. На выходе $S$ формируются $s_i$ разряды суммы. Триггер введен в схему для хранения значения переноса до следующего такта (следующей пары разрядов).

**Параллельный сумматор.** Более быстродействующим будет параллельный сумматор с последовательным переносом. Для примера рассмотрим четырехразрядный параллельный сумматор с последовательным переносом (рис. 5).

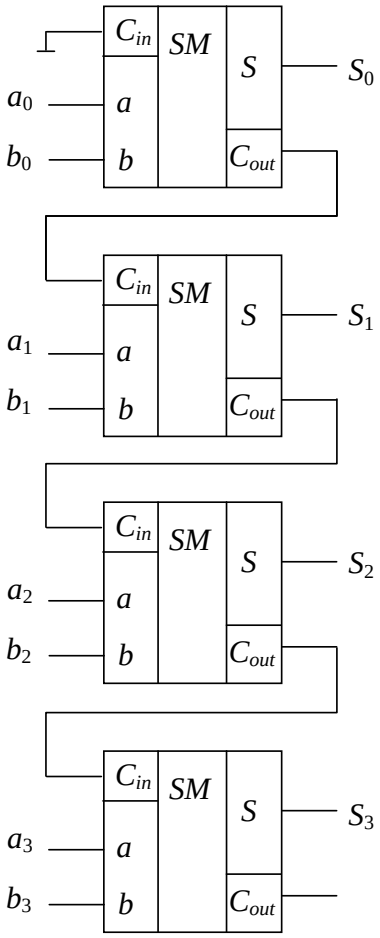



Рис. 5. Схема параллельного суммирования

Для каждого разряда в этой схеме используется отдельный полный одноразрядный сумматор. В младший разряд ($a_0$, $b_0$) переноса нет, поэтому $C_{in} = 0$. На каждый последующий разряд подается перенос из предыдущего. Хоть сумматор и называется параллельным, на самом деле все разряды обрабатываются не одновременно, а только после формирования переноса для данного разряда. Отсюда следует, что быстродействие устройства определяется суммой задержек передачи сигнала переноса с младшего разряда на вход сумматора старшего разряда.

$T_{\text{сл(пар)}} \cong n\tau_{\text{лэ}}$.

Очевидно, $T_{\text{сл(посл)}} \geq T_{\text{сл(пар)}}$, так как $\Delta t \cong (3,6)k\tau_{\text{лэ}}$, где $k$ – коэффициент запаса, обеспечивающий полное окончание всех переходных процессов в сумматоре последовательного действия $k \in [1{,}2;\ 1{,}3]$.

Недостатком такого параллельного суммирования является большое время распространения сигналов переноса $P_i$. Параллельные безрегистровые сумматоры обеспечивают наибольшую скорость суммирования, если снабжены схемой ускоренного переноса.

### 2.15. Арифметические действия над числами с плавающей запятой

**Сложение (вычитание) чисел с плавающей запятой.** При сложении (вычитании) чисел складываемые цифры (разряды) должны иметь одинаковый вес. Это требование выполняется, если складываемые (вычитаемые) числа имеют одинаковые порядки. Пусть имеются два числа с плавающей запятой:

$$A = \pm m_A r^{\pm p_A},$$

$$B = \pm m_B r^{\pm p_B}.$$

Алгоритм сложения (вычитания) чисел с произвольными знаками состоит в следующем.

1. Произвести сравнение порядков $p_A$ и $p_B$. Для этого из порядка числа $A$ вычитается порядок числа $B$. Разность $p = p_A - p_B$ указывает, на сколько разрядов требуется сдвинуть вправо мантиссу числа с меньшим порядком. Если $p = p_A - p_B > 0$, то $p_A > p_B$ и для выравнивания порядков необходимо сдвинуть вправо мантиссу $M_B$. Если $p = p_A - p_B < 0$, то $p_B > p_A$ и для выравнивания порядков необходимо сдвинуть вправо мантиссу $M_A$. Если $p = p_A - p_B = 0$, то $p_A = p_B$ и порядки слагаемых выравнивать не требуется.

2. Выполнить сдвиг соответствующей мантиссы на один разряд, повторяя его до тех пор, пока $p \neq 0$.

3. Выполнить сложение (вычитание) мантисс $M_A$ и $M_B$ по правилу сложения (вычитания) правильных дробей.

4. Если при сложении (вычитании) мантисс произошло переполнение, то произвести нормализацию результата путем сдвига мантиссы вместе со знаковым разрядом вправо на один разряд с увеличением порядка на единицу. Если же произошла денормализация, то выполнить сдвиг мантиссы результата на соответствующее количество разрядов влево с соответствующим уменьшением порядка суммы.

5. Конец алгоритма.

*Пример.* $M_A = -0{,}10110 \quad p_A = +0111$

$M_B = -0{,}11011 \quad p_B = +0101$

$[M_A]_{доп} = 1{,}01010$

$[M_B]_{доп} = 1{,}00101$

$$p = [p_A]_{доп} + [-p_B]_{доп} = \begin{array}{r} 0.0111 \\ +\ \underline{1.1011} \\ 1\ 0.0010 \end{array} > 0$$

Так как $[p_A - p_B]_{доп} > 0$, то сдвигу подвергается мантисса $M_B$.

В рассматриваемом примере при каждом сдвиге мантиссы на один разряд из положительной разности порядков производим последовательное вычитание единицы до тех пор, пока в результате не будет получен ноль. При этом выполняется анализ разности порядков на каждом шаге. Если она отлична от нуля, то производится очередной сдвиг соответствующей мантиссы. В случае если разность $[p_A - p_B]_{доп} < 0$, необходимо либо прибавлять единицу до нулевого результата, либо изменить знак разности на противоположный и, как и выше, выполнять вычитание единицы.

$[M_B]_{доп} = 1{,}00101$ $\qquad$ $0.0010$

$[-1]_{доп} = \underline{1.1111}$

$[M_B]_{доп} = 1{,}\mathbf{1}0010\ 1$ $\qquad$ $0.0001 \neq 0$

$[M_B]_{доп} = 1{,}\mathbf{11}001\ 01$ $\qquad$ $[-1]_{доп} = \underline{1.1111}$

$0.0000 = 0$

$[M_B]_{доп} = 1{,}11001\ 01$

$[M_A]_{доп} = \underline{1{,}01010}$

$[M_{A+B}]_{доп} = 11{,}00011\ 01$ $\qquad$ $p_{A+B} = \max(p_A, p_B) = p_A = +0111$

Полученный результат нормализован. После выполнения операции округления мантиса результата будет иметь вид $[M_{A+B}]_{доп} = 1{,}00011$.

**Умножение чисел с плавающей запятой.** При умножении чисел их мантиссы перемножаются, а порядки складываются. В арифметико-логическом устройстве ЭВМ операция умножения реализуется в виде следующей последовательности действий:

1. мантиссы, представленные в прямом коде, перемножаются;
2. порядки складываются с применением обратного или дополнительного кодов;
3. знак произведения формируется используя операцию суммы по модулю два.

**Деление чисел с плавающей запятой.** При выполнении деления чисел с плавающей запятой мантиссу делимого делят на мантиссу делителя, а из порядка делимого вычитают порядок делителя. Знак частного формируется так же, как и для произведения.

### 2.16. Машинные методы умножения чисел в прямых кодах

Операцию умножения можно разложить на ряд последовательных сложений. Сложением управляют разряды множителя: если в очередном разряде множителя содержится единица, то к сумме добавляется множимое. При этом, в зависимости от метода умножения, выполняется сдвиг либо множимого, либо частичной суммы. Наряду с этим умножение можно начинать как с младших, так и со старших разрядов множителя. Для умножения используют модули сомножителей. Знак произведения определяется сложением по модулю два знаковых разрядов сомножителей.

Введем некоторые обозначения, используемые ниже: Мн и Мт – соответственно множимое и множитель, $\Pi_i^{ч}$ – $i$-е частичное произведение, $\Sigma_i^{ч}$ – $i$-я частичная сумма.

Ниже приводится схема четырех алгоритмов умножения (рис. 6). В схеме четыре алгоритма разбиты на две группы по признаку – начала умножения с младших или старших разрядов Мт. Обе эти группы, в свою очередь, содержат по два алгоритма умножения, каждый из которых характеризуется сдвигом (вправо или влево) частичной суммы или произведения.

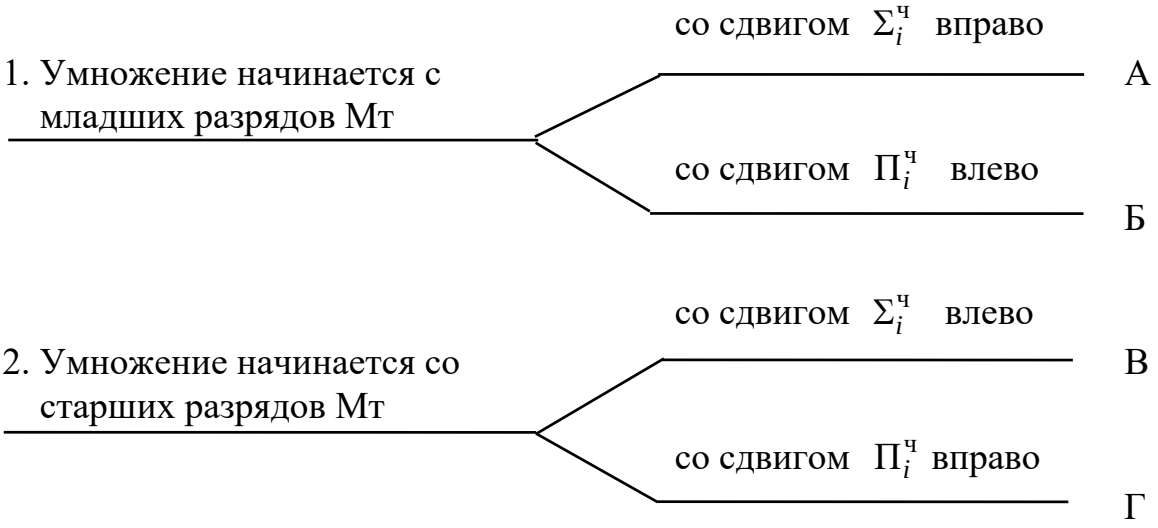



Рис. 6. Схема алгоритмов умножения

Остановимся более подробно на реализации умножения согласно *алгоритму А.*

Представим $\text{Мн} = A = 0, a_1 a_2 \ \ldots \ a_n$

$$\text{Мт} = B = 0, b_1 b_2 \ \ldots \ b_n = b_1 \cdot 2^{-1} + b_2 \cdot 2^{-2} + \ldots + b_n \cdot 2^{-n}$$

$$\text{Мн} \cdot \text{Мт} = A \cdot B = 0, a_1 a_2 \ \ldots \ a_n \left( b_1 \cdot 2^{-1} + b_2 \cdot 2^{-2} + \ldots + b_n \cdot 2^{-n} \right) =$$

$$= 0 + \left(b_1 \cdot 0, a_1 \ \ldots \ a_n\right) \cdot 2^{-1} + (b_2 \cdot 0, a_1 \ \ldots \ a_n) \cdot 2^{-2} + \ldots + \left(b_n \cdot 0, a_1 \ \ldots \ a_n\right) \cdot 2^{-n} =$$

$$= 0 + b_1 \cdot A \cdot 2^{-1} + b_2 \cdot A \cdot 2^{-2} + \ \ldots \ + b_{n-1} \cdot A \cdot 2^{-(n-1)} + b_n \cdot A \cdot 2^{-n} =$$

$$= 0 + b_n \cdot A \cdot 2^{-n} + b_{n-1} \cdot A \cdot 2^{-(n-1)} + \ \ldots \ + b_2 \cdot A \cdot 2^{-2} + b_1 \cdot A \cdot 2^{-1} =$$

$$= \left( (\ldots \left( \left(0 + b_n \cdot A\right) \cdot 2^{-1} + b_{n-1} \cdot A \right) \cdot 2^{-1} + \ \ldots \ + b_2 \cdot A) \cdot 2^{-1} + b_1 \cdot A \right) \cdot 2^{-1}.$$

В этом выражении (и следующих трех), умножение на $2^{-i}$ означает сдвиг вправо на *i* разрядов, а на $2^i$ – влево на *i* разрядов соответственно.

**Алгоритм А.**

1. Обнуляем содержимое сумматора результата $\Sigma_i^{ч} = 0$, $i = 0$.

2. Формируем очередное частичное произведение $\Pi_i^{ч} = b_{n-i} \cdot A$.

3. Формируем новую частичную сумму $\Sigma_i^{ч} = \Sigma_{i-1}^{ч} + \Pi_i^{ч}$.

4. Выполняем сдвиг полученной частичной суммы вправо на один разряд $\Sigma_i^{ч} \cdot 2^{-1}$. При этом содержимое младшего разряда сумматора переносится в освободившийся старший разряд регистра множителя, в котором тоже выполняется сдвиг всех разрядов на один вправо.

5. Увеличиваем счетчик *i* числа выполненных умножений на единицу. Сравниваем его с *n*. Если они равны (выполнено умножение на все разряды множителя), то переходим к действию 6, иначе возвращаемся к действию 2.

6. Получено искомое произведение $\text{Мн} \cdot \text{Мт}$. Старшие разряды произведения находятся в сумматоре, младшие − в регистре Мт.

7. Конец алгоритма.

Ниже приведены (без вывода) остальные реализации трех других алгоритмов умножения (Б, В и Г):

$$\text{Мн}\cdot\text{Мт} = A\cdot B = 0 + b_n\cdot A + b_{n-1}\cdot A\cdot 2^1 + \ldots + b_1\cdot A\cdot 2^{n-1} \quad \text{(алгоритм Б)}$$

$$\text{Мн}\cdot\text{Мт} = A\cdot B = \left(\ldots\left(0 + b_1\cdot A\right)\cdot 2^1 + b_2\cdot A\right)\cdot 2^1 + \ldots + b_n\cdot A) \quad \text{(алгоритм В)}$$

$$\text{Мн}\cdot\text{Мт} = A\cdot B = 0 + b_1\cdot A\cdot 2^{-1} + b_2\cdot A\cdot 2^{-2} + \ldots + b_n\cdot A\cdot 2^{-n} \quad \text{(алгоритм Г)}$$

Данные три алгоритма похожи на алгоритм А с тем отличием, что сдвигу подвергается не $\Sigma_i^{\text{ч}}$, а $\Pi_i^{\text{ч}}$ (алгоритмы Б и Г), и сдвиг производится не вправо ($2^{-i}$), а влево ($2^i$) (алгоритмы Б и В). Эти алгоритмы предлагается разработать самостоятельно.

Рассмотрим пример умножения чисел согласно алгоритму А.

*Пример.* Мн = 0,1011

Мт = 0,1101

$b_1 \ldots b_4$

| | | |
|---|---|---|
| 0,0000 | $\Sigma_0^{\text{ч}}$ | начальное содержимое сумматора |
| $+$ <u>0,1011</u> | $\Pi_1^{\text{ч}} = \text{Мн}\cdot b_4$ | первое частичное произведение |
| 0,1011 | $\Sigma_1^{\text{ч}}$ | первая частичная сумма |
| 0,**0**101 1 | $\Sigma_1^{\text{ч}}\cdot 2^{-1}$ | сдвиг первой частичной суммы |
| $+$ <u>0,0000</u> | $\Pi_2^{\text{ч}} = \text{Мн}\cdot b_3$ | второе частичное произведение |
| 0,0101 1 | $\Sigma_2^{\text{ч}}$ | вторая частичная сумма |
| 0,**0**010 11 | $\Sigma_2^{\text{ч}}\cdot 2^{-1}$ | сдвиг второй частичной суммы |
| $+$ <u>0,1011</u> | $\Pi_3^{\text{ч}} = \text{Мн}\cdot b_2$ | третье частичное произведение |
| 0,1101 11 | $\Sigma_3^{\text{ч}}$ | третья частичная сумма |
| 0,**0**110 111 | $\Sigma_3^{\text{ч}}\cdot 2^{-1}$ | сдвиг третьей частичной суммы |
| $+$ <u>0,1011</u> | $\Pi_4^{\text{ч}} = \text{Мн}\cdot b_1$ | четвертое частичное произведение |
| 1,0001 111 | $\Sigma_4^{\text{ч}}$ | возникло переполнение |
| 0,**1**000 1111 | $\Sigma_4^{\text{ч}}\cdot 2^{-1}$ | Мн·Мт (сдвиг частичной суммы) |

Заметим, что при умножении чисел по алгоритму А на отдельных этапах операции возможно переполнение (попадание значащей единицы в знаковый разряд). Однако при последующем сдвиге переполнение устраняется. При использовании других алгоритмов (Б, В, Г) переполнения не возникает. Структурные схемы устройств для выполнения умножения по рассмотренным алгоритмам приведены на рис. 7.

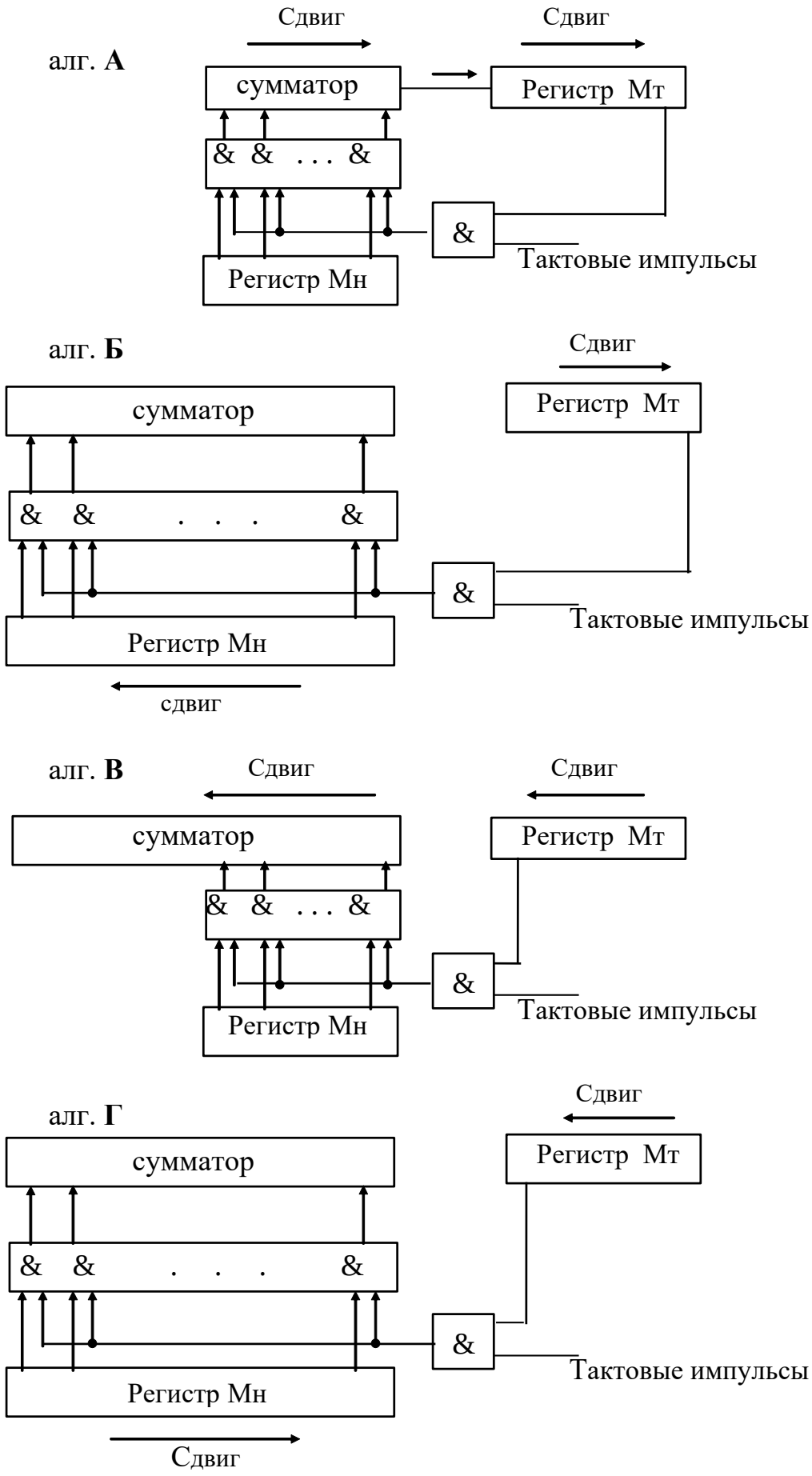



Рис. 7. Структурные схемы устройств умножения по алгоритмам А, Б, В и Г

В этих структурных схемах элемент, расположенный между сумматором и регистром множимого, и построенный на логических элементах & – умножитель на один разряд множителя. На его выходах и формируется частичное произведение $П_i^ч$. Также надо отметить что в алгоритмах Б и Г сдвиг $П_i^ч$ осуществляется до его формирования за счет сдвига множимого. При этом формируемое $П_i^ч$ получается также сдвинутое. Для синхронизации процесса умножения в схему введен элемент «логическое И» (&), на входы которого подаются тактовые импульсы и значение разряда множителя участвующего в формировании очередного $П_i^ч$. При единичном тактовом сигнале и единичном разряде Мт в сумматор поступает частичное произведение, равное Мн, при нулевом разряде Мт – нулевое произведение.

Время умножения чисел $t_{умн} = (t_{сл} + t_{сдв}) \cdot n$, где $n$ – число разрядов Мт. Сдвиг и сложение нельзя выполнять одновременно. Это наглядно показано на временной диаграмме, изображенной на рис. 8.

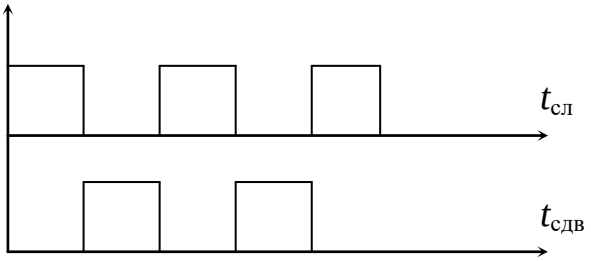



Рис. 8. Временные диаграммы умножения

### 2.17. Ускорение операции умножения

Арифметические операции, к числу которых относится умножение, часто встречаются при решении задач на ЭВМ. Умножение является длинной операцией. Временные затраты на умножение чисел в прямых кодах можно оценить по формуле

$$T_{умн} = \sum_{i=1}^{n} (p_i t_{сл} + t_{сдв}), \qquad (3)$$

где $p_i$ – вероятность появления единицы в разрядах множителя; $t_{сл}$ – время формирования очередной частичной суммы; $t_{сдв}$ – время выполнения сдвига числа на один разряд; $n$ – число тактов умножения (разрядов множителя).

Анализируя выражение (3), можно предложить различные пути сокращения величины $T_{умн}$ – уменьшение времени на сдвиг, на формирование очередной суммы, уменьшение числа разрядов множителя. Этого можно достигнуть логическими или аппаратными методами. Рассмотрим логические методы ускорения умножения. Один из наиболее простых способов состоит в том, чтобы при наличии нулевого разряда в множителе не выполнять формирование очередного (нулевого) частичного произведения, не изменяющего содержимое сумматора. В зависимости от алгоритма умножения выполняется сдвиг либо частичной суммы, либо частичного произведения, не выполняя суммирования.

### 2.17.1. Умножение с хранением переносов

При сложении *n*-разрядных двоичных чисел разряды суммы формируются не одновременно, а друг за другом, по мере того как формируется и передается в следующий разряд очередной перенос. При этом максимальное время сложения составит $n \cdot t_{сл}$, где $t_{сл}$ – время срабатывания одноразрядного двоичного сумматора, на выходе которого формируются сигналы суммы и переноса.

В то же время если исключить необходимость выполнения межразрядных переносов при сложении, то время умножения уменьшится в *n* раз. Переносы, формируемые при сложении, будут записываться в отдельный регистр. Содержимое этого регистра добавляется в сумматор в следующем такте вместе с очередным частичным произведением. При этом сложение может выполняться паралельно по всем разрядам, не дожидаясь прохождения переноса. На последнем такте (при умножении на последний разряд множителя) сложение выполняется с учетом межразрядных переносов. В заключение следует отметить, что этот метод используется *только* с алгоритмом А.

*Пример.* Мн = 0,1111
Мт = 0,1011

| | |
|---|---|
| 0,0000 | $\Sigma_0^{ч}$ |
| 0,0000 | регистр переносов |
| $^{+}$ 0,1111 | $\Pi_1^{ч} = Мн \cdot b_4$ |
| 0,1111 | $\Sigma_1^{ч}$ |
| 0,0000 | регистр переносов |
| 0,**0**111 1 | $\Sigma_1^{ч} \cdot 2^{-1}$ |
| $^{+}$ 0,1111 | $\Pi_2^{ч} = Мн \cdot b_3$ |
| 0,1000 1 | $\Sigma_2^{ч}$ |
| 0,0111 | регистр переносов |
| 0,**0**100 01 | $\Sigma_2^{ч} \cdot 2^{-1}$ |
| $^{+}$ 0,0000 | $\Pi_3^{ч} = Мн \cdot b_2$ |
| 0,0011 01 | $\Sigma_3^{ч}$ |
| 0,0100 | регистр переносов |
| 0,**0**001 101 | $\Sigma_3^{ч} \cdot 2^{-1}$ |
| $^{+}$ 0,1111 | $\Pi_4^{ч} = Мн \cdot b_1$ |
| 1,0100 101 | $\Sigma_4^{ч}$ (возникло переполнение) |
| 0,**1**010 0101 | $\Sigma_4^{ч} \cdot 2^{-1}$ (Мн · Мт ) |

### 2.17.2. Умножение на два разряда множителя одновременно

Разбиение множителя на группы длиной *k*-разрядов означает переход к новой системе счисления с основанием $2^k$. Если при этом удается сократить количество элементарных действий, выполняемых в каждом такте умножения

(сложение и сдвиги), то число тактов (время) умножения сокращается в $k$ раз. Остановимся более подробно на умножении на два разряда множителя за один такт ($k = 2$). Выполним анализ пар разрядов множителя.

Возможны четыре случая сочетания разрядов множителя: 00, 01, 10, 11. Умножение на каждую из пар разрядов множителя должно выполняться за один такт автоматного времени, т. е. в каждом такте умножения должно выполняться не более одного сложения. Рассмотрим умножение на каждую из пар на примере алгоритма А.

В случае пары 00 необходимо выполнить только сдвиг частичной суммы на два разряда $-\Sigma_{i-1}^{\text{ч}} \cdot 2^{-2}$.

Для пары 01 выполняется добавление множимого в сумматор с последующим сдвигом суммы на два разряда $-(\Sigma_{i-1}^{\text{ч}} + \text{Мн})2^{-2}$.

При наличии пары 10 возможны следующие варианты действий:

а) $(\Sigma_{i-1}^{\text{ч}} + \text{Мн} + \text{Мн}) \cdot 2^{-2}$, т. е. в этом случае происходят два сложения, что противоречит требованию;

б) $(\Sigma_{i-1}^{\text{ч}} + 2\text{Мн}) \cdot 2^{-2}$, в этом случае требуется дополнительный регистр для хранения удвоенного множимого;

в) $(\Sigma_{i-1}^{\text{ч}} + \text{Мн} \cdot 2^{1}) \cdot 2^{-2}$, что соответствует добавлению к частичной сумме сдвинутого на один разряд влево множимого;

г) $(\Sigma_{i-1}^{\text{ч}} \cdot 2^{-1} + \text{Мн}) \cdot 2^{-1}$, т. е. частичная сумма сдвигается на один разряд вправо до и после добавления к ней множимого.

При умножении на пару 11 к частичной сумме необходимо добавить утроенное множимое. Пару 11 можно представить в виде $11 = (2^2 - 1)$.

$\text{Мн} \cdot 11 = \text{Мн} \cdot (2^2 - 1) = \text{Мн} \cdot 2^2 - \text{Мн}$, т. е. в текущем такте к частичной сумме добавляется множимое, взятое со знаком минус. Добавление $\text{Мн} \cdot 2^2$ реализуется путем увеличения на единицу следующей старшей пары разрядов Мн.

В табл. 1 представлены правила преобразования множителя для избыточной двоичной системы счисления $(0,\overline{1},1)$ в случае умножения, начиная с младших разрядов.

Таблица 1

| Анализируемая пара разрядов | Перенос из младшей пары | Преобразованная пара | Перенос в старшую пару |
|---|---|---|---|
| 00 | 0 | 00 | 0 |
| 01 | 0 | 01 | 0 |
| 10 | 0 | 10 | 0 |
| 11 | 0 | $0\overline{1}$ | 1 |
| 00 | 1 | 01 | 0 |
| 01 | 1 | 10 | 0 |
| 10 | 1 | $0\overline{1}$ | 1 |
| 11 | 1 | 00 | 1 |

*Пример.* Мн = 0101

Мт = 11000111

Мт разбиваем на пары и преобразуем согласно табл. 1.

$Мт^{п} = 010\bar{1}00100\bar{1}$ – преобразованный Мт.

Умножение будем осуществлять согласно алгоритму А.

$[-Мн]_{доп} = 1.1011$

$[2\,Мн]_{доп} = 0.1010$

| | |
|---|---|
| 0.0000 | $\Sigma_0^{ч}$ |
| $^{+}$ 1.1011 | $\Pi_1^{ч} = -Мн$ |
| 1.1011 | $\Sigma_1^{ч}$ |
| 1.**11**10 11 | $\Sigma_1^{ч} \cdot 2^{-2}$ |
| $^{+}$ 0.1010 | $\Pi_2^{ч} = 2Мн$ |
| 0.1000 11 | $\Sigma_2^{ч}$ |
| 0.**00**10 0011 | $\Sigma_2^{ч} \cdot 2^{-2}$ |
| 0.**00**00 100011 | $\Sigma_3^{ч} \cdot 2^{-2}$ $(\Sigma_2^{ч} \cdot 2^{-4}$ т. к. $\Pi_3^{ч} = 0)$ |
| $^{+}$ 1.1011 | $\Pi_4^{ч} = -Мн$ |
| 1.1011 100011 | $\Sigma_4^{ч}$ |
| 1.**11**10 11100011 | $\Sigma_4^{ч} \cdot 2^{-2}$ |
| $^{+}$ 0.0101 | $\Pi_5^{ч} = Мн$ |
| 0.0011 11100011 | $\Sigma_5^{ч}$ |
| 0.**00**00 1111100011 | $\Sigma_5^{ч} \cdot 2^{-2}$ |

Появление любой из рассматриваемых пар множителя равновероятно. Следовательно, время умножения на два разряда множителя может быть выражено следующим соотношением: $T_{умн}^{2\,разр} = (n/2 + 1)[0{,}75 \cdot (t_{сл} + t_{сдв}) + 0{,}25 \cdot t_{сдв}]$, где $n$ – количество разрядов множителя.

### 2.17.3. Умножение на четыре разряда одновременно

В этом случае с помощью приема, аналогичного приему, использованному в случае умножения на два разряда одновременно, можно рассматривать сразу тетраду двоичных разрядов. Может быть выведено общее правило сокращенного умножения:

- Если цифра множителя $b_{i-1} < r/2$, то $\Sigma_{i-1}^{ч} + \Pi_i^{ч}$, где $\Pi_i^{ч} = Мн \cdot b_i$;
- Если цифра множителя $b_{i-1} \geq r/2$, то $\Sigma_{i-1}^{ч} + \Pi_i^{ч}$, где $\Pi_i^{ч} = [Мн(r - b_i)]_{доп}$.

Схема формирования сомножителей при умножении на четыре разряда множителя приведена на рис. 9.

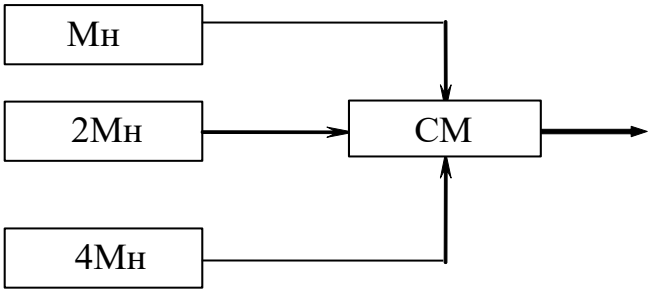



Рис. 9. Схема формирования сомножителей при умножении на четыре разряда

Анализ четырех двоичных разрядов одновременно дает возможность осуществить сдвиг на четыре разряда за один такт.

*Пример.* Мн = 011

Мт = *С*49

$Мт^{п} = 1\bar{4}5\bar{7}$

Можно заранее заготовить кратные множители: Мн, 2Мн, 4Мн, поместив их в дополнительные регистры. Это позволит сократить время, необходимое для формирования частичного произведения $\Pi_i^{ч}$, лежащего в пределах от нуля до семи множимых.

$[+7Мн]_{доп} = 0.10101$ $[+5Мн]_{доп} = 0.01111$ $[+4Мн]_{доп} = 0.01100$

$[-7Мн]_{доп} = 1.01011$ $[-4Мн]_{доп} = 1.10100$

| | |
|---|---|
| 0.00000 | $\Sigma_0^{ч}$ |
| + 1.01011 | $\Pi_1^{ч} = -7Мн$ |
| 1.01011 | $\Sigma_1^{ч}$ |
| 1.**1111**0 1011 | $\Sigma_1^{ч} \cdot 2^{-4}$ |
| + 0.01111 | $\Pi_2^{ч} = +5Мн$ |
| 0.01101 1011 | $\Sigma_2^{ч}$ |
| 0.**0000**0 1101 1011 | $\Sigma_2^{ч} \cdot 2^{-4}$ |
| + 1.10100 | $\Pi_3^{ч}$ |
| 1.10100 1101 1011 | $\Sigma_3^{ч}$ |
| 1.**1111**1 0100 1101 1011 | $\Sigma_3^{ч} \cdot 2^{-4}$ |
| + 0.00011 | $\Pi_4^{ч} = +Мн$ |
| 0.00010 0100 1101 1011 | $\Sigma_4^{ч}$ |
| 0.**0000**00010 0100 1101 1011 | $\Sigma_4^{ч} \cdot 2^{-4} = Мн \cdot Мт$ |

### 2.18. Умножение в дополнительных кодах

Умножение чисел в дополнительных кодах может выполняться по любому из рассмотренных выше четырех алгоритмов, но отличается тем, что

для получения верного результата необходимо вводить поправки. Умножение правильных дробей и целых чисел имеет некоторые различия.

Рассмотрим вначале **умножение дробных чисел**. Возможны четыре случая сочетания знаков сомножителей.

1. Мн > 0,
   Мт > 0.

В этом случае дополнительный код сомножителей совпадает с прямым кодом и умножение выполняется по правилам умножения в прямых кодах, в результате чего получается верный результат.

2. Мн > 0,
   Мт < 0.

Так как Мн и Мт имеют разные знаки, то результат будет иметь отрицательный знак. Следовательно, результат должен быть представлен в дополнительном коде $[Мн \cdot Мт]_{доп} = 2 - Мн \cdot Мт$. Для формирования произведения выполним умножение Мн на Мт′ (дополнение Мт):

$Мн \cdot Мт' = Мн \cdot (1 - Мт) = Мн - Мн \cdot Мт$.

Таким образом, погрешность умножения Δ равна разности $[Мн \cdot Мт]_{доп}$ и Мт · Мн′

$\Delta = 2 - \cancel{Мн \cdot Мт} - Мн + \cancel{Мн \cdot Мт} = 2 - Мн = [-Мн]_{доп}$

и должна быть внесена в полученный результат в качестве поправки.

*Пример.* Мн = 0,1011 (алгоритм умножения А)
Мт = − 0,1101
$\Delta = [-Мн]_{доп} = 1,0101$
$[-Мт]_{доп} = 1,0011$
$b_1 \ldots b_4$

| | | |
|---|---|---|
| 0,0000 | $\Sigma_0^ч$ | |
| + 0,1011 | $П_1^ч = Мн \cdot b_4$ | |
| 0,1011 | $\Sigma_1^ч$ | |
| 0,**0**101 1 | $\Sigma_1^ч \cdot 2^{-1}$ | |
| + 0,1011 | $П_2^ч = Мн \cdot b_3$ | |
| 1,0000 1 | $\Sigma_2^ч$ | (произошло переполнение) |
| 0,**1**000 01 | $\Sigma_2^ч \cdot 2^{-1}$ | |
| 0,**0**10 0001 | $\Sigma_3^ч \cdot 2^{-1}$ | $(\Sigma_2^ч \cdot 2^{-1}) \cdot 2^{-1}$, т. к. $П_3^ч = 0$ |
| 0,**0**010 0001 | $\Sigma_4^ч \cdot 2^{-1}$ | $((\Sigma_2^ч \cdot 2^{-1}) \cdot 2^{-1}) \cdot 2^{-1}$, т. к. $П_4^ч = 0$ |
| + 1,0101 | $\Delta = [-Мн]_{доп}$ | (поправка) |
| 1,0111 0001 | $\Sigma_5^ч = [Мн \cdot Мт]_{доп}$ | |
| − 0,1000 1111 | $Мн \cdot Мт$ | |

В этом примере, а также в некоторых примерах, следующих ниже, происходит переполнение разрядной сетки (только в случае использования алгоритма А),

в результате которого искажаются и знаковая и значащая часть числа. В этом случае после немодифицированного сдвига суммы вправо (согласно алгоритму) знаковый разряд сдвигается в значащую часть, а знаковый разряд меняет свое значение на противоположное.

3. $Мн < 0$,
   $Мт > 0$.

Аналогично предыдущему случаю

$[Мн \cdot Мт]_{доп} = 2 - Мн \cdot Мт$,

$Мн' \cdot Мт = (1 - Мн) \cdot Мт = Мт - Мн \cdot Мт$.

Таким образом, погрешность умножения равна разности $[Мн \cdot Мт]_{доп}$ и $Мт \cdot Мн'$ имеет следующий вид.

$\Delta = 2 - \cancel{Мн} \cdot Мт - Мт + \cancel{Мн} \cdot Мт = 2 - Мт = [-Мт]_{доп}$.

Использовать этот вариант неудобно, так как нужно вводить поправку $[-Мт]_{доп}$ в конце умножения, а в результате сдвигов Мт постепенно исчезает на регистре множителя и для поправки нужно либо вводить дополнительный регистр, либо вносить поправку в сумматор на первом такте умножения.

При этом сочетании знаков возможно умножение без ввода поправки. Рассмотрим этот вариант на примере умножения по алгоритму Г (это справедливо и для других алгоритмов).

$Мн \cdot Мт = A \cdot B = [A \cdot b_1 \cdot 2^{-1}]_{доп} + [A \cdot b_2 \cdot 2^{-2}]_{доп} + ... + [A \cdot b_n \cdot 2^{-n}]_{доп}$.

На основании теоремы, доказывающей что сумма дополнительных кодов есть дополнительный код суммы, получаем

$[A \cdot b_1 \cdot 2^{-1} + A \cdot b_2 \cdot 2^{-2} + ... + A \cdot b_n \cdot 2^{-n}]_{доп} = [Мн \cdot Мт]_{доп}$.

*Пример.* $Мн = -0{,}1011$ умножение будем выполнять
$Мт = 0{,}1101$ по алгоритму умножения Б
$[Мн]_{доп} = 1{,}0101$

| | $b_1 \ldots b_4$ |
|---|---|
| 0,00000000 | $\Sigma_0^ч$ |
| $^+$ 1,11110101 | $П_1^ч = [Мн \cdot b_4]_{доп}$ |
| 1,11110101 | $\Sigma_1^ч$ ($\Sigma_2^ч = \Sigma_1^ч$, т. к. $b_3 = 0$) |
| $^+$ 1,11010**100** | $П_3^ч = [Мн \cdot b_2 \cdot 2^2]_{доп}$ |
| 1,11001001 | $\Sigma_3^ч$ |
| $^+$ 1,10101**000** | $П_4^ч = [Мн \cdot b_1 \cdot 2^3]_{доп}$ |
| 1,01110001 | $[Мн \cdot Мт]_{доп}$ |
| −0,10001111 | $Мн \cdot Мт$ |

4. $Мн < 0$,
   $Мт < 0$.

При отрицательных сомножителях произведение $Мн \cdot Мт > 0$.

$Мн \cdot Мт = 2 - [Мн \cdot Мт]_{доп}$.

Как и в случае 2 выполним умножение $Мн \cdot Мт'$. Но в этом случае $Мн < 0$.

$[Мн]_{доп} \cdot Мт' = [Мн]_{доп} \cdot (1 - Мт) = [Мн]_{доп} - [Мн]_{доп} \cdot Мт =$
$[Мн]_{доп} - [Мн \cdot Мт]_{доп}$,
$\Delta = 2 - [Мн \cdot Мт]_{доп} - [Мн]_{доп} + [Мн \cdot Мт]_{доп} = 2 - [Мн]_{доп} = [-Мн]_{доп} = |Мн|$.

*Пример*. Мн = − 0,1011 умножение будем выполнять
Мт = − 0,1101 по алгоритму умножения Г
$[Мн]_{доп}$ = 1,0101
$[Мт]_{доп}$ = 1,0011
$\Delta = [-Мн]_{доп}$ = 0,1011

| | |
|---|---|
| 0,00000000 | $\Sigma_0^ч$ |
| + 1,**111**01010 | $П_1^ч = [Мн \cdot b_3 \cdot 2^{-3}]_{доп}$ |
| 1,11101010 | $\Sigma_1^ч$ |
| + 1,**1111**0101 | $П_4^ч = [Мн \cdot b_4 \cdot 2^{-4}]_{доп}$ |
| 1,11011111 | $\Sigma_4^ч$ |
| + 0,**1011**0000 | $\Delta$ (поправка) |
| 0,10001111 | $[Мн \cdot Мт]_{доп}$ |

Далее коротко остановимся на **умножении целых чисел**.

1. Мн > 0,
   Мт > 0.

Как отмечалось выше, в этом случае умножение выполняется по правилам умножения чисел в прямых кодах.

2. Мн > 0,
   Мт < 0,
   $[Мт]_{доп} = 2^n - Мт$.

Так как сомножители имеют разные знаки, то произведение Мн · Мт < 0, следовательно, $[Мн \cdot Мт]_{доп} = 2^{2n} - Мн \cdot Мт$. Однако при умножении $Мн \cdot [Мт]_{доп}$ получается $Мн \cdot (2^n - Мт) = 2^n \cdot Мн - Мн \cdot Мт$. Следовательно, погрешность в этом случае равна:

$\Delta = 2^{2n} - Мн \cdot Мт - 2^n Мн + Мн \cdot Мт = 2^{2n} - 2^n Мн = 2^n \cdot (2^n - Мн) = [-Мн]_{доп} \cdot 2^n$

*Пример*. Мн = +110 умножение будем выполнять
Мт = −101 по алгоритму умножения А
$[Мн]_{доп}$ = 0.110
$[Мт]_{доп}$ = 1.011
$\Delta = [- Мн]_{доп} \cdot 2^3 = 1.010 \cdot 2^3 = 1.010000$

| | |
|---|---|
| 0.000 | $\Sigma_0^ч$ |
| + 0.110 | $П_1^ч = Мн \cdot b_4$ |
| 0.110 | $\Sigma_1^ч$ |
| 0.**0**11 0 | $\Sigma_1^ч \cdot 2^{-1}$ |
| + 0.110 | $П_2^ч = Мн \cdot b_3$ |
| 1.001 0 | $\Sigma_2^ч$ (возникло переполнение) |

0.**1**00 10 $\quad \Sigma_2^{ч} \cdot 2^{-1}$ (немодифицированный сдвиг − коррекция)

0.**01**0 010 $\quad \Sigma_3^{ч} \cdot 2^{-1}$ $(\Pi_3^{ч} = 0)$

$^{+}$ 1.010000 $\quad \Delta$ (поправка)

1.100 010 $\quad [Мн \cdot Мт]_{доп}$

− 011 110 $\quad Мн \cdot Мт$

3. Мн < 0,
   Мт > 0.

Здесь, как и при умножении дробных чисел, возможны два случая:

а) с вводом поправки в получаемое произведение

$[Мн]_{доп} = 2^n - Мн$.

Как и ранее, требуется получить $[Мн \cdot Мт]_{доп} = 2^{2n} - Мн \cdot Мт$. Получаем

$(2^n - Мн) \cdot Мт = 2^n \cdot Мт - Мн \cdot Мт$.

$\Delta = 2^{2n} - Мн \cdot Мт - 2^n \cdot Мт + Мн \cdot Мт = 2^n(2^n - Мт) = [-Мт]_{доп} \cdot 2^n$;

б) вариант без ввода поправки рассмотрим применительно к алгоритму умножения Г (как и ранее это справедливо и для других алгоритмов):

$Мн \cdot Мт = A \cdot B = [A \cdot b_1 \cdot 2^{-1}]_{доп} + [A \cdot b_2 \cdot 2^{-2}]_{доп} + ... + [A \cdot b_n \cdot 2^{-n}]_{доп} =$

$= [A \cdot b_1 \cdot 2^{-1} + A \cdot b_2 \cdot 2^{-2} + ... + A \cdot b_n \cdot 2^{-n}]_{доп} = [Мн \cdot Мт]_{доп}$

*Пример*. Мн = −110 — умножение будем выполнять

Мт = 101 — по алгоритму умножения Г

$[Мн]_{доп} = 1.010$

$[Мт]_{доп} = 0.101$

$b_1 ... b_3$

0.000000 $\quad \Sigma_0^{ч}$

$^{+}$ 1.**1**01000 $\quad \Pi_1^{ч} = [Мн \cdot b_1]_{доп} \cdot 2^{-1}$

1.101000 $\quad \Sigma_1^{ч}$ $(\Sigma_2^{ч} = \Sigma_1^{ч}$ так как $b_2 = 0)$

$^{+}$ 1.**111**010 $\quad \Pi_3^{ч} = [Мн \cdot b_3]_{доп} \cdot 2^{-3}$

1.100010 $\quad \Sigma_3^{ч}$ $([Мн \cdot Мт]_{доп})$

− 011110 $\quad Мн \cdot Мт$

4. Мн < 0,
   Мт < 0.

При этом сочетании знаков сомножителей в результате должно быть получено положительное произведение

$Мн \cdot Мт = 2^{2n} - [Мн \quad Мт]_{доп}$,

$[Мн]_{доп} = 2^n - Мн$,

$[Мт]_{доп} = 2^n - Мт$.

При умножении $[Мн]_{доп} \cdot [Мт]_{доп}$ получается:

$[Мн]_{доп} \cdot [Мт]_{доп} = [Мн]_{доп} (2^n - Мт) = 2^n [Мн]_{доп} - \underbrace{[Мн]_{доп} \cdot Мт}_{[Мн \cdot Мт]_{доп}}$,

$\Delta = 2^{2n} - [\text{Мн} \cdot \text{Мт}]_{\text{доп}} - 2^n\,[\text{Мн}]_{\text{доп}} + [\text{Мн} \cdot \text{Мт}]_{\text{доп}} = 2^{2n} - 2^n[\text{Мн}]_{\text{доп}} = [-\text{Мн}]_{\text{доп}} \cdot 2^n.$

*Пример.* Мн = −110

Мт = −101

$[\text{Мн}]_{\text{доп}} = 1.010$

$[\text{Мт}]_{\text{доп}} = 1.011$

$\overset{b_1 \ldots b_3}{\Delta = [-\text{Мн}]_{\text{доп}} \cdot 2^3 = 0.110 \cdot 2^3 = 0.110000}$

При умножении используем алгоритм Г.

| | |
|---|---|
| 0.000000 | $\Sigma_0^{\text{ч}}$ $(\Sigma_1^{\text{ч}} = \Sigma_0^{\text{ч}}$, т. к. $b_1 = 0)$ |
| 1.**11**0100 | $\Pi_2^{\text{ч}} = [\text{Мн} \cdot b_2]_{\text{доп}} \cdot 2^{-2}$ |
| 1.110100 | $\Sigma_2^{\text{ч}}$ |
| 1.**111**010 | $\Pi_3^{\text{ч}} = [\text{Мн} \cdot b_3]_{\text{доп}} \cdot 2^{-3}$ |
| 1.101110 | $\Sigma_3^{\text{ч}}$ |
| 0.110000 | $\Delta$ (поправка) |
| 0.011110 | Мн Мт |

### 2.19. Умножение на два разряда множителя в дополнительных кодах

При умножении в прямых кодах на два разряда множителя по алгоритмам В и Г произведение может быть получено с ошибкой. Например, если множитель будет 101011…. В этом случае преобразованный множитель по отмеченным алгоритмам будет иметь вид $100\overline{1}0\overline{1}$…. При умножении на этот преобразованный множитель будет получено неверное произведение.

Для получения верного результата при умножении чисел в дополнительном коде на два разряда, в отличие от умножения в прямых кодах, преобразованию подвергается не только пара 11, но также и пара 10. Это позволяет упростить анализ кода и формирование преобразованного множителя.

| пара | старшая | младшая | старшая | младшая |
|---|---|---|---|---|
| до преобразования | 00 | $1\overline{1}$ | 00 | $1\overline{0}$ |
| после преобразования | 01 | $0\overline{1}$ | 01 | $\overline{1}0$ |
| действие | +4Мн | −Мн | +4Мн | −2Мн |

В табл. 2 приведены различные варианты преобразования *i* и *i* + 1 пары множителя.

Таблица 2

| Преобразуемая пара | | Младшая пара | | Преобразованная пара | | Алгоритм | |
|---|---|---|---|---|---|---|---|
| *x* | *y* | *z* | − | *x*′ | *y*′ | А | Г |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 | 0 | 0 | − | 0 | 0 | 1 | 6 |
| 0 | 0 | 1 | − | 0 | 1 | 2 | 7 |
| 0 | 1 | 0 | − | 0 | 1 | 2 | 7 |

Окончание таблицы 2

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | – | $\overline{1}$ | 0 | 3 | 8 |
| 1 | 0 | 0 | – | $\overline{1}$ | $\overline{0}$ | 4 | 9 |
| 1 | 0 | 1 | – | 0 | $\overline{1}$ | 5 | 10 |
| 1 | 1 | 0 | – | 0 | $\overline{1}$ | 5 | 10 |
| 1 | 1 | 1 | – | 0 | 0 | 1 | 6 |

Действия, осуществляемые при выполнении умножения, например, алгоритмов А и Г следующие:

(1) $\sum_i^r = \sum_{i-1}^r \cdot 2^{-2}$

(2) $\sum_i^r = (\sum_{i-1}^r + \text{Мн}) \cdot 2^{-2}$

(3) $\sum_i^r = (\sum_{i-1}^r \cdot 2^{-1} + \text{Мн}) \cdot 2^{-1}$

(4) $\sum_i^r = (\sum_{i-1}^r \cdot 2^{-1} - \text{Мн}) \cdot 2^{-1}$

(5) $\sum_i^r = (\sum_{i-1}^r - \text{Мн}) \cdot 2^{-2}$

(6) $\sum_i^r = \sum_{i-1}^r$ и сдвиг Мн $(\text{Мн} \cdot 2^{-2})$

(7) $\sum_i^r = \sum_{i-1}^r + \text{Мн} \cdot 2^{-2}$

(8) $\sum_i^r = \sum_{i-1}^r + 2\text{Мн} \cdot 2^{-2}$

(9) $\sum_i^r = \sum_{i-1}^r - 2\text{Мн} \cdot 2^{-2}$

(10) $\sum_i^r = \sum_{i-1}^r - \text{Мн} \cdot 2^{-2}$

Если анализируемая пара имеет старшую цифру 1, то умножение на эту пару будет соответствовать вычитанию одного или двух множителей. На рис. 10 приведена логическая схема для формирования сигнала выполняемого действия при анализе пары *xy* и старшего разряда *z* соседней младшей пары при умножении чисел согласно алгоритму Г.

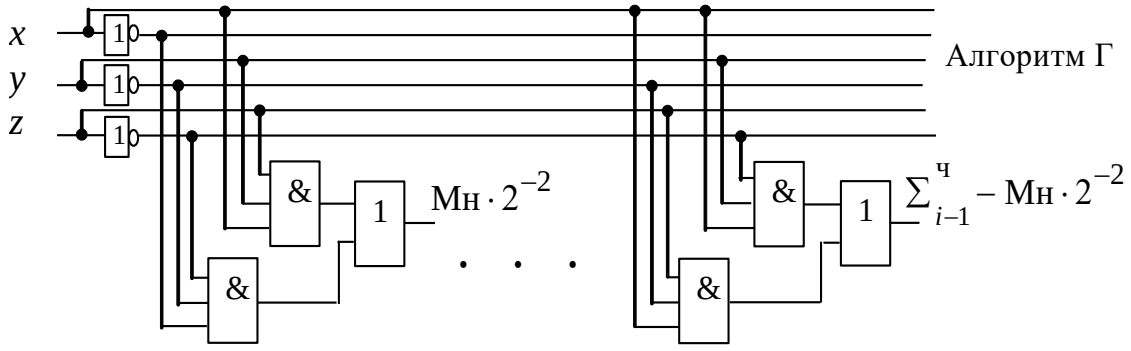



Рис. 10. Логическая схема формирования сигнала выполняемого действия

Как было показано выше, при умножении чисел в дополнительных кодах в общем случае необходимо вводить поправку для получения верного произведения. Однако при умножении на два разряда множителя, этого выполнять не требуется ввиду того, что знак тоже участвует в преобразовании. Если на умножение поступает отрицательный множитель, то при преобразовании его старшей (знаковой) пары (1.1 или 1.0) образуются новые пары ($0.\overline{1}$ или $\overline{1}.0$), а разряд, который обычно передается в старшую пару, отбра-

сывается. За счет этого производится поправка.

$Мт = -10110,$

$Мт^{п}_{доп} = 1.0\ 10\ 10,$

$Мт^{п}_{доп} = 01\ \overline{01}\ \overline{01}\ \overline{10}.$

Если при преобразовании возникает дополнительная (старшая) пара, как показано выше, то она просто отбрасывается, так как при умножении на эту дополнительную пару (01) происходит добавление множимого (+Мн), а затем, при вводе поправки (−Мн), множимое вычитается.

*Пример*. Выполнить умножение Мн на Мт.

$Мн = +\ 0101$ умножение выполним согласно

$Мт = +\ 1100111$ алгоритму Г

$[Мн]_{доп} = 0.0101$

$[Мт]_{доп} = 0.110\ 01\ 11$

Анализ пар Мт можно производить начиная от старших (при умножении по алгоритмам В и Г) и от младших разрядов (алгоритмы А и Б).

$[Мт]^{пр}_{доп} = 10\ \overline{10}\ 10\ \overline{01}$

$[-Мн]_{доп} = 1.1011$

$[-2Мн]_{доп} = 1.0110$

$[2Мн]_{доп} = 0.1010$

$0.0000\ \ 00000000 \qquad \Sigma^{ч}_{0}$

$+\ \underline{0.\mathbf{00}10\ \ 10000000} \qquad \Pi^{ч}_{1} = 2Мн \cdot 2^{-2}$

$0.0010\ \ 10000000 \qquad \Sigma^{ч}_{1}$

$+\ \underline{1.\mathbf{1111}\ \ 01100000} \qquad \Pi^{ч}_{2} = -2Мн \cdot 2^{-4}$

$0.0001\ \ 11100000 \qquad \Sigma^{ч}_{2}$

$+\ \underline{0.\mathbf{0000\ \ 00}101000} \qquad \Pi^{ч}_{3} = 2Мн \cdot 2^{-6}$

$0.0010\ \ 00001000 \qquad \Sigma^{ч}_{3}$

$+\ \underline{1.\mathbf{1111\ \ 1111}1011} \qquad \Pi^{ч}_{4} = -Мн \cdot 2^{-8}$

$0.0010\ \ 00000011 \qquad \Sigma^{ч}_{4} = [Мн\ \cdot\ Мт]_{доп}$

*Пример.* Выполнить умножение Мн на Мт.

$Мн = -0101$ умножение выполним согласно

$Мт = -1100111$ алгоритму Б

$[Мн]_{доп} = 1.1011$

$[Мт]_{доп} = 1.0\ 01\ 10\ 01$

$[Мт]^{пр}_{доп} = 1.0\ 10\ \overline{10}\ \overline{01}$

$[2Мн]_{доп} = 1.0110 \qquad [-2Мн]_{доп} = 0.1010$

$0.00000000\ 0000 \qquad \Sigma^{ч}_{0}$

$+\ \underline{1.11111111\ 1011} \qquad \Pi^{ч}_{1} = Мн$

| | |
|---|---|
| 1.11111111 1011 | $\Sigma_1^{ч}$ |
| $^{+}$ <u>0.00000010 10**00**</u> | $П_2^{ч} = -2Мн \cdot 2^2$ |
| 0.00000010 0011 | $\Sigma_2^{ч}$ |
| $^{+}$ <u>1.11110110 **0000**</u> | $П_3^{ч} = 2Мн \cdot 2^4$ |
| 1.11111000 0011 | $\Sigma_3^{ч}$ |
| $^{+}$ <u>0.00101**00 0000**</u> | $П_4^{ч} = -2Мн \cdot 2^6$ |
| 0.00100000 0011 | $\Sigma_4^{ч} = Мн \cdot Мт$ |

### 2.20. Матричные методы умножения

Кроме рассмотренных методов ускоренного умножения существуют методы умножения, основанные на использовании матриц промежуточных результатов [1].

Пусть имеем сомножители

Мн = $A = a_n \ldots a_2\ a_1$

Мт = $B = b_n \ldots b_2\ b_1$

$$
\begin{array}{r}
\times\ A = a_n \ldots a_2\ a_1 \\
B = b_n \ldots b_2\ b_1 \\
\hline
a_n b_1 \ldots a_3 b_1\ \ a_2 b_1\ \ a_1 b_1 \\
+\ a_n b_2\ \ \ldots\ \ a_2 b_2\ \ a_1 b_2 \\
\quad \ldots \\
+\ a_n b_n \ldots a_2 b_n\ \ a_1 b_n \\
\hline
C = C_{2n} \quad \ldots \quad C_2\ \ C_1
\end{array}
$$

Рассмотрим схему умножения чисел согласно алгоритму Б. Данная схема умножения может быть представлена в виде матрицы (табл.3).

Таблица 3

| $A \cdot B$ | $a_n$ | . . . | $a_2$ | $a_1$ |
|---|---|---|---|---|
| $b_1$ | $a_n b_1$ | . . . | $a_2 b_1$ | $a_1 b_1$ |
| $b_2$ | $a_n b_2$ | . . . | $a_2 b_2$ | $a_1 b_2$ |
| . | . | | . | . |
| . | . | . . . | . | . |
| . | . | | . | . |
| $b_n$ | $a_n b_n$ | . . . | $a_2 b_n$ | $a_1 b_n$ |

Каждый элемент $a_i b_j$ ($i, j = \overline{1, n}$) принимает значение 0 или 1. Произведение $A \cdot B$ может быть получено, если суммировать элементы матрицы (по диагонали).

$$
\begin{array}{l|l|l|l|}
 & a_3 b_1 & a_2 b_1 & a_1 b_1 \\
\ldots & a_2 b_2 & a_1 b_2 & \\
 & a_1 b_3 & & \\
\end{array}
$$

Для суммирования по столбцам могут быть использованы счетчики. Однако при достаточно большом значении величины $n$ потребуются счетчики с большим числом входов, что существенно увеличит время сложения. Но этот принцип умножения может быть реализован на устройствах, имеющих не более трех входов. В их качестве могут быть использованы одноразрядные двоичные сумматоры и полусумматоры.

На рис. 11 приведена структурная схема устройства умножения для реализации матричного алгоритма. В узлах схемы расположены одноразрядные двоичные сумматоры (*SM*) и одноразрядные двоичные полусумматоры (*HS*). Полусумматор отличается от сумматора тем, что не имеет входного переноса.

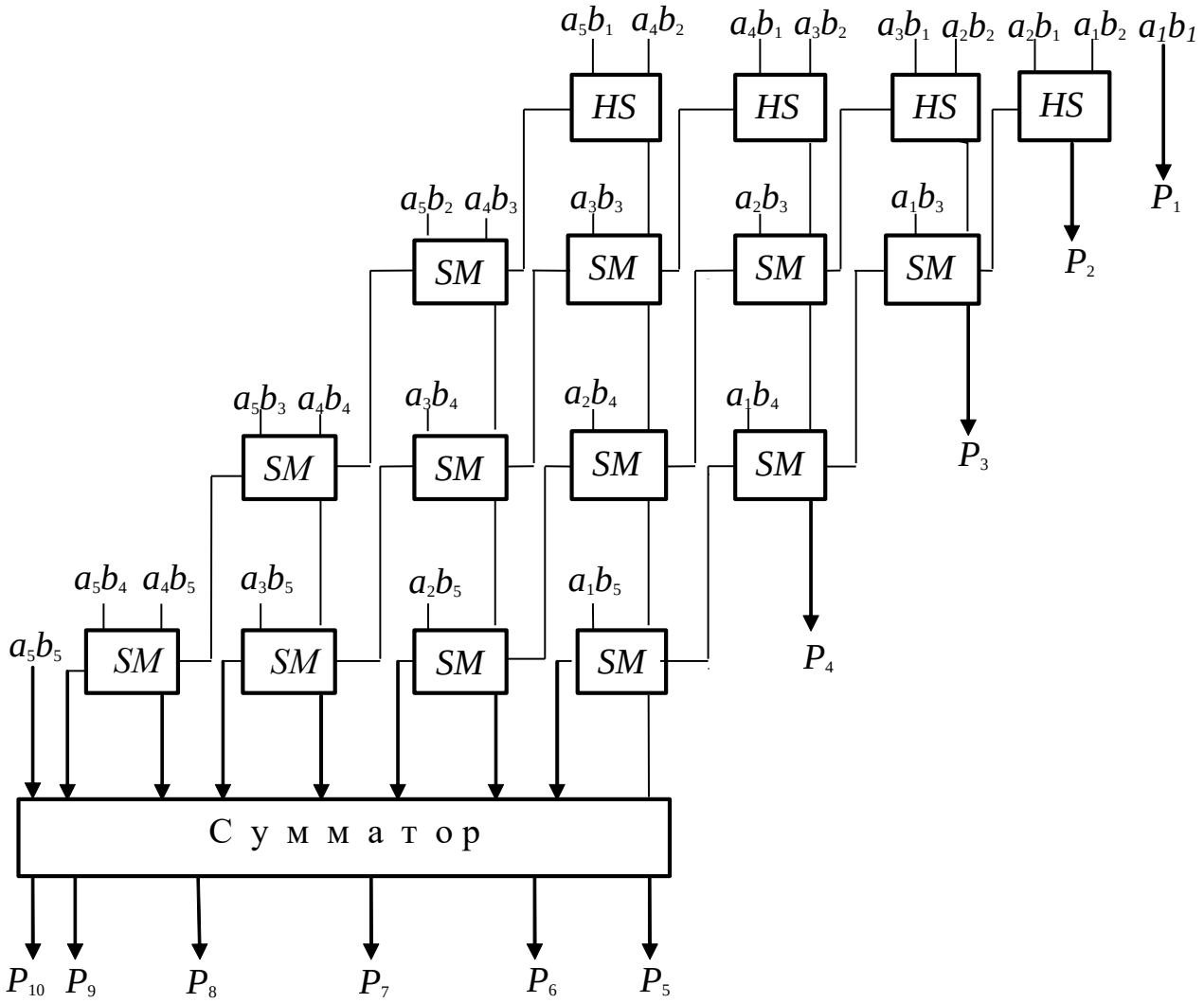



Рис. 11. Структурная схема матричного умножения

Реализация методов матричного умножения требует большего количества оборудования, чем метод последовательного умножения, и дает больший выигрыш во времени. В связи с увеличением степени интеграции элементной базы ограничения по количеству оборудования становятся не столь строгими.

### 2.21. Машинные методы деления

Деление – простое многократное вычитание делителя вначале из делимого, затем из остатков. Введем обозначения, используемые ниже: Дм – делимое, Дт – делитель, $A_i$ – очередной ($i$-й) остаток, Чт – частное. Известны два основных подхода к операции деления:

- с восстановлением остатков;
- без восстановления остатков.

Независимо от метода деления после каждого вычитания делителя формируется очередная цифра частного. Операция деления является операцией, дающей не всегда точный результат, поэтому признаком окончания операции деления может быть достижение заданной точности (по числу полученных разрядов). Если при выполнении деления получен нулевой $i$-й остаток, то деление прекращается и в оставшиеся разряды частного записываются нули. Первым шагом деления двух чисел является пробное вычитание, выявляющее соотношение между делимым и делителем. При делении в случае переполнения следует для чисел с фиксированной запятой процесс остановить, с плавающей запятой продолжить до конца, а потом, после получения последней $n$-й цифры частного, число сдвинуть вправо на один разряд с добавлением единицы к порядку, равному разности порядков делимого и делителя.

#### 2.21.1. Деление чисел в прямых кодах

*Алгоритм деления с восстановлением остатка* состоит в следующем.

1. Выполняется пробное вычитание с формированием первого остатка $A_1 = [\text{Дм}]_{\text{доп}} + [-\text{Дт}]_{\text{доп}}$. Далее, если $A_1 < 0$, то в первый разряд, расположенный слева от запятой, заносится нуль (0), иначе − единица (1) – что является признаком переполнения и осуществляется переход к п. 5.

2. Если $A_i < 0$, то восстанавливаем предыдущий остаток $A_i = A_i + [\text{Дт}]_{\text{доп}}$.

3. Формирование очередного остатка. $A_{i+1} = A_i \cdot 2 + [-\text{Дт}]_{\text{доп}}$ если $A_{i+1} < 0$, то в очередной разряд частного справа от запятой записывается нуль (Чт($n$) = 0), иначе − единица (Чт($n$) = 1).

4. Если достигнута заданная точность частного или получен нулевой остаток $A_{i+1}$, то процесс деления окончен и осуществляется переход к п. 5, иначе возвращаемся к п. 2 алгоритма.

5. Конец алгоритма.

Из рассмотренного алгоритма видно следующее:

1) необходимо затрачивать дополнительно время на восстановление остатка;

2) процесс деления нерегулярный, в зависимости от делимого и делителя частное будет содержать нулей больше или меньше, и чем больше нулей, тем больше требуется времени на восстановление остатков.

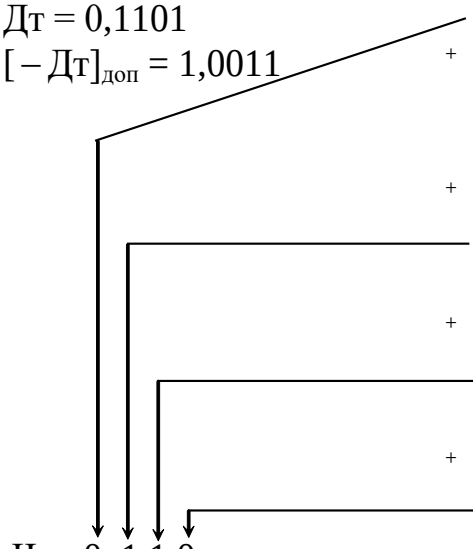
Рассмотрим пример деления чисел в прямых кодах с использованием рассмотренного выше алгоритма.

*Пример*.

Дм = 0,1011

Дт = 0,1101

$[-\text{Дт}]_{\text{доп}}$ = 1,0011



| | | |
|---|---|---|
| | 0,1011 | Дм ($A_0$) |
| + | 1,0011 | $[-\text{Дт}]_{\text{доп}}$ |
| | **1**,1110 | $A_1 < 0$ |
| + | 0,1101 | + Дт (восстановление $A_0$) |
| | 0,1011 | $A_0$ |
| | 1,0110 | $A_0$ 2 |
| + | 1,0011 | $[-\text{Дт}]_{\text{доп}}$ |
| | **0**,1001 | $A_2$ |
| | 1,0010 | $A_2$ 2 |
| + | 1,0011 | $[-\text{Дт}]_{\text{доп}}$ |
| | **0**,0101 | $A_3$ |
| | 0,1010 | $A_3$ 2 |
| + | 1,0011 | $[-\text{Дт}]_{\text{доп}}$ |
| | **1**,1101 | $A_4$ |
| Чт= 0, 1 1 0 . . . | . . . | |

Как видно из примера, в случае $A_i < 0$ для получения остатка $A_{i+1}$ необходимо выполнить

$A_{i+1} = (A_i + \text{Дт}) \cdot 2 - \text{Дт} = A_i \cdot 2 + 2\text{Дт} - \text{Дт} = A_i \cdot 2 + \text{Дт}$.

Из этого следует, что восстанавливать отрицательный остаток необязательно. Достаточно сдвинуть полученный остаток влево на один разряд и добавить делитель. Это является основой алгоритма для выполнения деления без восстановления остатка.

*Алгоритм деления без восстановления остатка.*

1. Выполняется пробное вычитание с формированием первого остатка $A_1 = [\text{Дм}]_{\text{доп}} + [-\text{Дт}]_{\text{доп}}$. Далее, если $A_1 < 0$, то в первый разряд, расположенный слева от запятой, заносится нуль (0), иначе − единица (1) – что является признаком переполнения и осуществляется переход к п. 5.

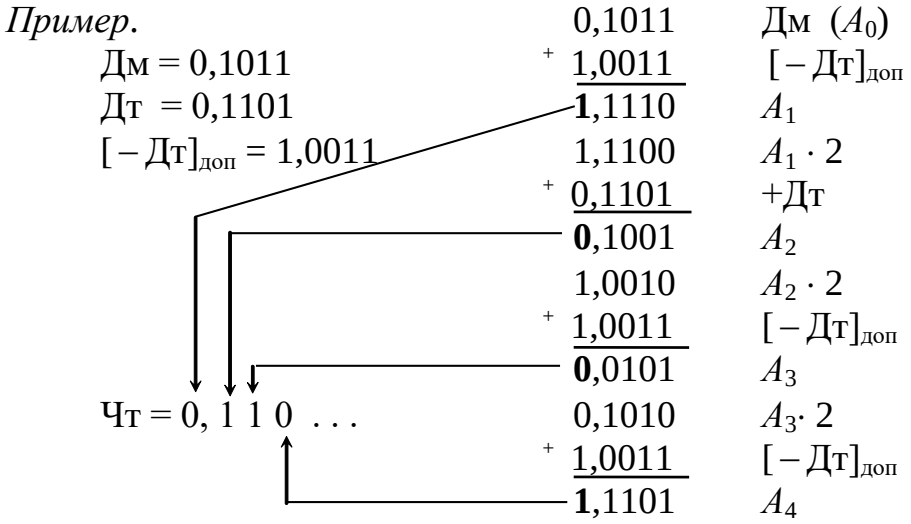
2. Формирование очередного остатка. Если $A_i < 0$, то $A_{i+1} = A_i \cdot 2 + [\text{Дт}]_{\text{доп}}$, иначе $A_{i+1} = A_i \cdot 2 + [-\text{Дт}]_{\text{доп}}$.

3. Если $A_{i+1} < 0$, то в очередной разряд частного справа от запятой записывается нуль (Чт($n$) = 0), иначе записывается единица (Чт($n$) = 1).

4. Если достигнута заданная точность частого или получен нулевой остаток $A_{i+1}$, то процесс деления окончен и осуществляется переход к п. 5, иначе возвращаемся к п. 2 алгоритма.

5. Конец алгоритма.

Рассмотрим пример демонстрирующий алгоритм деления чисел без восстановления остатков.




### 2.21.2. Деление чисел в дополнительных кодах

При делении чисел знаковая и значащая части частного формируются раздельно. Знак частного формируется согласно формуле

Знак Чт = Знак Дм ⊕ Знак Дт.

Основой деления чисел в дополнительных кодах является деление без восстановления остатка. В отличие от деления в прямых кодах, здесь как для определения цифры частного, так и для определения последующего действия сравнивается знак делимого (остатка) со знаком делителя.

Ниже приведен алгоритм деления чисел в дополнительных кодах.

1. Выполняется пробное вычитание: если Знак Дм ≠ Знаку Дт, то первый остаток $A_1 = [\text{Дм}]_{\text{доп}} + [\text{Дт}]_{\text{доп}}$, иначе $A_1 = [\text{Дм}]_{\text{доп}} + [-\text{Дт}]_{\text{доп.}}$ Далее формируется первый разряд, расположенный слева от запятой − нуль (0) (если знак $A_1$ ≠ знаку Дт), иначе − единица (1).

2. Формирование очередного остатка. Если Знак $A_i$ ≠ Знаку Дт, то $A_{i+1} = A_i \cdot 2 + [\text{Дт}]_{\text{доп}}$, иначе $A_{i+1} = A_i \cdot 2 + [-\text{Дт}]_{\text{доп}}$.

3. Если Знак $A_{i+1}$ ≠ Знаку Дт, то в очередной разряд частного справа от запятой заносится нуль (Чт($n$) = 0), иначе − единица (Чт($n$) = 1).

4. Если достигнута заданная точность частого или получен нулевой остаток $A_{i+1}$, то процесс деления окончен, иначе возвращаемся к п. 2 алгоритма.

5. Конец алгоритма.

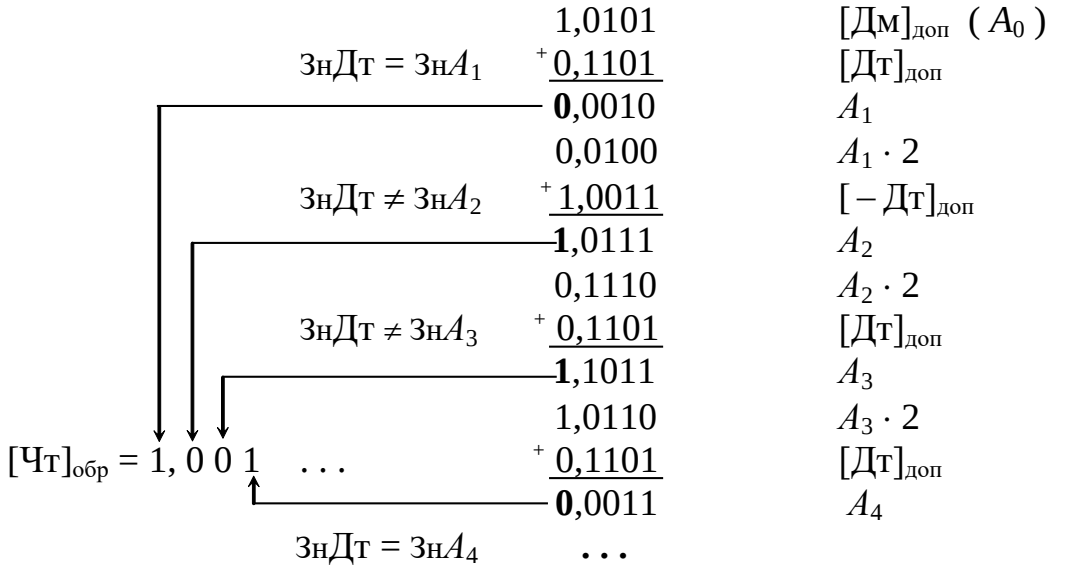
Рассмотрим пример деления чисел демонстрирующий использование приведенного алгоритма.

*Пример*.

Дм = − 0,1011
Дт = 0,1101

На операцию деления Дм и Дт поступают в дополнительном коде.

$[\text{Дм}]_{\text{доп}} = 1,0101$
$[\text{Дт}]_{\text{доп}} = 0,1101$
$[-\text{Дт}]_{\text{доп}} = 1,0011$




**2.22. Схема устройства деления**

На рис. 12 приведена структурная схема устройства, выполняющего деление чисел в дополнительном коде.

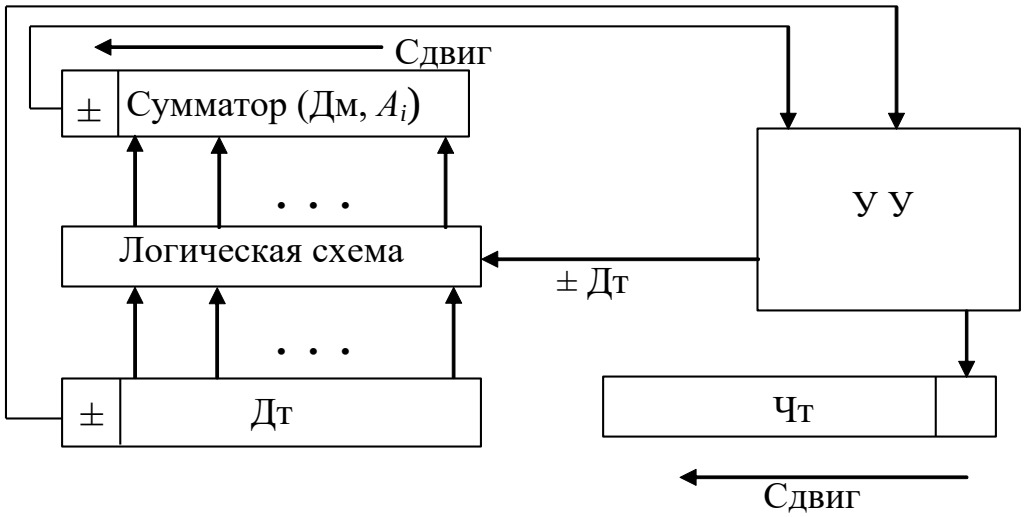



Рис. 12. Структурная схема устройства деления

Устройство управления (УУ), анализируя знаки сумматора (Дм, $A_i$) и делителя, вырабатывает сигнал, подаваемый на логическую схему, на выходе которой формируется ±Дт (для очередного вычитания). Кроме этого УУ формирует разряд частного.

**2.23. Методы ускорения деления**

Логические методы основываются на анализе остатка, по виду которого можно сформировать несколько цифр частного в пределах одного такта [1,9].

При этом Дт выбирается (формируется) таким образом, чтобы после запятой шла единица, т.е. чтобы он был нормализован. Если очередной остаток получился настолько мал, что после запятой следует $n + 1$ нулей, то в частное

может быть записано $n$ нулей или единиц, а остаток может быть сдвинут на $r$ разрядов влево.

Реализация этого метода ускорения кроме устройства управления делением требует логическую схему, осуществляющую две функции:

1) сдвиг модулей делителя и делимого до тех пор, пока у модуля делителя после запятой не останется ни одного нуля;

2) выявление остатков вида 0,0…01 или 1,1…10.

Степень сложности логической схемы определяется количеством разрядов, участвующих в косвенном сравнении модулей делителя и делимого (остатков).

Обычно реализация логических методов ускорения при делении несколько сложнее, чем при умножении. При делении необходимо либо каждый остаток переводить в прямой код, либо, если остаток оставить в дополнительном коде, анализировать два разряда одновременно 0,0… или 1,1… .

### 2.24. Двоично-десятичные коды

Пусть число $A$ представлено в системе счисления с основанием $r$:

$$A_r = \sum_{i=1}^{n} a_i \cdot r^{p-i} \qquad (r \neq 2^k, \ k = 1, 2, 3, ...).$$

Цифры $a_i$ будем представлять двоичными разрядами $d_1, d_2, \ldots, d_m$. Каждому двоичному разряду припишем веса $p_1, p_2, \ldots, p_m$. Тогда каждый разряд $a_i$ числа $A$ будет иметь вид $a_i = \sum_{l=1}^{m} d_l \cdot p_l$ , а все число

$$A_r = \sum_{i=1}^{n} \left( \sum_{l=1}^{m} d_l \cdot p_l \right) \cdot r^{p-i}, \qquad (4)$$

где $n$ и $m$ определяют общее число двоичных разрядов.

Если каждый разряд числа имеет вес и при $r \neq 2^k$ не выполняется равенство $p_k = r \cdot p_{k-1}$, то системы принято называть *взвешенными*. Количество разрядов m должно удовлетворять выражению $m \geq \log_2 r$. Если десятичное число записано в виде выражения (4), то будем говорить, что число представлено в двоично-десятичном коде. Наибольшее распространение получили двоично-десятичные коды, в которых десятичная цифра представляется двоичной тетрадой – *BCD*-коды (*Binary coded decimal*). Существует множество способов кодирования десятичных цифр. Существенным при этом является простота представления инверсных кодов и простота формирования сигнала переноса из одной десятичной цифры в другую.

Сформулируем набор требований, позволяющих упростить выполнение арифметических операций и операций перевода чисел:

− четность − четным десятичным цифрам соответствуют только четные двоичные коды и наоборот. Это обеспечивает эффективность операций округления, умножения и деления чисел в *BCD*-кодах;

− дополняемость − сумма двоичного кода и инверсного ему кода любой десятичной цифры должна быть равна 9. Это обеспечивает эффективность операции алгебраического сложения в BCD-кодах;

− упорядоченность, т. е. большей десятичной цифре соответствует большая тетрада и наоборот;

− единственность представления десятичной цифры двоичной тетрадой;

− взвешенность, т. е. каждому разряду двоичного представления десятичной цифры поставлен в соответствие вес. Это обеспечивает эффективность всех арифметических и логических операций в BCD-кодах.

Если каждая десятичная цифра кодируется соответствующим двоичным эквивалентом, то такое кодирование называется *кодом прямого замещения*.

*BCD*-код – код, взаимодополняемый до 15. Это создает некоторые неудобства при суммировании чисел − ввод поправки в некоторых случаях. В то же время этот код имеет одно существенное достоинство – аддитивность: *сумма кодов равна коду суммы*.

0011 код 3
$^{+}$0101 код 5
1000 код 8

Основной недостаток этого кода заключается в том, что инверсия какой-либо цифры оказывается цифрой, дополняющей данную цифру до 15, а не до 9.

$a = 0100 \rightarrow 4$

$\overline{a} = 1011 \rightarrow 11$, то $a + \overline{a} = 1111 = 15$

В табл. 4 показаны различные способы кодирования десятичных цифр.

Таблица 4

| Десятичные цифры | Эквивалент в коде | | | | | | |
|---|---|---|---|---|---|---|---|
| | 8421 | 2421 | 7421 | 5211 | 8421+ 3 | 3*a* + 2 | 2 из 5 |
| 0 | 0000 | 0000 | 0000 | 0000 | 0011 | 00010 | 11000 |
| 1 | 0001 | 0001 | 0001 | 0001 | 0100 | 00101 | 00011 |
| 2 | 0010 | 0010 | 0010 | 0011 | 0101 | 01000 | 00101 |
| 3 | 0011 | 0011 | 0011 | 0101 | 0110 | 01011 | 00110 |
| 4 | 0100 | 0100 | 0100 | 0111 | 0111 | 01110 | 01001 |
| 5 | 0101 | 1011 | 0101 | 1000 | 1000 | 10001 | 01010 |
| 6 | 0110 | 1100 | 0110 | 1010 | 1001 | 10100 | 01100 |
| 7 | 0111 | 1101 | 1000 | 1100 | 1010 | 10111 | 10001 |
| 8 | 1000 | 1110 | 1001 | 1110 | 1011 | 11010 | 10010 |
| 9 | 1001 | 1111 | 1010 | 1111 | 1100 | 11101 | 10100 |

### 2.24.1. Суммирование чисел с одинаковыми знаками в *BCD*-коде

При выполнении операций над отмеченными кодами возможны следующие особенности:

− наличие разрешенных и запрещенных комбинаций, свидетельствующих о правильности результата или необходимости его коррекции;

− при сложении тетрад возможен потетрадный (16 единиц), а не поразрядный (10 единиц) перенос, что также требует корректировки результата.

При сложении чисел в *BCD*-коде возможны три случая:

1. $(a + b) \leq 9$. В этом случае если действия выполняются по правилам

двоичной арифметики, то величина получаемой суммы не превышает девяти и коррекция результата не требуется.

| | |
|---|---|
| 5 | 0101 |
| + 3 | + 0011 |
| 8 | 1000 |

2. $10 \le (a + b) \le 15$. Если результат сложения двух чисел попадает в данный диапазон чисел, то тетрада результата будет содержать единицы в разрядах с весом 8 и 4 или 8 и 2 – назовем это запрещенными комбинациями.

| | | | |
|---|---|---|---|
| 5 | 0101 | 9 | 1001 |
| + 6 | + 0110 | + 4 | + 0100 |
| 11 | 1011 | 13 | 1101 |

В этом случае в тетраде накопилось более девяти единиц и должен быть выполнен десятичный перенос. Перенос единицы в старший разряд выполняется принудительно с помощью логической схемы. Условием для формирования единицы переноса является возникновение запрещенной комбинации. Однако тетраду надо освободить от десяти избыточных единиц. Это тоже делается принудительно добавлением 0110 (шестерки), что приводит к возникновению шестнадцатеричного переноса, который вместе с добавленной шестеркой унесет и лишний десяток из тетрады. Этот перенос игнорируется. Структура схемы, выявляющей запрещенную комбинацию и формирующей перенос, приведена на рис. 13.

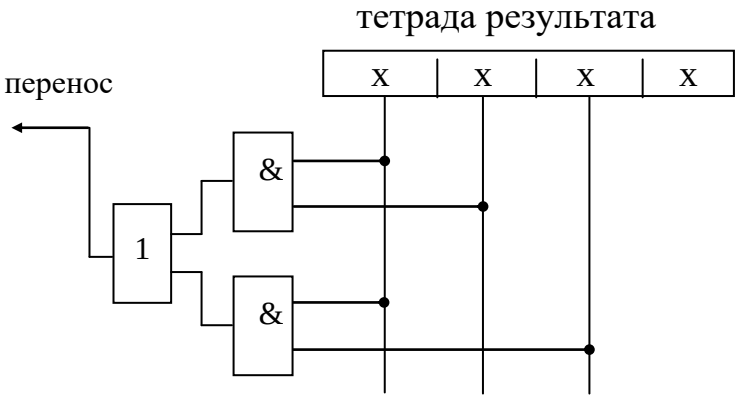



Рис. 13. Схема определения запрещенной комбинации

3. $(a + b) \ge 16$. Здесь в процессе суммирования возникает шестнадцатеричный перенос, в результате которого тетраду покидают 16 единиц вместо 10, т. е. вместе с десятком тетраду покидают те шесть единиц, которые ей принадлежат. Чтобы восстановить верное значение этой тетрады, необходимо к ней добавить 0110 (шесть).

| | |
|---|---|
| 8 | 1000 |
| + 9 | + 1001 |
| 11 | 1 0001 |
| | 0110 ( +6 корекция) |
| | 0111 |

Таким образом, из сказанного выше можно сформулировать следующие правила потетрадного сложения чисел в *BCD*-кодах:

1) Если при потетрадном сложении перенос в соседнюю старшую тетраду не возникает, то результат суммирования не требует коррекции.

2) Коррекция результата потетрадного сложения путем добавления поправки 0110 требуется в случае, если возникает:

а) потетрадный перенос в старшую тетраду;

б) запрещенная комбинация.

Устройство, которое работает по сформулированным выше правилам, называется ***одноразрядным двоично-десятичным сумматором*** (рис. 14).

*Пример*. Необходимо сложить числа: $A = 169$ и $B = 378$ в *BCD*-коде.

| | |
|---|---|
| A = 169 | $A = 0.0001\ 0110\ 1001$ |
| $+\ B = 378$ | $+\ B = 0.0011\ 0111\ 1000$ |
| $A + B = 547$ | $A + B = 0.0101\ 1110\ 0001$ |
| | $0110\ 0110$ |
| | $0.0101\ 0100\ 0111$ |

перенос игнорируется

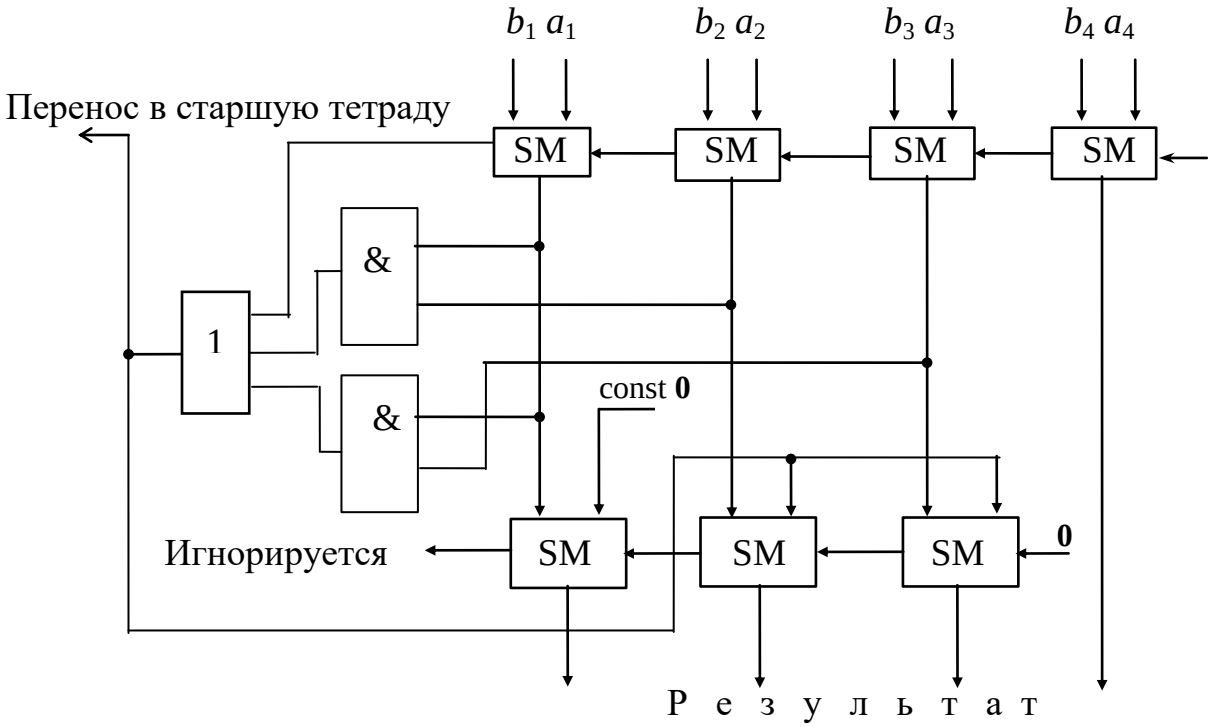



Рис. 14. Структурная схема одноразрядного двоично-десятичного сумматора

### 2.24.2. Суммирование чисел с разными знаками в *BCD*-коде

Отрицательные *BCD*-коды должны представляться в прямом, обратном или дополнительном кодах. Особенностью *BCD*-кодов является то, что инверсия тетрады означает дополнение до 15, а для соответствующей десятичной цифры необходимо дополнить до 9. Следовательно, нужно убрать разницу. Один из приемов формирования обратного *BCD*-кода состоит в добавлении во все тетрады отрицательного числа 0110, затем их инверсии.

При сложении чисел с разными знаками возможны следующие случаи.

1. $a - b \geq 0$

| | | | |
|---|---|---|---|
| $a =$ | 7 | 0 . 0111 | $[\,a\,]_{обр}$ |
| $b =$ | $-3$ | $+$1 . 1100 | $[\,b\,]_{обр}$ |
| | 4 | 10 . 0011 | |
| | | ⌊⟶1 | |
| | | 0 . 0100 | |

При образовании инверсии отрицательной тетрады в нее добавляются 15 единиц. Эти 15 единиц находятся и в сумме. Но благодаря шестнадцатеричному переносу из тетрады суммы уходит 16 единиц (одна единица восстанавливается добавлением по цепи циклического переноса).

2. $a - b < 0$

| | | | |
|---|---|---|---|
| $a =$ | 3 | 0 . 0011 | $[\,a\,]_{обр}$ |
| $b =$ | $-7$ | $+$1 . 1000 | $[\,b\,]_{обр}$ |
| | $-4$ | 1 . 1011 | |
| | | 0 . 0100 | |

Здесь, как и в предыдущем примере, в тетраде суммы пятнадцать лишних единиц. При переходе от инверсной формы к прямой лишние единицы уничтожаются сами собой. Это то же самое, что от значащей части суммы вычесть пятнадцать: 1011 − 1111 = 0100. Рассмотрим несколько примеров.

*Пример*. Необходимо сложить числа: $A$ = 378 и $B$ = –169 в *BCD*-коде.

| | |
|---|---|
| A = 378 | 0. 0011 0111 1000 |
| – B = 169 | $+$1. 1110 1001 0110 |
| $A - B$ = 209 | 10. 0010 0000 1110 |
| | ⌊циклический перенос 1 |
| | 0. 0010 0000 1111 |

Из последней тетрады нет переноса − это соответствует заему в нее 16 единиц (вместо необходимых десяти). Следовательно, из нее необходимо удалить лишние шесть единиц. Для этого в тетраду добавляется десятка (1010) – дополнение шести до шестнадцати:

| | |
|---|---|
| | 0. 0010 0000 1111 |
| | 1010 |
| | 0. 0010 0000 1001 |
| | + 2 0 9 |

*Пример*. Необходимо сложить числа: $A$ = 169 и $B$ = –378 в *BCD*-коде.

| | |
|---|---|
| $-A$ = 169 | 0. 0001 0110 1001 |
| $B$ = 378 | $+$1. 1100 1000 0111 |
| $A - B$ = − 209 | 1. 1101 1111 0000 |
| | 0110 |
| | 1. 1101 1111 0110 |
| | 0010 0000 1001 |
| | − 2 0 9 |

Из последней тетрады есть перенос, значит в нее произошел заем. Для удаления 6 из отрицательной тетрады результата необходимо добавить в нее 6.

Таким образом, в тетраду производится заем, если результат:

- положительный и из тетрады нет переноса;
- отрицательный и из тетрады есть перенос.

В случае заема в тетраду необходима коррекция.

### 2.24.3. *BCD*-коды с избытком 3

Иначе говоря, это коды чисел из системы (*BCD* + 3). В этом коде каждая десятичная цифра $a_i$ представляется в виде двоичного эквивалента суммы $a_i + 3$. В отличие от *BCD*-кода код *BCD* + 3 – самодополняющийся, но не имеющий свойства взвешенности. Наиболее часто применяется в десятичной арифметике, так как при выполнении двоичного суммирования легко выделить десятичный перенос.

Возможны следующие два случая сложения чисел в *BCD*-коде +3:

1) $a + b \le 9$ , $[ ( a + 3 ) + ( b + 3 ) ] \le 15$ и, следовательно, в тетраде суммы будут лишние 6 единиц. Чтобы тетрада суммы осталась тоже с избытком 3, нужно вычесть 3;

2) $a + b \ge 10$; $[ ( a + 3 ) + ( b + 3 ) ] \ge 16$. Здесь во всех случаях возникает шестнадцатеричный перенос, вместе с которым тетраду суммы покинут и шесть избыточных единиц. Чтобы тетрада суммы осталась с избытком 3, надо добавить 3.

Если складываются числа с разными знаками, то избыток в тетраде суммы будет равен нулю и суммирование, таким образом, сводится к правилам суммирования в BCD-коде.

*Пример.* Выполнить сложение чисел 169 и 378 в *BCD*-коде +3.

| | |
|---|---|
| | 0.0100 1001 1100 |
| $A = 169$ | $^{+}$0.0110 1010 1011 |
| $^{+}\ B = 378$ | 0.1011 0100 0111 |
| $A + B = 547$ | –0011+0011+0011 |
| | 0.1000 0111 1010 |
| | 8 7 10 |

*Пример*. Выполнить вычитание из числа 378 числа 169 в *BCD*-коде +3.

| | |
|---|---|
| $A = 378$ | 0.0110 1010 1011 |
| $-\ B = 169$ | $^{+}$1.1011 0110 0011 |
| $A - B = 209$ | 1 0.0010 0000 1110 |
| | циклический перенос 1 |
| | 0.0010 0000 1111 |
| | +0011+0011–0011 |
| | 0.0101 0011 1100 |
| | 5 3 12 |

*Пример.* Выполнить вычитание из числа 169 числа 378 в *BCD*-коде +3.

| | |
|---|---|
| $A = 169$ | 0.0100 1001 1100 |
| $-\ B = 378$ | $^{+}$ 1.1001 0101 0100 |
| $A - B = -209$ | 1.1101 1111 0000 |
| | –0011 –0011 +0011 |
| | 1.1010 1100 0011 |
| | − 0101 0011 1100 |
| | 5 3 12 |

*Правило*. Если из тетрады был перенос, надо добавить 0011, если переноса не было − вычесть 0011 (или добавить 1100), независимо от знака слагаемых и знака суммы.

### 2.24.4. *BCD*-код с избытком 6 для одного из слагаемых

При сложении чисел с одинаковыми знаками неважно, к какому из слагаемых добавить 0110. Причем это равносильно добавлению 0011 к каждому слагаемому.

При суммировании чисел в коде с избытком 6 коррекция может понадобиться только в случае, когда сумма меньше 16. В остальных случаях (сумма больше 16) возникает перенос, удаляющий из тетрады 6 лишних единиц, и коррекции результата не требуется.

Результат должен быть без избытка.

*Пример.*

| | |
|---|---|
| $A = 169$ | 0.0111 1100 1111 |
| $^{+}\ B = 378$ | $^{+}$ 0.0011 0111 1000 |
| $A + B = 547$ | 0.1011 0100 0111 |
| | –0110 |
| | 0.0101 0100 0111 |

Суммирование чисел с разными знаками производится в *BCD*-коде, но в сущности тоже с избытком 6.

### Контрольные вопросы и задания

1. Дайте определение системы счисления.
2. Дайте понятие базиса и основания системы счисления.
3. Выполните перевод чисел из десятичной системы счисления в двоичную:
   а) 123,46; б) 79,135; в) 325,753; г) 295,71; д) 197,34; е) 253,97.
4. Выполните перевод чисел из десятичной системы счисления в восьмеричную:
   а) 521,47; б) 179,35; в) 312,743; г) 315,734; д) 517,137; е) 831,91.
5. Выполните перевод чисел из десятичной системы счисления в шестнадцатиричную:
   а) 853,65; б) 579,32; в) 125,613; г) 455,91; д) 199,912; е) 754,87.
6. Выполните перевод чисел из двоичной системы счисления в восьмеричную:
   а) 101101,10101; б) 10101011,01010; в) 1000001111,001111;
   г) 11001010,101011; д) 101111010,011011; е) 10101110,011110111.

7. Выполните перевод чисел из восьмеричной системы счисления в шестнадцатеричную:

а) 531,37; б) 256,55; в) 342,631; г) 415,723; д) 572,165; е) 1014,43.

8. Выполните перевод чисел из шестнадцатеричной системы счисления в восьмеричную:

а) 1*AC*,*B*5; б) 5*F*9,32; в) *ABC*,1*D*; г) 465,96; д) 389,9*D*; е) 3*AC*,8*E*.

9. Выполните сложение и вычитание чисел в двоичной системе счисления:

а) 110010,1101; б) 10101001,100101; в) 1111001,11101;
100101,1010; 10001111,001011; 100011,0011.

10. Выполните сложение и вычитание чисел в восьмеричной системе счисления:

а) 416,25; б) 412,51; в) 2153,134;
165,16; 376,54; 267,326.

11. Выполните сложение и вычитание чисел в шестнадцатеричной системе счисления:

а) 141*C*,*B*5; б) 5*F*2,32; в) *A*8*C*,1*B*;
465,16; 379,5*D*; 2*A*9,8*E*.

12. Выполните сложение над числами в дополнительном и обратном кодах для всех случаев сочетания знаков *A* и *B*, ответ записать в естественной (знаковой) форме:

а) *A* = 11010; б) *A* = 10101; в) *A* = 00111;
*B* = 01011; *B* = 10011; *B* = 10011.

13. Выполните сложение над числами в дополнительном и обратном кодах для всех случаев сочетания знаков *A* и *B*, ответ записать в естественной (знаковой) форме:

а) *A* = 0,10011; б) *A* = 0,10111; в) *A* = 0,00111;
*B* = 0,01011; *B* = 0,00111; *B* = 0,10011.

14. Выполните сложение над числами в модифицированном дополнительном и обратном кодах для всех случаев сочетания знаков *A* и *B*, ответ записать в естественной (знаковой) форме, зафиксировать случаи переполнение разрядной сетки:

а) *A* = 11100; б) *A* = 11011; в) *A* = 11001; г) *A* = 0,10101;
*B* = 11001; *B* = 01111; *B* = 11111; *B* = 0,11101.

15. Выполните умножение над числами в прямом коде и для всех случаев сочетания знаков Мн и Мт в дополнительном коде, использовав для этого все алгоритмы умножения:

а) Мн = 0,01011; б) Мн = 10110;
Мт = 0,01111; Мт = 10010.

16. Выполните деление чисел с восстановлением и без восстановления остатков и деление для всех случаев сочетания знаков Дм и Дт:

а) Дм = 0,00111; б) Дм = 10010;
Дт = 0,10111; Дт = 11110.

## 3. Контроль работы цифрового автомата

Излагаемый ниже материал раздела основан на [1, 2]. В процессе вычислений происходит постоянная передача и преобразование информации, находящейся в памяти ЭВМ, арифметическом или управляющем устройствах. Таким образом, при проектировании ЭВМ необходимо предусмотреть меры как выявления ошибок, так и их исправления. Эта функция возлагается на систему контроля. Система контроля – совокупность аппаратных и программных методов и средств, обеспечивающих определение правильности работы автомата в целом или его отдельных узлов, а также автоматическое исправление выявленных ошибок. Различают следующие виды ошибок вычислений, возникающих:

- из-за погрешностей в исходных данных;
- вследствие методических погрешностей;
- из-за неисправностей в работе машины.

Объектом для системы контроля является третий вид ошибок. Решение задачи контроля возможно только при наличии определенной избыточности информации (аппаратной или информационной). Аппаратная избыточность может заключаться, например, в дублировании арифметических устройств. Логические методы могут основываться на выполнении «двойного счета» и последующего сравнения результатов, проверке соотношений вычисляемых функций ($\sin^2 x + \cos^2 x = 1$), контроле результата при изменении шага вычисления и т. д. Однако эти методы направлены на выявление факта появления ошибки в вычислении, но не определяют место, где она произошла.

### 3.1. Некоторые понятия теории кодирования

*Систематический код* − код, включающий кроме $m$ информационных $k$ контрольных разрядов. При этом возникает избыточность (абсолютная и относительная). Абсолютная избыточность равна $k$, а относительная определяется соотношением $k/n$, где $n = m + k$ – общее количество разрядов в кодовом слове ($m$ – число информационных разрядов).

*Корректирующая способность кода* определяется вероятностью обнаружения или исправления ошибки. Если вероятность искажения одного символа $n$-разрядного слова равна $p$, то вероятность искажения $k$ символов по теореме умножения вероятностей будет равна $\omega = \rho^k (1 - \rho)^{n-k}$.

Число кодовых комбинаций, каждая из которых содержит $k$ искаженных элементов, равна числу сочетаний из $n$ по $k$:

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

Тогда полная вероятность искажения информации определяется по формуле:

$$P_{\sum} = \sum_{i=1}^{k} \frac{n!}{i!(n-i)!} p^i (1-p)^{n-i}.$$

Корректирующая способность кода связана также с понятием кодового расстояния.

Кодовое расстояние $d(A, B)$ для кодовых комбинаций $A$ и $B$ определяется

как вес такой третьей кодовой комбинации, которая является их поразрядной суммой по модулю 2.

Вес кодовой комбинации $V(A)$ – количество единиц, содержащихся в кодовой комбинации.

В теории кодирования [7] показано, что систематический код обладает способностью обнаружить ошибки только тогда, когда минимальное кодовое расстояние для него больше или равно $2t$:

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

где $t$ – кратность обнаруживаемых ошибок.

Это означает, что между соседними разрешенными кодовыми словами должно существовать по крайней мере одно кодовое слово.

В случае если необходимо не только обнаруживать, но и исправлять ошибку (указать место ошибки), минимальное кодовое расстояние должно быть $d_{\min} \geq 2t + 1$.

### 3.2. Обнаружение и исправление одиночных ошибок путем использования дополнительных разрядов

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

Таблица 5

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

Рассмотренный способ контроля по методу четности-нечетности может быть видоизменен для выявления места ошибки в числе. Длинное число разбивается на группы разрядов, каждая из которых содержит $k$ разрядов.

Контрольные разряды выделяются всем группам по строкам и по столбцам согласно следующей схеме:

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

Если ошибка произошла в разряде $a_s$ (единица изменилась на нуль или наоборот), то при проверке на четность (нечетность) сумма по $i$-й строке и $j$-му столбцу, на пересечении которых находится элемент $a_s$, изменится. Следовательно, можно зафиксировать нарушение четности (нечетности) по этой строке и столбцу. Это не только позволит обнаружить ошибку, но и локализовать ее место. Изменив значение разряда $a_s$ на противоположное, можно исправить возникшую ошибку.

Контроль по методу четности-нечетности используется для контроля записи и считывания информации, а также для выполнения арифметических операций.

### 3.3. Коды Хемминга

Американский ученый Р. Хемминг предложил способ кодирования информации, позволяющий не только обнаруживать, но и исправлять одиночные ошибки [1]. Эти коды – систематические. Пусть разрядность слова равна $m$. Для контроля информации требуется $k$ дополнительных разрядов. Число $k$ выбирается согласно следующим правилам.

1. Контролирующее число $k$ выбирается так, чтобы оно имело количество комбинаций, достаточное для распознавания одной из $m + k$ позиций или для сигнализации отсутствия ошибки. Полученное таким образом число описывает $n = m + k + 1$ событий. Следовательно, необходимо, чтобы выполнялось неравенство $2^k \geq (m + n + 1)$.

2. $(m + k)$ − разрядные позиции нумеруются от единицы до $(m + k)$, начиная от младшей значащей. Контрольные разряды $k$ обозначаются $P_0$, $P_1$, $P_2$, …,$P_{k-1}$ и помещаются в разряды, имеющие номера 1, 2, 4, 8, …, $2^{k-1}$ $(m + k)$-разрядного числа. Остальные $m$ разрядов могут быть размещены в любом порядке между контрольными разрядами.

3. Контрольные разряды $P_0$, $P_1$, $P_2$, …, $P_{k-1}$ выбраны таким образом, чтобы для определенных разрядов слова служить в качестве контрольных избыточных разрядов.

| *Проверка* | *Проверяемые разряды* | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | … | | | | |
| 2 | 2 | 3 | 6 | 7 | 10 | 11 | 14 | 15 | 18 | 19 | 22 | … | |
| 3 | 4 | 5 | 6 | 7 | 12 | 13 | 14 | 15 | 20 | 21 | 22 | 23 | … |
| 4 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 24 | 25 | 26 | 27 | … |
| … | | . | . | . | | | | | | | | | |

$P_0$ выбрано с таким расчетом, чтобы в позициях 1, 3, 5, 7, 9, 11 … число единиц каждого слова было четным, $P_1$ выбрано для того, чтобы выполнялось условие четности в разрядах 2, 3, 6, 7, 10, 11, 14, 15 …, аналогично $P_2$ контролирует позиции 4, 5, 6, 7, 12, 13, 14, 15, 20… и $P_3$ − для разрядов 8, 9, 10, 11, 12, 13, 14, 15, 24, 25… .

На основании рассмотренных правил в табл. 6 показаны семиразрядные коды. Контрольные разряды обозначены $P_0$, $P_1$ и $P_2$ и помещены в позициях 1, 2 и 4. Операция обнаружения и исправления ошибок выполняется путем нахождения $k$-разрядного контрольного числа. При этом младший значащий разряд контрольного числа находится посредством проведения контроля на четность над разрядами 1, 3, 5, 7, 9… . Если контроль показывает правильность передачи, то пишется нуль, иначе − единица.

Таблица 6

| Число | Разряды | | | | | | |
|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| | $A$ | $B$ | $C$ | $P_2$ | $D$ | $P_1$ | $P_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 7 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 8 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 10 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 11 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 12 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 13 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 14 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Следующий разряд контрольного числа определяется путем проверки на четность разрядов 2, 3, 7, 10, 11, 14, 15,… . Остальные разряды контрольного числа находятся аналогично.

Если контрольное число равно нулю, то это означает, что при передаче информации ошибка не произошла. Если же контрольное число не равно нулю, то оно указывает на тот разряд числа, где зафиксирована ошибка, которую необходимо исправить.

Пусть, например, передается число шесть 0110011, а принимается в виде 0110111, т. е. произошла ошибка в третьем разряде. Выполняя контроль на четность с помощью разрядов $P_0$, $P_1$ и $P_2$, находим:

| | | | *Контрольное число* |
|---|---|---|---|
| $P_0$ | (1, 3, 5, 7) = (1, 1, 1, 0) | нечетность | 1 |
| $P_1$ | (2, 3, 6, 7) = (1, 1, 1, 0) | нечетность | 1 |

$P_2$ (4, 5, 6, 7) = (0, 1, 1, 0) четность 0

Полученное контрольное число равно 011, что соответствует ошибке в третьем разряде.

Таким образом, дополнительный разряд $P_i$ выбран так, чтобы проверять четность той совокупности разрядных позиций, чьи контрольные числа содержат единицу в позиции $2^i$.

**Контрольные вопросы и задания**

1. Какие виды ошибок могут возникнуть при вычислениях?
2. Что такое систематический код?
3. Что такое корректирующая способность кода?
4. В чем состоит достоинство кода Хемминга?
5. В чем состоит способ контроля по методу четности-нечетности?

## 4. Логические основы вычислительной техники

### 4.1. Двоичные переменные и булевы функции

Для формального описания устройств вычислительной техники при их анализе и синтезе используется аппарат *алгебры логики* [1, 9, 12, 18]. Алгебру логики называют также булевой алгеброй. Алгебра логики – это математический аппарат, с помощью которого записывают (кодируют), упрощают, вычисляют и преобразовывают логические высказывания. Логическое высказывание – это любое утверждение, в отношении которого можно однозначно сказать истинно оно или ложно. Основными понятиями алгебры логики являются двоичные переменные и логические функции.

*Двоичные (логические, булевы) переменные* могут принимать только два значения: 0 (ложь) и 1 (истина) − и обозначаются символами $x_1, x_2, \ldots, x_n$. Двоичные переменные являются аргументами булевых (логических, переключательных) функций.

Булева функция $f$, зависящая от $n$ переменных $x_1, x_2, \ldots, x_n$, называется *булевой*, или переключательной, если функция $f$ и любая из ее переменных $x_i$, ($i = 1, \ldots, n$) принимают значения только из множества {0, 1}.

Иначе говоря, булева функция – это функция и аргументы и значение которой принадлежат множеству {0, 1}. Множество {0, 1} обозначим через $B$.

Булеву функцию от $n$ переменных можно рассматривать как $n$-местную алгебраическую операцию на множестве $B$. При этом алгебра $<B;\Omega>$, где $\Omega$ – множество всевозможных булевых функций, называется *алгеброй логики (булевой алгеброй)*.

Конечность области определения функции имеет существенное достоинство: такие функции можно задавать перечислением значений при различных значениях аргументов (переменных). Для того чтобы задать значение функции от $n$ переменных, надо определить значения для каждого из $2^n$ возможных наборов. Эти значения записывают в таблицу истинности в порядке соответствующих двоичных чисел (рассмотрим позже).

| $x_1$ | $x_2$ | ... | $x_{n-1}$ | $x_n$ | $f$ |
|---|---|---|---|---|---|
| 0 | 0 | ... | 0 | 0 | $f(0, 0, ..., 0, 0)$ |
| 0 | 0 | ... | 0 | 1 | $f(0, 0, ..., 0, 1)$ |
| 0 | 0 | ... | 1 | 0 | $f(0, 0, ..., 1, 0)$ |
| 0 | 0 | ... | 1 | 1 | $f(0, 0, ..., 1, 1)$ |
| ... | ... | ... | ... | ... | . . . |
| 1 | 1 | ... | 0 | 1 | $f(1, 1,..., 0, 1)$ |
| 1 | 1 | ... | 1 | 0 | $f(1, 1,..., 1, 0)$ |
| 1 | 1 | ... | 1 | 1 | $f(1, 1,..., 1, 1)$ |

Для того чтобы задать функцию, достаточно выписать значения $f(0, 0, ..., 0, 0)$, $f(0, 0, ..., 0, 1)$, $f(0, 0, ..., 1, 0)$, $f(0, 0, ..., 1, 1)$,..., $f(1, 1, ..., 0, 0)$, $f(1, 1, ..., 0, 1)$, $f(1, 1, ..., 1, 0)$, $f(1, 1, ..., 1, 1)$. Этот набор называют *вектором значений функции*.

Таким образом, булевы функции на конечном множестве своих аргумен-

тов могут принимать значения 0 и 1 и обозначаются $f(x_1, x_2, \ldots, x_n)$. Булевы функции могут служить аргументами более сложных булевых функций.

### 4.2. Способы задания булевых функций

Для задания произвольной булевой функции широко используются **табличный (матричный)** и **аналитический** способы. При табличном способе булева функция $f(x_1, ...,x_n)$ задается таблицей истинности (табл. 7), в левой части которой представлены все возможные двоичные наборы длины $n$, а в правой указываются значения функции на этих наборах.

Под двоичным набором понимается совокупность значений переменных $x_1, x_2, ..., x_n$ булевой функции $f$. Двоичный набор имеет длину $n$, если он представлен $n$ цифрами из множества {0, 1}. В табл. 7 представлены все двоичные наборы длиной 3.

Иногда двоичные наборы из таблицы истинности булевой функции удобно представлять их номерами. Запишем переменные $x_1, x_2, ..., x_n$ в порядке возрастания их индексов. Тогда любому двоичному набору можно поставить в соответствие целое десятичное число $N$, называемое номером набора. Например, двоичные наборы 011 и 101 имеют номера 3 и 5 соответственно.

Таблица 7

| № набора | $x_1x_2x_3$ | $f$ |
|---|---|---|
| 0 | 0 0 0 | 0 |
| 1 | 0 0 1 | 1 |
| 2 | 0 1 0 | 0 |
| 3 | 0 1 1 | 0 |
| 4 | 1 0 0 | 1 |
| 5 | 1 0 1 | 1 |
| 6 | 1 1 0 | 0 |
| 7 | 1 1 1 | 1 |

Булевы функции, зависящие от большого числа переменных, задавать таблицей истинности неудобно в силу ее громоздкости. Например, таблица истинности булевой функции 8 переменных будет содержать $2^8 = 256$ строк. Для задания функций многих переменных удобно использовать модификацию таблицы истинности. Рассмотрим способ построения такой таблицы истинности для функции $n$ переменных. Множество из $n$ переменных функции разбивается на два подмножества: $x_1, x_2, ..., x_{j-1}$ и $x_j, x_{j+1}, ..., x_n$. Переменными $x_1, x_2, ..., x_{j-1}$ отмечают строки таблицы истинности, задавая в каждой строке значение соответствующего двоичного набора длины $j - 1$. Переменными $x_j, x_{j+i}, ..., x_n$ отмечают ее столбцы, задавая в каждом столбце значения соответствующего двоичного набора длиной $n - j + 1$. Значение функции записывается в клетке на пересечении соответствующей строки и столбца (табл. 8).

При аналитическом способе булева функция задается формулой, т. е. аналитическим выражением, построенным из операций булевой алгебры. Анали-

тический способ задания булевых функций занимает особое место в проектировании цифровых автоматов. Фактически все преобразования над булевыми функциями, необходимые для построения цифровых автоматов, ведутся с использованием аналитического представления функций.

Таблица 8

| $x_1, x_2, \ldots, x_{j-1}$ | $x_j, x_{j+1}, \ldots, x_n$ | | | |
|---|---|---|---|---|
| | 00...0 | 0...1 | ... | 11...1 |
| 00...0 | 0 | 1 | ... | 1 |
| 00...1 | 1 | 0 | ... | 0 |
| ... | ... | ... | ... | ... |
| 11...1 | 1 | 0 | ... | 1 |

### 4.3. Основные логические операции и логические элементы

Логический элемент в электронных схемах – это устройство, реализующее некоторую операцию алгебры логики. При этом логические сигналы 0 и 1 задаются разными уровнями напряжения. Обычно сигнал логического нуля задается низким уровнем напряжения, а единица – высоким.

Логические схемы состоят из множества логических элементов, выполняющих логические операции. Для изображения логических элементов используются условные графические обозначения (УГО), описывающие только выполняемую элементом функцию и не зависящие от его внутреннего устройства. Существует несколько общепринятых стандартов условных обозначений логических элементов. Среди них наиболее распространенными являются американский стандарт *milspec*806B и стандарт МЭК 117-15 A, созданный Международной Электротехнической Комиссией. Часто в литературе используются обозначения элементов согласно европейского стандарта *DIN* 4070. В отечественной литературе УГО элементов, в основном, приводятся в ГОСТ 2.743−82.

Существует не более чем $2^k$ (где $k = 2^n$) различных булевых функций $n$ переменных [1]. К этому выводу легко прийти, пользуясь простыми комбинаторными рассуждениями, и вспомнив, что на каждом из $2^n$ наборов функции могут принимать два значения.

Функций от одной переменной четыре: это *константа* 0 ($f_0$), *константа* 1 ($f_1$), *тождественная функция* ($f_2$), т. е. функция, значение которой совпадает со значением переменной, и функция *отрицания* ($f_3$), значение которой противоположно значению переменной. Отрицание будем обозначать $\overline{x}$.

| $x$ | $f_0$ | $f_1$ | $f_2$ | $f_3$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |

Функции от некоторого числа переменных можно рассматривать как функции от большего числа переменных, при этом значения функции не меняются при изменении этих «добавочных» переменных. Такие переменные назы-

ваются фиктивными, в отличие от остальных – существенных (действительных).

Переменная $x_i$ называется *фиктивной* (несущественной) переменной функции $f(x_1, \ldots, x_n)$, если

$f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$

для любых значений $x_1, \ldots, x_{i-1}, x_{i+1}, \cdots, x_n$. Иначе переменная $x_i$ называется *существенной*.

Функции двух переменных представлены в табл. 9.

Таблица 9

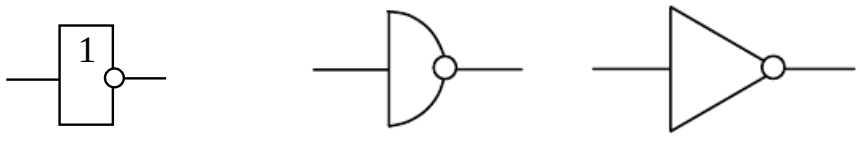
| $x_1x_2$ | $f_0$ | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ | $f_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 01 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 10 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 11 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Отметим наиболее часто используемые функции из числа приведенных в таблице:

- $f_0(x_1, x_2) = 0$ – тождественный ноль (константа 0);
- $f_1(x_1, x_2) = x_1 \cdot x_2$ – конъюнкция (логическое произведение, И); обозначается знаком &, /\ или · (символы операции конъюнкции в логических выражениях можно опускать, например, $x_1x_2$);
- $f_3(x_1, x_2) = x_1$ – повторение $x_1$;
- $f_5(x_1, x_2) = x_2$ – повторение $x_2$;
- $f_6(x_1, x_2) = x_1 \oplus x_2$ – сложение по модулю 2 или mod 2;
- $f_7(x_1, x_2) = x_1 \vee x_2$ – дизъюнкция (логическое сложение, ИЛИ); обозначается v или +;
- $f_8(x_1, x_2) = x_1 \downarrow x_2$ – функция Вебба (стрелка Пирса, ИЛИ-НЕ);
- $f_9(x_1, x_2) = x_1 \sim x_2$ – эквивалентность;
- $f_{13}(x_1, x_2) = x_1 \rightarrow x_2$ – импликация;
- $f_{14}(x_1, x_2) = x_1 \setminus x_2$ – штрих Шеффера (И-НЕ);
- $f_{15}(x_1, x_2) = 1$ – тождественная единица (константа 1).

Основными операциями булевой алгебры являются: отрицание, логическое сложение и логическое умножение. В булевой алгебре возведение в степень и извлечение корня являются вырожденными логическими операциями, поскольку значения, принимаемые аргументами функции при возведении в степень и извлечении корня, остаются неизменными, если принять справедливость равенств $1 \cdot 1 = 1$ и $0 \cdot 0 = 0$. Операции вычитания и деления не рассматриваются и не допускаются.

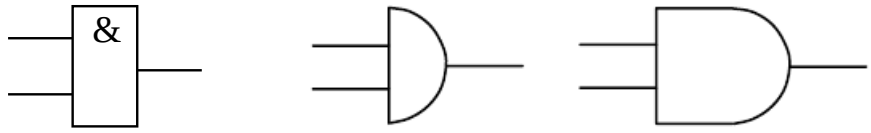
**Логическое отрицание** (**функция НЕ**). Логическим отрицанием высказывания $x$ называется такое сложное высказывание $f(x)$, которое истинно, когда $x$ ложно, и наоборот. Функция НЕ записывается следующим образом: $f = \overline{x}$. УГО элемента реализующего функцию НЕ приведено на рис. 15.




*а* *б* *в*

Рис. 15. УГО логического элемента НЕ:
*а* − по ГОСТ и МЭК; *б* − по DIN; *в* − по milspec

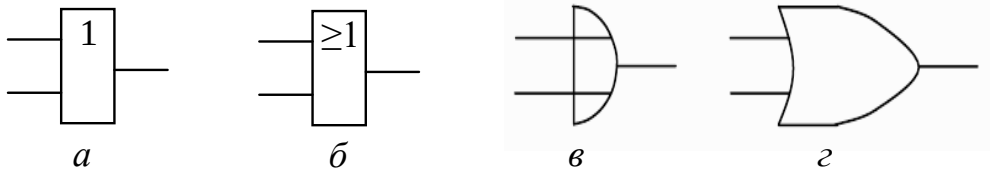
**Логическое умножение (конъюнкция)**. Конъюнкция (функция И) двух переменных $x_1$ и $x_2$ − это сложное высказывание, которое истинно только тогда, когда истинны $x_1$ и $x_2$, и ложно для всех остальных наборов переменных. Булева функция конъюнкции имеет вид $f = x_1x_2$. Для обозначения операции конъюнкции используются также символы & и Λ. Функция логического умножения (И) от $n$ переменных имеет вид $f_2(x_1, x_2, \ldots, x_n) = x_1\, x_2\ \ldots\ x_n = \Lambda\, x_i$. УГО элемента, реализующего операцию логического умножения, приведено на рис. 16.




*а* *б* *в*

Рис. 16. УГО логического элемента И:
*а* − по ГОСТ и МЭК; *б* − по DIN; *в* − по milspec

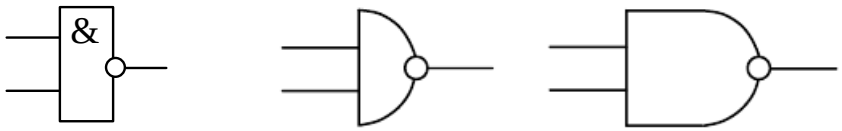
**Логическое сложение (дизъюнкция)**. Дизъюнкция (функция ИЛИ) двух переменных $x_1$ и $x_2$ – это сложное высказывание, которое истинно тогда, когда истинна хотя бы одна из переменных $x_1$ и $x_2$, и ложно, когда они обе ложны. Булева функция дизъюнкции имеет вид $f = x_1 \vee x_2$. Для обозначения операции дизъюнкции используется также символ v. Функция логического сложения (ИЛИ) от $n$ переменных имеет вид $f_2(x_1, x_2, \ldots, x_n) = x_1 \vee x_2 \vee \ldots \vee x_n = \vee\, x_i$. УГО элемента, реализующего операцию логического сложения, изображено на рис. 17.




*а* *б* *в* *г*

Рис. 17. УГО логического элемента ИЛИ:
*а* − по ГОСТ; *б* − по МЭК; *в* − по DIN; *г* − по milspec

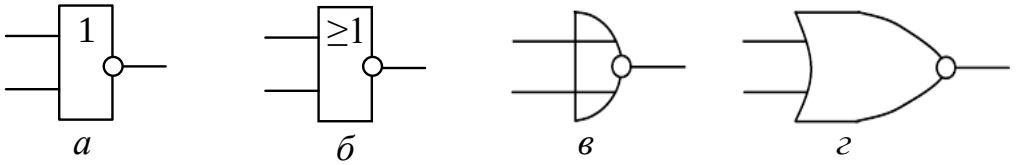
**Отрицание конъюнкции (операция штрих Шеффера)**. Отрицание конъюнкции (функция И-НЕ) двух переменных $x_1$ и $x_2$ – сложное высказывание, ложное только при истинности обоих переменных $x_1$ и $x_2$. Булева функция И-НЕ имеет вид $f = \overline{x_1\, x_2}$. УГО элемента, реализующего указанную операцию, изображено на рис. 18 и называется элементом Шеффера или элементом И-НЕ.




*а* *б* *в*

Рис. 18. УГО логического элемента И-НЕ:
*а* − по ГОСТ и МЭК; *б* − по DIN; *в* − по milspec

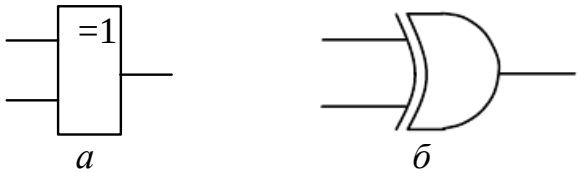
**Отрицание дизъюнкции (операция «стрелка Пирса», функция Вебба)**. Отрицание дизъюнкции (функция ИЛИ-НЕ) двух переменных $x_1$ и $x_2$ – сложное высказывание, истинное только тогда, когда обе переменные принимают ложное значение. Булева функция ИЛИ-НЕ имеет вид $f = \overline{x_1 \vee x_2}$. УГО элемента, реализующего указанную операцию, приведено на рис. 19 и называется элементом Пирса или элементом ИЛИ-НЕ.




*а* *б* *в* *г*

Рис. 19. УГО логического элемента ИЛИ-НЕ:
*а* − по ГОСТ; *б* − по МЭК; *в* − по DIN; *г* − по milspec

**Сложение по модулю 2**. Сложение по модулю 2 − это сложное высказывание, которое истинно только тогда, когда истинна только одна из переменных $x_1$ и $x_2$. Булева функция «сумма по модулю 2» имеет вид $f = x_1 \oplus x_2$. Если число переменных $n > 2$, то функция истинна на тех наборах, в которых число единиц нечетно. УГО элемента, реализующего операцию «сумма по модулю 2» изображено на рис. 20.




*а* *б*

Рис. 20. УГО логического элемента сложение по модулю 2:
*а* − по ГОСТ и МЭК; *б* − по milspec

**Импликация** − это высказывание, принимающее ложное значение только в случае, если $x_1$ истинно, а $x_2$ ложно.

Простейшие булевы функции позволяют строить новые булевы функции с помощью обобщенной операции, называемой **операцией суперпозиции**.

*Суперпозицией булевых функций* $f_0$ и $f_1, ..., f_n$ называется функция $f(x_1, ..., x_m) = f_0(g_1(x_1, ..., x_m), ..., g_k(x_1, ..., x_m))$, где каждая из функций $g_i(x_1, ..., x_m)$ либо совпадает с одной из переменных (тождественная функция), либо – с одной из функций $f_1, ..., f_n$.

Функция $f(x, y) = (\overline{x \,\&\, y})$ является суперпозицией функций $\overline{\ }$ и &. Функция $g(x, y) = x \oplus (x \vee y)$ является суперпозицией функций $\oplus$ и $\vee$. Функция $h(x, y, z) = (x \,\&\, y) \oplus z$ – суперпозиция функций $\oplus$ и &.

Суперпозиция функций одного аргумента порождает функции одного аргумента. Суперпозиция функций двух аргументов дает возможность строить функции любого числа аргументов. Суперпозиция булевых функций представляется в виде логических формул. Однако необходимо отметить следующее:

- одна и та же функция может быть представлена разными формулами;
- каждой формуле соответствует своя суперпозиция и, следовательно, при ее реализации, своя схема соединений элементов;
- между формулами представления булевых функций и схемами, их реализующими, существует взаимно однозначное соответствие. Исключение составляют те функции, для которых отсутствуют элементы их реализующие, например, импликация.

Очевидно, среди схем, реализующих данную функцию, есть наиболее простая (например, по количеству элементов, образующих эту схему, и числу связей между ними). Поиск логической формулы, соответствующей этой схеме, представляет большой практический интерес. Преобразование формул булевых функций основано на использовании соотношений булевой алгебры.

### 4.4. Основные законы булевой алгебры

Основные законы булевой алгебры [1, 2, 12, 14, 18] позволяют проводить эквивалентные преобразования булевых функций, записанных с помощью операций И, ИЛИ, НЕ, приводить их к удобному для дальнейшего использования виду и упрощать запись.

При выполнении преобразований функций алгебры логики могут быть полезны следующие соотношения:

- всегда истинны высказывания: $x \vee 1 = 1$; $x \vee \overline{x} = 1$; $\overline{0} = 1$;
- всегда ложны высказывания: $x \cdot 0 = 0$; $x \cdot \overline{x} = 0$; $\overline{1} = 0$;
- правило двойного отрицания: $\overline{\overline{x}} = x$;
- правило повторения (идемпотентности): $x \vee x \vee \ldots \vee x = x$;
  $x \cdot x \cdot \ldots \cdot x = x$.

*Закон рефлексивности:*

- для дизъюнкции: $x \vee x = x$;
- для конъюнкции: $x \cdot x = x$.

*Переместительный закон:*

- для дизъюнкции: $x_1 \vee x_2 = x_2 \vee x_1$;
- для конъюнкции: $x_1 \cdot x_2 = x_2 \cdot x_1$;
- для сложения по модулю два: $x_1 \oplus x_2 = x_2 \oplus x_1$.

*Сочетательный закон:*

- для дизъюнкции: $x_1 \vee (x_2 \vee x_3) = (x_1 \vee x_2) \vee x_3$;
- для конъюнкции: $x_1 \cdot (x_2 \cdot x_3) = (x_1 \cdot x_2) \cdot x_3$;
- для суммы по модулю два: $x_1 \oplus (x_2 \oplus x_3) = (x_1 \oplus x_2) \oplus x_3$, т. е. группирова-

ние переменных внутри дизъюнкции (конъюнкции) не изменяет значений функции.

*Распределительный закон:*

– для дизъюнкции: $x_1 \vee x_2 \cdot x_3 = (x_1 \vee x_2)(x_1 \vee x_3)$, т. е. дизъюнкция переменной и конъюнкции эквивалентна конъюнкции дизъюнкций этой переменной с сомножителями;

– для конъюнкции: $x_1 \cdot (x_2 \vee x_3) = x_1 \cdot x_2 \vee x_1 \cdot x_3$, т. е. конъюнкция переменной и дизъюнкции равносильна дизъюнкции конъюнкций этой переменной со слагаемыми.

*Закон инверсии* (*правило де Моргана*):

– для дизъюнкции: $\overline{x_1 \vee x_2} = \overline{x_1} \cdot \overline{x_2}$;

– для конъюнкции: $\overline{x_1 \cdot x_2} = \overline{x_1} \vee \overline{x_2}$, т. е. отрицание дизъюнкции (конъюнкции) переменных равно конъюнкции (дизъюнкции) отрицаний этих переменных.

Правило де Моргана справедливо для любого числа переменных:

$\overline{x_1 \vee x_2 \vee \ldots \vee x_n} = \overline{x_1} \cdot \overline{x_2} \cdot \ldots \cdot \overline{x_n}$,

$\overline{x_1 \cdot x_2 \cdot \ldots \cdot x_n} = \overline{x_1} \vee \overline{x_2} \vee \ldots \vee \overline{x_n}$.

Справедливы также следствия из правила де Моргана:

$\overline{\overline{x_1 \vee x_2}} = \overline{\overline{x_1} \cdot \overline{x_2}}$,

$\overline{\overline{x_1 \cdot x_2}} = \overline{\overline{x_1} \vee \overline{x_2}}$.

Переместительный и сочетательный законы для дизъюнкции и конъюнкции, а также распределительный закон для конъюнкции совпадают с законами обычной алгебры. Но в обычной алгебре нет законов, аналогичных распределительному для дизъюнкции и законам инверсии. Их справедливость доказывается посредством составления таблиц истинности для левой и правой частей формулы.

*Правило склеивания:* $x_1 \cdot x_2 \vee x_1 \cdot \overline{x_2} = x_1$.

Следующие соотношения могут быть легко выведены из рассмотренных выше.

*Закон поглощения:* $x_1 \vee x_1 \cdot x_2 = x_1$ ;

$x_1 \cdot (x_1 \vee x_2) = x_1$;

$x_1 \vee x_1 \cdot x_2 = x_1 \cdot 1 \vee x_1 \cdot x_2 = x_1 \cdot (1 \vee x_2) = x_1 \cdot 1 = x_1$;

$x_1 \cdot (x_1 \vee x_2) = x_1 \cdot x_1 \vee x_1 \cdot x_2 = x_1 \vee x_1 \cdot x_2 = x_1$.

Использование основных законов и правил булевой алгебры позвляет выполнить равносильные преобразования логических формул, что приведет к их упрощению или приведению к определенному виду. На приводимых ниже примерах показаны некоторые приемы и способы, применяемые при упрощении логических формул.

$\overline{x \vee y} \cdot (x \cdot \overline{y}) = \overline{x} \cdot \overline{y} \cdot (x \cdot \overline{y}) = \overline{x} \cdot x \cdot \overline{y} \cdot \overline{y} = 0 \cdot \overline{y} \cdot \overline{y} = 0 \cdot \overline{y} = 0$.

В этом примере законы булевой алгебры применяются в следующей последовательности: правило де Моргана, сочетательный закон, правило операций переменной с ее инверсией и правило операций с константами.

$\bar{x}\cdot y\vee\overline{x\vee y}\vee x = \bar{x}\cdot y\vee\bar{x}\cdot\bar{y}\vee x = \bar{x}\cdot(y\vee\bar{y})\vee x = \bar{x}\vee x = 1.$

В этом выражении применяется правило де Моргана, выносится за скобки общий множитель, используется правило операций переменной с ее инверсией.

$(x\vee y)\cdot(\bar{x}\vee y)\cdot(\bar{x}\vee\bar{y}) = (x\vee y)\cdot(\bar{x}\vee y)\cdot(\bar{x}\vee y)\cdot(\bar{x}\vee\bar{y}) = y\cdot\bar{x}.$

Здесь применяется правило повторения (идемпотентности), затем комбинируются два первых и два последних сомножителя и используется закон склеивания.

$x\cdot\bar{y}\vee\bar{x}\cdot y\cdot z\vee x\cdot z = x\cdot\bar{y}\vee\bar{x}\cdot y\cdot z\vee x\cdot z\cdot(y+\bar{y}) =$

$= x\cdot\bar{y}\vee\bar{x}\cdot y\cdot z\vee x\cdot y\cdot z\vee x\cdot\bar{y}\cdot z = (x\cdot\bar{y}\vee x\cdot\bar{y}\cdot z)\vee(\bar{x}\cdot y\cdot z\vee x\cdot y\cdot z) =$

$= x\cdot\bar{y}\vee y\cdot z.$

Вводится вспомогательный логический сомножитель $(y+\bar{y})$, затем комбинируются два крайних и два средних логических слагаемых и используется закон поглощения.

$\overline{\overline{x\cdot y}\vee z} = \overline{\overline{x\cdot y}}\cdot\bar{z} = (\bar{x}\vee\bar{y})\cdot z.$

Сначала добиваемся, чтобы знак отрицания стоял только перед отдельными переменными, а не перед их комбинациями. Для этого дважды применяем правило де Моргана; затем используем закон двойного отрицания.

$x\cdot y\vee x\cdot y\cdot z\vee x\cdot z\cdot p = x\cdot(y\cdot(1\vee z)\vee z\cdot p) = x\cdot(y\vee z\cdot p).$

В выражении сначала выносятся за скобки общие множители, затем применяется правило операций с константами.

$x\vee\overline{y\cdot\bar{z}}\vee\overline{x\vee y\vee z} = x\vee\bar{y}\vee z\vee\bar{x}\cdot\bar{y}\cdot\bar{z} = x\vee\bar{y}\vee z\vee x\cdot\bar{y}\cdot z =$

$= x\vee z\vee(\bar{y}\vee x\cdot\bar{y}\cdot z) = x\vee z\vee\bar{y}.$

К отрицаниям неэлементарных формул применяется правило де Моргана, далее используются законы двойного отрицания и склеивания.

$x\cdot\bar{y}\vee x\cdot y\cdot z\vee x\cdot\bar{y}\cdot z\vee x\cdot\overline{y\cdot z} = x\cdot(\bar{y}\vee y\cdot z\vee\bar{y}\cdot z\vee\overline{y\cdot z}) =$

$= x\cdot((\bar{y}\vee\bar{y}\cdot z)\vee(y\cdot z\vee\overline{y\cdot z})) = x\cdot(\bar{y}\vee\bar{y}\cdot z\vee 1) = x\cdot 1 = x.$

Общий множитель $x$ выносится за скобки; комбинируются слагаемые в скобках: первое с третьим и второе с четвертым; к дизъюнкции $y\cdot z+\overline{y\cdot z}$ применяется правило операции переменной с ее инверсией.

$(x\cdot\bar{y}\vee z)\cdot(\bar{x}\vee y)\vee\bar{z} = x\cdot\bar{y}\cdot\bar{x}\vee x\cdot\bar{y}\cdot y\vee z\cdot\bar{x}\vee z\cdot y\vee\bar{z} =$

$= 0\vee 0\vee z\cdot\bar{x}\vee z\cdot y\vee\bar{z} = z\cdot\bar{x}\vee(z\vee\bar{z})\cdot(y\vee\bar{z}) = z\cdot\bar{x}\vee 1\cdot(y\vee\bar{z}) =$

$= z\cdot\bar{x}\vee y\vee\bar{z} = (z\cdot\bar{x}\vee\bar{z})\vee y = (z\vee\bar{z})\cdot(\bar{x}\vee\bar{z})\vee y = 1\cdot(\bar{x}\vee\bar{z})\vee y =$

$= \bar{x}\vee\bar{z}\vee y.$

Используются распределительный закон для дизъюнкции, правило операции переменной с ее инверсией, правило операций с константами, переместительный закон и распределительный закон для конъюнкции.

$$x\cdot y\cdot(\bar{x}\cdot z\vee\overline{\overline{x\cdot\bar{y}\cdot z}}\vee z\cdot t)=x\cdot y\cdot(\bar{x}\cdot z\vee\overline{\overline{x\cdot\bar{y}}}\vee\bar{z}\vee z\cdot t)\ =$$

$$=\ x\cdot y\cdot(\bar{x}\cdot z\vee x\cdot y\vee\bar{z}\vee z\cdot t)\ =\ x\cdot y\vee x\cdot y\cdot\bar{z}\vee x\cdot y\cdot z\cdot t=x\cdot y.$$

Используются правило де Моргана, закон двойного отрицания и закон поглощения.

### 4.5. Формы представления булевых функций

Основными понятиями, лежащими в основе представления булевых функций в различных формах, являются понятия элементарной конъюнкции и элементарной дизъюнкции.

*Элементарной конъюнкцией* называется логическое произведение любого конечного числа различных между собой булевых переменных, взятых со знаком инверсии или без него.

Например, логические выражения вида $x_1\bar{x}_2\bar{x}_3$, $\bar{x}_1x_4$, $x_1x_2x_4$ являются элементарными конъюнкциями, а выражения вида $\overline{x_1x_2x_3}$, $\overline{\bar{x}_1x_4}$, $x_1\overline{x_2x_4}$ не являются элементарными конъюнкциями.

*Элементарной дизъюнкцией* называется логическая сумма любого конечного числа различных между собой булевых переменных, взятых со знаком инверсии или без него

Примером логического выражения, являющегося элементарной дизъюнкцией, могут служить $x_1\vee x_2\vee\bar{x}_3$, $x_1\vee\bar{x}_4$, $x_1\vee\bar{x}_2\vee\bar{x}_4$, а выражения вида $\overline{x_1\vee x_2\vee\bar{x}_3}$, $\overline{x_1\vee x_4}$, $\bar{x}_1\vee\overline{x_2\vee x_4}$ не являются элементарными дизъюнкциями.

*Дизъюнктивной нормальной формой* (ДНФ) булевой функции называется дизъюнкция конечного числа элементарных конъюнкций:

$f_{\text{ДНФ}}=x_1\bar{x}_2\bar{x}_3\vee x_1x_4\vee x_2x_3\bar{x}_4\vee x_1\bar{x}_2x_3.$

Число переменных, входящих в элементарную конъюнкцию, определяет *ранг* этой конъюнкции.

*Совершенной ДНФ* (СДНФ) булевой функции от $n$ переменных называется такая ДНФ, в которой все конъюнкции имеют ранг $n$. СДНФ записывается по таблице истинности согласно правилу: *для каждого набора переменных, на котором булева функция принимает единичное значение, записывается конъюнкция ранга n и все эти конъюнкции объединяются знаками дизъюнкции; переменная имеет знак инверсии, если на соответствующем наборе имеет нулевое значение*:

$f_{\text{СДНФ}}=x_1x_2\bar{x}_3\vee\bar{x}_1\bar{x}_2x_3\vee\bar{x}_1x_2\bar{x}_3\vee x_1x_2x_3.$

В общем виде это можно записать следующим образом:

$$f(x_1,x_2,\ ...,\ x_n)=\vee x_1^{\sigma}x_2^{\sigma}...\ x_n^{\sigma},$$

где

$$x^{\sigma}=\left\{\begin{array}{l}\bar{x},\ \text{если}\ \sigma=0,\\ x,\ \text{если}\ \sigma=1.\end{array}\right\}$$

Элементарные конъюнкции, образующие СДНФ, называют также *кон-*

*ституентами* (составляющими) *единицы* (минтерм), так как они соответствуют наборам, при которых функция принимает значение, равное единице.

Конъюнктивной нормальной формой (КНФ) булевой функции называется конъюнкция конечного числа элементарных дизъюнкций:

$f_{\text{КНФ}} = (x_1 \vee \overline{x_2} \vee \overline{x_3})(\overline{x_1} \vee \overline{x_4})(x_2 \vee \overline{x_3} \vee x_4)(x_1 \vee x_2 \vee x_3)$.

*Совершенной КНФ* (СКНФ) булевой функции от *n* переменных называется такая КНФ, в которой все дизъюнкции имеют ранг *n*. СКНФ записывается по таблице истинности согласно правилу: *для каждого набора переменных, на котором булева функция принимает нулевое значение, записывается дизъюнкция ранга n и все эти дизъюнкции объединяются знаками конъюнкции; переменная имеет знак инверсии, если на соответствующем наборе имеет единичное значение*:

$f_{\text{СКНФ}} = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})(x_1 \vee \overline{x_2} \vee \overline{x_3})(\overline{x_1} \vee x_2 \vee x_3)(x_1 \vee x_2 \vee x_3)$.

Элементарные дизъюнкции, образующие СКНФ, называют *конституентами* (составляющими) *нуля* (макстерм), так как они соответствуют наборам, при которых функция принимает нулевое значение.

### 4.6. Системы булевых функций

Любая булева функция может быть представлена аналитически одной из вышерассмотренных нормальных форм, которые используют ограниченное число элементарных булевых функций. Например, для СДНФ такими функциями являются «конъюнкция», «дизъюнкция» и «отрицание». Следовательно, существуют системы булевых функций, с помощью которых можно аналитически представить любую сложную булеву функцию. Проектирование цифровых устройств основано на знании таких систем булевых функций. Последнее особенно важно для определения набора элементарных логических схем, из которых может быть построено требуемое цифровое устройство.

*Функционально полной системой булевых функций* (ФПСБФ) называется совокупность таких булевых функций ($f_1$, $f_2$, ..., $f_k$), посредством которых можно записать произвольную булеву функцию *f*.

Это обусловливает целесообразность постановки задачи определения свойств, которыми должны обладать функции, составляющие ФПСБФ.

Решение этой задачи основано на понятии замкнутого относительно операции суперпозиции класса функций. Класс булевых функций, функционально замкнутый по операции суперпозиции, есть множество функций, любая суперпозиция которых дает функцию, также принадлежащую этому множеству. Среди функционально замкнутых классов выделяют классы обычного типа, называемые предполными, которые обладают следующими свойствами. Предполный класс *S* не совпадает с множеством *P* всех возможных булевых функций, однако если в него включить любую не входящую в *S* булеву функцию, то новый функционально замкнутый класс будет совпадать с множеством *P*. Проведенные исследования показали, что предполных классов пять, а для построения ФПСБФ необходимо и достаточно, чтобы ее функции не содержались полностью ни в одном из пяти предполных классов.

Наряду с нормальными формами представления функций алгебры логики в вычислительной технике широко используются логические полиномиальные формы. Преобразования над формулами булевых функций иногда удобно выполнять в алгебре **Жегалкина**. Алгебра Жегалкина включает две двухместные операции: конъюнкцию и сложение по модулю 2, а также константу 1.

**Теорема Жегалкина**. Любая булева функция может быть представлена многочленом вида

$$f\left(x_1x_2...x_n\right) = k_0 \oplus k_1x_1 \oplus k_2x_2 \oplus ... \oplus k_{n+1}x_1x_2 \oplus k_{n+2}x_1x_3 \oplus ... \oplus k_{2^n-1}x_1x_2...x_n,$$

где $k_i$ – коэффициент, принимающий значения 0 или 1.

Теорема позволяет представить любую булеву функцию в виде полиномов различной степени.

Задача построения полинома Жегалкина сводится к нахождению коэффициентов $k_i$. Для любых конституент единицы $k_1$ и $k_2$ имеет место следующее соотношение: $k_1 \vee k_2 = k_1 \oplus k_2$. Оно позволяет выполнить переход от СДНФ к полиному Жегалкина. Для этого достаточно заменить в СДНФ символ «+» (дизъюнкции) на символ «⊕» и выполнить подстановку вида $\overline{x} = x \oplus 1$ с последующими преобразованиями в алгебре Жегалкина.

Перечислим предполные классы булевых функций [1,9].

*Класс линейных функций*. Булева функция называется линейной, если ее можно представить полиномом первой степени:

$$f\left(x_1x_2...x_n\right) = k_0 \oplus k_1x_1 \oplus k_2x_2 \oplus ... \oplus k_nx_n.$$

Примем без доказательства утверждение, что при суперпозиции линейных функций получаем линейную функцию. Существует восемь линейных функций от двух переменных (табл. 10). Следовательно, в функционально полном базисе должна содержаться хотя бы одна нелинейная булева функция.

Таблица 10

| $k_2$ $k_1$ $k_0$ | $f(x_1x_2) = k_0 \oplus k_1x_1 \oplus k_2x_2$ |
|---|---|
| 0 0 0 | 0 |
| 0 0 1 | 1 |
| 0 1 0 | $x_1$ |
| 0 1 1 | $x_1 \oplus 1$ |
| 1 0 0 | $x_2$ |
| 1 0 1 | $x_2 \oplus 1$ |
| 1 1 0 | $x_1 \oplus x_2$ |
| 1 1 1 | $1 \oplus x_1 \oplus x_2$ |

*Класс функций, сохраняющих нуль*. К булевым функциям, сохраняющим константу 0, относят такие булевы функции $f(x_1, ..., x_n)$, для которых справедливо соотношение $f(0, ..., 0) = 0$.

*Класс функций, сохраняющих единицу.* К булевым функциям, сохраняющим константу 1, относят такие булевы функции $f(x_1, ..., x_n)$, для которых справедливо соотношение $f(1, ..., 1) = 1$.

*Класс монотонных функций*. Булева функция называется монотонной, если при любом возрастании набора переменных значения этой функции не убывают. Двоичный набор $A = (a_1, a_2, ..., a_n)$ не меньше двоичного набора $B = (b_1, b_2, ..., b_n)$ , если для каждой пары $(a_i, b_i)$ $i = 1...n$ справедливо соотношение $a_i \geq b_i$.

Если для двух наборов $A$ и $B$ существует $a_i < b_i$ (например, $f(0,1)$ и $f(1,0)$), то наборы считаются несравнимыми. Таким образом, если $f(0,0) \geq f(0,1) \geq f(1,1)$ или $f(0,0) \geq f(1,0) \geq f(1,1)$, то функция $f$ является монотонной.

*Класс самодвойственных функций*. Булевы функции $f_1(x_1, x_2, ..., x_n)$ и $f_2(x_1, x_2, ..., x_n)$ называются двойственными друг другу, если выполняется соотношение $f_1(x_1, x_2, ..., x_n) = \overline{f_2}(\overline{x_1}, \overline{x_2}, ..., \overline{x_n})$.

К самодвойственным булевым функциям относят такие булевы функции, которые двойственны по отношению к самим себе, т. е. булева функция называется самодвойственной, если на любых двух противоположных наборах $x_1, x_2, ..., x_n$ и $\overline{x}_1, \overline{x}_2, ..., \overline{x}_n$ она принимает противоположные значения $f(x_1, x_2, ..., x_n) = \overline{f}(\overline{x_1}, \overline{x_2}, ..., \overline{x_n})$.

Любая булева функция, полученная с помощью операции суперпозиции и подстановки из функций одного класса, принадлежит этому же классу.

*Базисом* называется полная система булевых функций, с помощью которой любая булева функция может быть представлена суперпозицией исходных функций.

Приведем без доказательства формулировку теоремы о функциональной полноте. Доказательство ее приведено в [12].

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

- не являющуюся линейной;
- не сохраняющую нуль;
- не сохраняющую единицу;
- не являющуюся монотонной;
- не являющуюся самодвойственной.

Иначе говоря, система булевых функций является функционально полной тогда и только тогда, когда она целиком не содержится ни в одном из предполных классов.

Рассмотрим примеры ФПСБФ. Для удобства изложения материала сведем элементарные булевы функции двух переменных и некоторые функции одной переменной в таблицу Поста (табл. 11), классифицируя каждую из них по признакам принадлежности к предполным классам.

Таблица 11

| Функция | Вид функции | | | | |
|---|---|---|---|---|---|
| | немонотонные | нелинейные | несамодвойственные | не сохраняющие 0 | не сохраняющие 1 |
| $F_0 \equiv 0$ | – | – | + | – | + |
| $F_1 = x_1 \cdot x_2$ | – | + | + | – | – |
| $F_3 = x_1$ | – | – | – | – | – |
| $F_5 = x_2$ | – | – | – | – | – |
| $F_6 = x_1 \oplus x_2$ | + | – | + | – | + |
| $F_7 = x_1 \vee x_2$ | – | + | + | – | – |
| $F_8 = x_1 \downarrow x_2$ | + | + | + | + | + |
| $F_9 = x_1 \sim x_2$ | + | – | + | + | – |
| $F_{13} = x_1 \rightarrow x_2$ | + | + | + | – | + |
| $F_{14} = x_1 / x_2$ | + | + | + | + | + |
| $F_{15} \equiv 1$ | – | – | + | + | – |

Из таблицы видно, что каждая из функций $F_8$ и $F_{14}$ является ФПСБФ. Иными словами, используя, например, только булеву функцию $F_{14}$ – «штрих Шеффера», можно записать в виде формулы любую булеву функцию. Признаком функциональной полноты, очевидно, является наличие плюса в каждом столбце таблицы хотя бы для одной из составляющих систему булевых функций. К таким ФСПБФ, наиболее распространенным в практике построения цифровых автоматов, следует отнести:

$\{\wedge, \vee, \text{не}\}$, $\{\wedge, \oplus, \text{не}\}$, $\{\wedge, \oplus, 1\}$, $\{\wedge, \text{не}\}$, $\{\vee, \text{не}\}$.

Иногда удобно строить ФПСБФ при наличии констант, т. е. булевых функций «константа 0», «константа 1». Как следует из таблицы, функция «константа 0» несамодвойственна и не сохраняет 1; функция «константа 1» несамодвойственна и не сохраняет 0. Вместе с тем константы являются линейными и монотонными функциями. Отсюда непосредственно (на основании теоремы о функциональной полноте) вытекает следующее: система булевых функций является ослабленно функционально полной, если она содержит хотя бы одну нелинейную и одну немонотонную булевы функции. Примерами ослабленных ФПСБФ могут служить следующие системы:

$\{\wedge, \oplus\}$, $\{\wedge, \sim\}$, $\{\vee, \oplus\}$, $\{\vee, \sim\}$, $\{\rightarrow\}$.

Логические элементы, реализующие булевы функции функционально полного набора, образуют *функционально полный набор логических элементов,* с помощью которых можно построить любую логическую схему.

Базис может быть избыточным и минимальным.

Минимальный базис – такой базис, когда удаление одной (любой) функции превращает систему булевых функций в неполную. Иначе говоря, функционально полный базис – это набор операций булевой алгебры (и соответствую-

щих им элементов), позволяющих построить любую булеву функцию.

Функционально полным базисом является базис И, ИЛИ, НЕ. В то же время он функционально избыточен. Удаление из него элемента И или ИЛИ превращает его в минимальный базис.

Для примера рассмотрим базис, образованный, например, элементами И и НЕ. В этом базисе реализуем функцию ИЛИ, тем самым докажем функциональную полноту выбранного базиса (рис. 21).

$$x_1 \vee x_2 = \overline{\overline{x_1 \vee x_2}} = \overline{\overline{x_1} \cdot \overline{x_2}}\,.$$

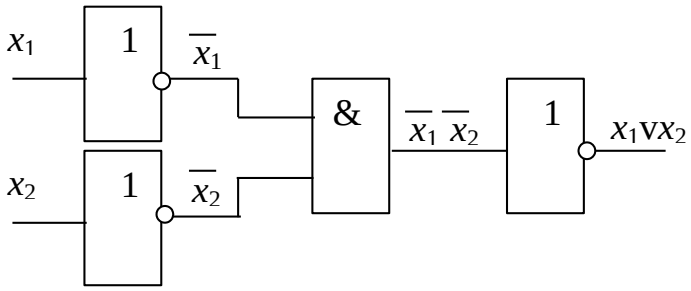



Рис. 21. Реализация операции ИЛИ в базисе И, НЕ

### 4.7. Кубическое задание булевых функций

Булева функция может быть задана:

- аналитически;
- словесным описанием;
- таблицей истинности;
- картами (диаграммами) Венна, Вейча, Карно;
- логической схемой.

Более компактной формой записи булевых функций является форма, когда вместо полного перечисления всех конъюнкций (дизъюнкций) используют номера наборов, на которых функция принимает единичное значение. Так, например, форма записи $f(x_1x_2x_3)$ = V $F(0, 2, 3)$ означает, что функция от трех переменных принимает единичное значение на 0, 2 и 3 наборах таблицы истинности. Такая форма записи называется *числовой*.

Некоторые методы минимизации ориентируются на задание булевой функции в виде кубического покрытия. При этом булева функция удобно интерпретируется с использованием ее геометрического представления. Так, функцию двух переменных можно интерпретировать как плоскость, заданную в системе координат $x_1x_2$. Получится квадрат, вершины которого соответствуют комбинациям переменных. Для геометрической интерпретации функции трех переменных используется куб, заданный в системе координат $x_1x_2x_3$. Геометрическое представление функции двух и трех переменных приведено на рис. 22.

Кубическое представление булевых функций наиболее пригодно для машинных методов их анализа, так как позволяет компактно представлять булевы функции от большого количества переменных. Это свойство определило наибольшую распространенность кубического представления булевых функций

при автоматизации проектирования цифровой аппаратуры. Именно поэтому кубическое представление булевых функций заслуживает более подробного рассмотрения.

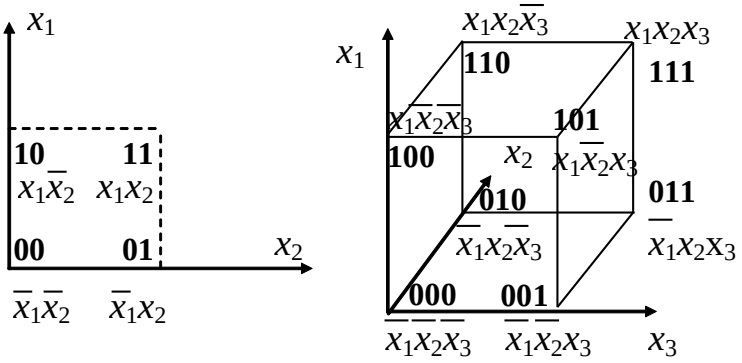



Рис. 22. Геометрическое представление функции двух и трех переменных

Для представления булевых функций в кубической форме используется кубический алфавит {0, 1, *x*}, основанный на двух логических примитивах 0 и 1, символ *x* в этом алфавите может рассматриваться как *универсум*, т. е. $x = 0 \vee 1$. Этот символ называется *свободной (независимой)* координатой. Он может принимать как нулевое, так и единичное значение, остальные компоненты называются *связанными*. Куб, содержащий свободные координаты, покрывает кубы, на которых он образован. Количество символов *x* в кубе определяет его *размерность (ранг)*. Например, куб 0011 имеет *нулевую размерность* и в геометрическом представлении это вершина; куб 0*x*11 – *первую размерность и представляет собой* ребро, покрывающее две вершины (0111 и 0011); куб *xx*11 имеет вторую размерность, представляет собой грань, покрывающую два набора первой размерности (или четыре набора нулевого размерности) и т. д. Кубы, образующиеся в результате последовательного выполнения операции склеивания, назовем *r*-кубами, где *r* – размерность полученного куба, покрывающего $2^r$ двоичных наборов.

Любой набор переменных в кубическом представлении булевых функций принято называть *кубом* в *n*-мерном пространстве. Переменные куба принято называть *координатами*.

Совокупность кубов, покрывающих все входные наборы функции, называется *покрытием* или *кубическим покрытием*. Множество кубов, на которых функция равна нулю, называют нулевым покрытием ($C^0$), а на которых равна единице – единичным покрытием ($C^1$). Обозначение $C^0$ и $C^1$ происходит от английского слова «*Cover*» – покрывать. В дальнейшем, при рассмотрении алгоритмов, множества $C^0$ и $C^1$ будем обозначать соответственно *D* и *L*.

Способ кубического представления булевых функций и операции над кубами в этом представлении образуют *кубическое исчисление*.

Новое представление булевой функции получается путем отображения булевой функции *n* переменных на *n*-мерный куб (*n*-куб).

Для отображения булевой функции *n* переменных на *n*-кубе устанавливается соответствие между членами СДНФ и вершинами *n*-куба. На *n*-кубе опре-

делим координатную систему с координатами ($e_1$, ..., $e_n$), $e_i \subseteq \{0, 1\}$.

Установим соответствие между членом СДНФ $x_1^{e1} x_2^{e2} ... x_n^{en}$ и некоторой вершиной $e_1e_2....e_n$ куба. При этом $x_i^{ei} = x_i$, если $e_i = 1$, и $x_i^{ei} = \overline{x_i}$, если $e_i = 0$.

Рассмотрим сказанное выше на примере следующей функции.

$f = \overline{x_1}x_2x_3 \vee x_1\overline{x}_2\overline{x_3} \vee x_1\overline{x_2}x_3 \vee x_1x_2\overline{x_3} \vee x_1x_2x_3$.

Как следует из таблицы истинности (табл. 12), функция $f$ определена на трех группах наборов переменных: $L = \{3, 4, 5, 6, 7\}$, $D = \{0, 2\}$ и $N = \{1\}$, где $N$ – множество наборов, на которых функция $f$ не определена.

Конъюнкции, образованные максимальным количеством букв (максимального ранга, конституенты единицы) принято называть 0-кубами, в геометрическом представлении это вершины. Множество 0-кубов образуют кубический комплекс $K^0$. Для функции, заданной табл. 12 комплекс $K^0$ имеет вид

$$K^0 = \left\{ \begin{matrix} 000 \\ 001 \\ 100 \\ 101 \\ 111 \end{matrix} \right\}.$$

Таблица 12

| | $x_1$ | $x_2$ | $x_3$ | $f$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 |
| 7 | 1 | 1 | 1 | 1 |

Над 0-кубами, кодовое расстояние которых равно 1, выполняется операция склеивания, в результате которой образуются новые кубы, содержащие свободные (независимые) координаты.

Геометрическая интерпретация сказанного выше приведена на рис. 23. В результате склеивания кубов 101 и 111 (0-кубы, вершины) образован 1-куб 1$x$1 (ребро), а 1-кубов 00$x$ и 10$x$ – 2-куб $x$0$x$ (грань).

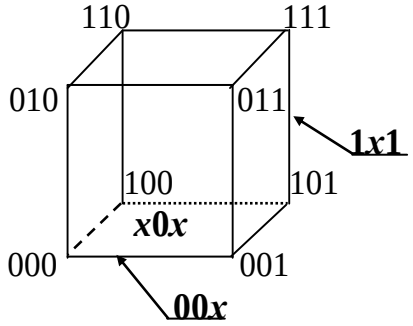



Рис. 23. Образование новых кубов

Кубическое представление булевых функций позволяет обойтись тремя переменными 0, 1, $x$ вместо $x_1$, $x_2$, ..., $x_n$.

Количество свободных координат в кубе определяет его размерность $r$. Чем больше $r$, тем больше свободных координат и тем меньше входов будет иметь реализующая этот куб схема (логический элемент).

Цена схемы, в общем случае, определяется количеством входов элементов, используемых для ее реализации:

$$C = \sum_{i=1}^{k} (n - r_i) + k ,$$

где $k$ – количество полученных кубов; $n - r_i$ – количество единичных и нулевых значений.

Два $r$-куба могут образовать $r + 1$-куб, если в этих $r$-кубах все координаты, в том числе и свободные, совпадают, за исключением лишь какой-либо одной, которая в этих кубах имеет противоположное значение.

На рис. 24 приведено изображение куба, соответствующего булевой функции от четырех переменных (гиперкуб).

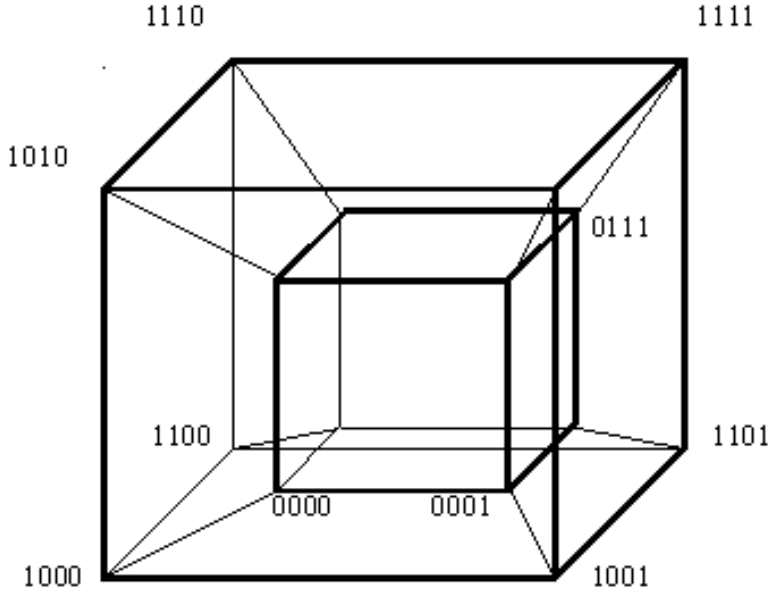



Рис. 24. Геометрическое представление функции четырех переменных

Как следует из рисунка, геометрическое представление 4-куба уже довольно сложное. Поэтому для функций, зависящих более чем от четырех переменных, предпочтительным является аналитическое представление булевых функций.

### 4.8. Характеристики логических схем

Преобразования булевых функций применяются для получения компактных выражений, на основе которых строятся логические схемы. Эти схемы называются комбинационными, так как значения на их выходах зависят только от комбинаций входных сигналов.

В результате преобразований получается обычно несколько решений. Для сравнения различных вариантов схем, реализующих полученные функции, используют их основные характеристики: сложность и быстродействие.

Сложность схемы оценивается количеством оборудования, составляющего схему. Если ориентироваться на произвольную элементную базу, то для оценки затрат оборудования можно использовать оценку сложности схем по Квайну.

Сложность (цена) по Квайну определяется суммарным числом входов логических элементов из которых состоит схема. При такой оценке единица сложности – один вход логического элемента. Цена инверсного входа обычно принимается равной двум. Такой подход к оценке сложности оправдан по следующим причинам:

− сложность схемы легко вычисляется по булевым функциям, на основе которых строится схема: для ДНФ сложность схемы равна сумме количества букв (букве со знаком отрицания соответствует цена 2) и количества знаков дизъюнкции, увеличенного на 1 для каждого дизъюнктивного выражения;

− рассматриваемые в учебном пособии классические методы минимизации булевых функций обеспечивают минимальность схемы именно в смысле цены по Квайну.

Практика показывает, что схема с минимальной ценой по Квайну обычно реализуется наименьшим числом конструктивных элементов – корпусов интегральных микросхем.

Быстродействие схемы оценивается максимальной задержкой сигнала при прохождении его от входа схемы к выходу, т. е. определяется промежутком времени от момента поступления входных сигналов до момента установления соответствующих значений выходных. Задержка сигнала кратна числу элементов, через которые проходит сигнал от входа к выходу схемы. Поэтому быстродействие схемы характеризуется значением $k \cdot t_{эл}$, где $t_{эл}$ – задержка сигнала на одном элементе. Значение $k$ определяется количеством уровней (рангом) схемы, и рассчитывается следующим образом. Входам приписывается нулевой уровень. Логические элементы, связанные только со входами схемы относятся к первому уровню. Элемент относится к уровню $n$, если он связан по входам с элементами нижних уровней. Пример определения ранга $r$ схемы приведен на рис. 25. В приведенном примере ранг $r = 3$.

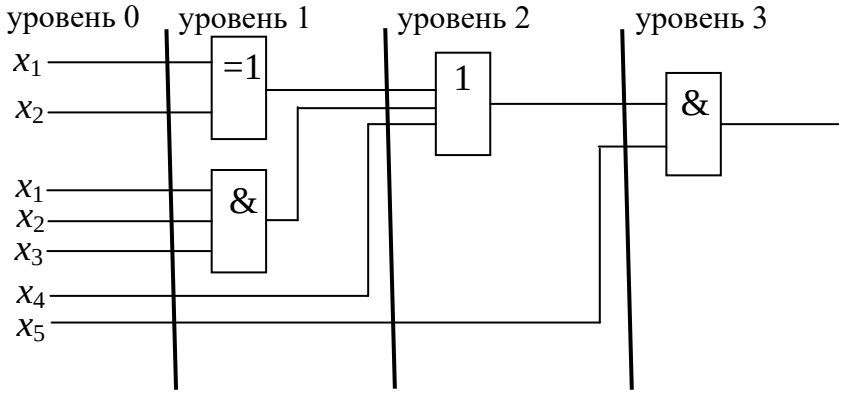



Рис. 25. Определение ранга схемы.

### 4.9. Минимизация булевых функций

При проектировании цифровых автоматов широко используются методы минимизации булевых функций [12], позволяющие получать экономичные схемы. Общая задача минимизации булевых функций может быть сформулирована

следующим образом: найти эквивалентное выражение заданной функции алгебры логики в форме, содержащей минимально возможное число букв. Хотя, в общем случае, под минимизацией может иметься в виду получение выражений с минимальным числом инверсных переменных либо с минимальным числом вхождений какой-либо одной переменной и т. п. Следует отметить, что в общей постановке данная задача пока не решена, однако хорошо исследована в классе дизъюнктивно-конъюнктивных форм.

*Минимальной дизъюнктивной нормальной формой* (МДНФ) булевой функции называется ДНФ, содержащая наименьшее число букв (по отношению ко всем другим ДНФ, представляющим заданную функцию).

Функция $g(x_1, ..., x_n)$ называется *импликантой* функции $f(x_1, ..., x_n)$, если для любого набора переменных, на котором $g = 1$, справедливо $f = 1$. Рассмотрим на примере функции, заданной таблицей истинности (табл. 13), введенные выше понятия.

Таблица 13

| $x_3x_2x_1$ | $f$ | $g_1$ | $g_2$ | $g_3$ | $g_4$ | $g_5$ | $g_6$ | $g_7$ |
|---|---|---|---|---|---|---|---|---|
| 000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 001 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 011 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 101 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 110 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 111 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

$f = \overline{x_1}x_2x_3 \vee x_1x_2\overline{x_3} \vee x_1x_2x_3 = g_7$;

$g_1 = x_1x_2\overline{x_3}$;

$g_2 = x_1\overline{x_2}x_3$;

$g_3 = \overline{x_1}x_2x_3 \vee x_1x_2\overline{x_3} = x_1x_2\,(x_3 \vee \overline{x_3}) = x_1x_2$;

$g_4 = \overline{x_1}x_2x_3$;

$g_5 = \overline{x_1}x_2x_3 \vee x_1x_2\overline{x_3} = x_2x_3$;

$g_6 = \overline{x_1}x_2x_3 \vee x_1x_2\overline{x_3}$.

Импликанта $g$ булевой функции $f$, являющаяся элементарной конъюнкцией, называется *простой*, если никакая часть импликанты $g$ не является импликантой функции $f$. Иными словами *простая импликанта* − импликанта, к которой нельзя применить операцию склеивания.

Из примера видно, что импликанты $g_3 = x_1x_2$ и $g_5 = x_2x_3$ являются простыми импликантами функции $f$. Импликанты $g_1$, $g_2$, $g_4$, $g_6$ не являются простыми, так как их части являются импликантами функции $f$, например, $g_1$ является частью $g_3$. Приведем без доказательства два утверждения, полезные при получении минимальной ДНФ.

1. Дизъюнкция любого числа импликант булевой функции $f$ также является импликантой этой функции.

2. Любая булева функция $f$ эквивалентна дизъюнкции всех своих простых импликант. Такая форма представления булевой функции называется *сокращенной* ДНФ.

Перебор всех возможных импликант для булевой функции $f$ из рассмотренного примера дает возможность убедиться, что простых импликант всего две: $g_3$ и $g_5$. Следовательно, сокращенная ДНФ функции $f$ имеет вид:

$f = g_3 \vee g_5 = x_1x_2 \vee x_2x_3$.

Как видно из табл. 13, импликанты $g_3$, $g_5$ в совокупности покрывают своими единицами все единицы функции f. Получение сокращенных ДНФ является первым этапом отыскания минимальных форм булевых функций. Как уже отмечалось, в сокращенную ДНФ входят все простые импликанты булевой функции. Иногда из сокращенной ДНФ можно исключить одну или несколько простых импликант, исключение которых не приводит к изменению значений функции на всевозможных значениях ее переменных. Такие простые импликанты назовем *лишними*.

Можно также ввести понятие существенной (обязательной) простой импликанты. *Существенная импликанта* − простая импликанта, образованная склеиванием таких минтермов, что по крайней мере для одного из них эта операция была единственной. Существенные импликанты образуют *ядро функции*.

Сокращенная ДНФ булевой функции называется *тупиковой*, если в ней отсутствуют лишние простые импликанты.

Исключение лишних простых импликант из сокращенной ДНФ булевой функции не является однозначным процессом, т. е. булева функция может иметь несколько тупиковых ДНФ.

*Кратчайшая ДНФ* − тупиковая ДНФ, имеющая минимальное число простых импликант.

*МДНФ (минимальная ДНФ)* − тупиковая ДНФ с минимальным числом вхождений переменных (минимальным числом букв) по сравнению с другими тупиковыми формами этой функции [12]. МДНФ тоже может быть несколько. Глушков В.М. отмечает также, что минимальная ДНФ булевой функции представляет собой дизъюнкцию некоторых простых импликант этой функции.

Минимизировать функции, т. е. находить наиболее простое выражение для исходной функции, можно различными методами. Все они практически различаются лишь на первом этапе − этапе получения сокращенной ДНФ. Следует отметить, что, к сожалению, поиск МДНФ всегда связан с некоторым перебором решений. Рассмотрим некоторые из них.

### 4.9.1. Метод Квайна

Более подробно метод минимизации Квайна изложен в [12].

**Теорема Квайна**. Для получения минимальной формы булевой функции необходимо в СДНФ произвести все возможные склеивания и поглощения так, чтобы в результате была получена сокращенная ДНФ. Сокращенная ДНФ в общем случае может содержать лишние простые импликанты, которые необходимо выявить и удалить из нее на втором этапе минимизации.

На первом этапе выполняется переход от функции, заданной в форме СДНФ, к сокращенной ДНФ. Это основано на использовании следующих соотношений:

1) *операция неполного склеивания* $Fx \vee F\bar{x} = Fx \vee F\bar{x} \vee F$, где $Fx$ и $F\bar{x}$ – две конъюнкции, а $F$ – конъюнкция, полученная в результате их склеивания (обычного) по $x$;

2) *операция поглощения* $F \vee F\tilde{x} = F$.

Справедливость обеих операций легко проверяется. Суть метода заключается в последовательном выполнении всех возможных склеиваний и затем всех поглощений, что приводит к сокращенной ДНФ. *Метод применим к совершенной ДНФ*. Из соотношения поглощения следует, что произвольное элементарное произведение поглощается любой его частью. Для доказательства достаточно показать, что произвольная простая импликанта $p = x_1x_2 \ldots x_n$ может быть получена. В самом деле, применяя к $p$ операцию развертывания (обратную операции склеивания) $F = F(x \vee \bar{x}) = Fx \vee F\bar{x}$ по всем недостающим переменным $x_k, \ldots, x_m$ исходной функции $f$, получаем совокупность $S$ конституент единицы. При склеивании всех конституент из $S$ получим импликанту $p$. Последнее очевидно, поскольку операция склеивания обратна операции развертывания. Множество $S$ конституент обязательно присутствует в совершенной ДНФ функции $f$, поскольку $p$ – ее импликанта.

В результате выполнения склеивания получается конъюнкция $n - 1$ ранга, а конъюнкции $Fx$ и $F\bar{x}$ остаются в исходном выражении и участвуют в сравнении с другими членами СДНФ. Таким образом, удается снизить ранг термов.

Склеивание и поглощение выполняются до тех пор, пока имеются члены, не участвовавшие в попарном сравнении. Термы, подвергшиеся операции склеивания, отмечаются. Неотмеченные термы представляют собой простые импликанты и включаются в сокращенную ДНФ. Все отмеченные конъюнкции ранга $n - 1$ подвергаются вновь операции склеивания до получения термов $n - 2$ ранга и так далее до тех пор, пока количество неотмеченных конъюнкций больше 2. В результате выполнения первого этапа получена сокращенная ДНФ.

Полученное логическое выражение не всегда оказывается минимальным. На втором этапе переходят от сокращенной ДНФ к тупиковым ДНФ и среди них выбирают МДНФ.

Для формирования тупиковых ДНФ строится *импликантная таблица* (*матрица*), строки которой отмечаются простыми импликантами сокращенной ДНФ, а столбцы – конституентами единицы исходной СДНФ. В строке напротив каждой простой импликанты ставится метка под теми наборами (конституентами единицы), на которых она принимает значение 1. Соответствующие конституенты поглощаются (покрываются) данной простой импликантой.

Из общего числа простых импликант необходимо отобрать их минимальное число, исключив лишние. Формирование тупиковых форм и выбор минимального покрытия начинается с выявления обязательных простых импликант,

т. е. таких, которые (и только они) покрывают некоторый исходный набор. Рассмотрим пример минимизации методом Квайна булевой функции

$$f_{\text{сокр ДНФ}} = \text{V}(1,2,5,6,7) = \underset{1}{\bar{x}_1\bar{x}_2x_3} \vee \underset{2}{\bar{x}_1x_2\bar{x}_3} \vee \underset{3}{x_1\bar{x}_2x_3} \vee \underset{4}{x_1x_2\bar{x}_3} \vee \underset{5}{x_1x_2x_3}.$$

Выполним операцию склеивания:

1 – 3 ($x_1$) $\bar{x}_2x_3$ 1 (первый и третий наборы склеиваются по $x_1$)
2 – 4 ($x_1$) $x_2\bar{x}_3$ 2 (второй и четвертый наборы – по $x_1$)
3 – 5 ($x_2$) $x_1x_3$ 3 (третий и пятый наборы – по $x_2$)
4 – 5 ($x_3$) $x_1x_2$ 4 (четвертый и пятый наборы – по $x_3$)

В результате выполнения первого шага склеивания получаем четыре новые конъюнкции, простых импликант из множества конституент не выявлено. Полученные конъюнкции более не склеиваются и образуют сокращенную ДНФ.

$$f_{\text{сокр ДНФ}} = \bar{x}_2x_3 \vee x_2\bar{x}_3 \vee x_1x_3 \vee x_1x_2.$$

Для выявления обязательных простых импликант и формирования на их основе минимального покрытия строится импликантная таблица (табл. 14).

Таблица 14

| | $\bar{x}_1\bar{x}_2x_3$ | $\bar{x}_1x_2\bar{x}_3$ | $x_1\bar{x}_2x_3$ | $x_1x_2\bar{x}_3$ | $x_1x_2x_3$ |
|---|---|---|---|---|---|
| $\bar{x}_2x_3$ | * | | * | | |
| $x_2\bar{x}_3$ | | * | | * | |
| $x_1x_3$ | | | * | | * |
| $x_1x_2$ | | | | * | * |

В строках импликантной таблицы записываются простые импликанты, а в столбцах – конституенты единицы. Звездочка ставится, если простая импликанта покрывает конституенту.

Простые импликанты $\bar{x}_2x_3$ и $x_2\bar{x}_3$ являются обязательными, так как только они покрывают конституенты $\bar{x}_1\bar{x}_2x_3$ и $\bar{x}_1x_2\bar{x}_3$, и включаются в минимальное покрытие. Эти же импликанты покрывают $x_1\bar{x}_2x_3$ и $x_1x_2\bar{x}_3$. Остается одна непокрытая конституента $x_1x_2x_3$, которая может быть покрыта одной из двух оставшихся простых импликант – $x_1x_3$ или $x_1x_2$. Это приводит к получению двух тупиковых форм:

$f_{\text{МДНФ}} = \bar{x}_2x_3 \text{ v } x_2\bar{x}_3 \text{ v } x_1x_3$ – первая тупиковая форма;
$f_{\text{МДНФ}} = \bar{x}_2x_3 \text{ v } x_2\bar{x}_3 \text{ v } x_1x_2$ – вторая тупиковая форма.

### 4.9.2. Метод Блейка – Порецкого

При минимизации функции методом Квайна требуется предварительно представить ее в СДНФ, что часто связано с дополнительными преобразованиями. Если исходить из произвольной ДНФ, то для получения промежуточной сокращенной ДНФ возможно применить метод Блейка–Порецкого [12]. Метод базируется на применении формулы обобщенного склеивания:

$$Ax \vee B\bar{x} = Ax \vee B\bar{x} \vee AB,$$

справедливость которой легко доказать. Действительно,

$$Ax = Ax \vee ABx, \qquad B\bar{x} = B\bar{x} \vee AB\bar{x}.$$

Следовательно,

$$Ax \vee B\bar{x} = Ax \vee ABx \vee B\bar{x} \vee AB\bar{x} = Ax \vee B\bar{x} \vee AB.$$

В основу метода положено следующее утверждение: если в произвольной ДНФ булевой функции *f* произвести все возможные обобщенные склеивания, а затем выполнить все поглощения, то в результате получится сокращенная ДНФ функции *f*.

Рассмотрим пример. Пусть булева функция *f* задана произвольной ДНФ.

$$f_{\text{ДНФ}} = \bar{x}_1\bar{x}_2 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2.$$

Необходимо, используя метод Блейка – Порецкого, получить сокращенную ДНФ функции *f*. Проводим обобщенные склеивания. Легко видеть, что первый и второй элемент исходной ДНФ допускают обобщенное склеивание по переменной $x_1$. В результате склеивания получим

$$\bar{x}_1\bar{x}_2 \vee x_1\bar{x}_2\bar{x}_3 = \bar{x}_1\bar{x}_2 \vee x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_2\bar{x}_3.$$

Первый и третий элемент исходной ДНФ допускают обобщенное склеивание как по переменной $x_1$, так и по $x_2$. После склеивания по $x_1$ имеем

$$\bar{x}_1\bar{x}_2 \vee x_1x_2 = \bar{x}_1\bar{x}_2 \vee x_1x_2 \vee x_2\bar{x}_2.$$

После склеивания по $x_2$ имеем

$$\bar{x}_1\bar{x}_2 \vee x_1x_2 = \bar{x}_1\bar{x}_2 \vee x_1x_2 \vee x_1\bar{x}_1.$$

Второй и третий элемент ДНФ допускают обобщенное склеивание по переменной $x_2$. После склеивания получаем

$$x_1\bar{x}_2\bar{x}_3 \vee x_1x_2 = x_1\bar{x}_2\bar{x}_3 \vee x_1x_2 \vee x_1\bar{x}_3.$$

Выполнив последнее обобщенное склеивание, приходим к ДНФ

$$f = \bar{x}_1\bar{x}_2 \vee x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_2\bar{x}_3 \vee x_1x_2 \vee x_1\bar{x}_3.$$

После выполнения поглощений получаем

$$f = \bar{x}_1\bar{x}_2 \vee \bar{x}_2\bar{x}_3 \vee x_1x_2 \vee x_1\bar{x}_3.$$

Попытки дальнейшего применения операции обобщенного склеивания и поглощения не дают результата. Следовательно, получена сокращенная ДНФ функции *f*.

### 4.9.3. Метод минимизирующих карт Вейча (Карно)

Рассмотрим графический (визуальный) метод минимизации булевых функций с помощью карт (диаграмм) Вейча, который является одним из наиболее удобных методов при небольшом числе переменных (до шести). Карты Карно были изобретены в 1952 Эдвардом В. Вейчем и усовершенствованы в 1953 Морисом Карно. Карта Вейча представляет собой развертку *n*-мерного куба на плоскости. При этом вершины куба представляются клетками карты, каждой из которых поставлена в соответствие конституента единицы или нуля. Переменные, обозначающие клетки диаграммы, расставляются таким образом, чтобы наборы, записанные в двух смежных клетках, имели кодовое расстояние, равное единице. *Кодовым расстоянием* между двумя наборами называется количество отличающихся координат (разрядов).

Например, если $A = 0xx0$ и $B = 01x1$, то кодовое расстояние между ними равно 2. Поскольку наборы с кодовым расстоянием равным единице располагаются в смежных клетках, они получили название *соседних наборов*. В клетку карты, соответствующую конституенте единицы, заносится 1, иначе − 0. Таким образом, для минимизации функции она должна быть представлена в форме СДНФ. Минимизация булевой функции с использованием карт в дизъюнктивной (конъюнктивной) форме заключается в объединении единичных (нулевых) клеток в контуры, каждому такому контуру соответствует простая импликанта.

Можно сформулировать следующие правила минимизации с использованием минимизирующих карт:

1. В карте Вейча (Карно) группы единиц (для получения МДНФ) и группы нулей (для получения МКНФ) необходимо обвести контурами. Внутри контура должны находиться только одноименные значения функции. Этот процесс соответствует операции *склеивания* или нахождения импликант данной функции.

2. Количество клеток внутри контура должно быть равно степени двойки.

3. При проведении контуров крайние строки карты (верхние и нижние, левые и правые), а также угловые клетки считаются соседними.

4. Каждый контур должен включать максимально возможное количество клеток. В этом случае он будет соответствовать простой импликанте (имплиценте).

5. Все единицы (нули) в карте (даже одиночные) должны быть охвачены контурами. Любая единица (нуль) может входить в контуры произвольное количество раз.

6. Число контуров должно быть минимальным. Множество контуров, покрывающих все единицы (нули) функции образуют тупиковую ДНФ (КНФ). *Целью минимизации* является нахождение кратчайшей из множества тупиковых форм.

7. В элементарной конъюнкции (дизъюнкции), которая соответствует одному контуру, остаются только те переменные, значение которых не изменяется внутри обведенного контура. Переменные булевой функции входят в элементарную коньюнкцию (для значений функции 1) без инверсии, если их значение на соответствующих координатах равно единице и с инверсией, если равно нулю. Для значений булевой функции, равных нулю, записываются элементарные дизьюнкции, куда переменные входят без инверсии, если их значение на соответствующих координатах равно нулю и с инверсией - если равно единице.

Рассмотрим преведенные правила на примере минимизации функции.

Пусть задана функция $f_{СДНФ} = x_1x_2x_3 \vee x_1x_2\bar{x}_3 \vee x_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3$.

На рис. 26 показана заполненная карта Вейча, соответствующая функции $f_{СДНФ}$. На карте обозначены четыре контура, каждый из которых содержит по две клетки. Контур 2 можно считать лишним, так как он покрывает клетки, уже

покрытые двумя другими контурами (1 и 3). Аналогично можно считать лишним контур 3 (покрывается контурами 2 и 4). Здесь возможны две тупиковые формы булевой функции. Таким образом, по данной карте может быть получена одна из тупиковых форм:

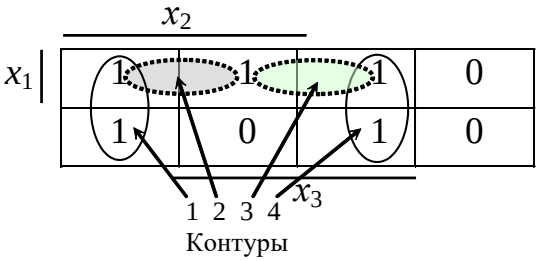



Рис. 26. Карта Вейча для $f_{\text{СДНФ}}$

$f_{\text{МДНФ}} = \overline{x}_2 x_3 \vee x_2 \overline{x}_3 \vee x_1 x_2$ – первая тупиковая форма;

$f_{\text{МДНФ}} = \overline{x}_2 x_3 \vee x_2 \overline{x}_3 \vee x_1 x_3$ – вторая тупиковая форма.

Если функция задана в форме ДНФ, то необязательно ее приводить к форме СДНФ, что является одним из преимуществ карты Вейча. Для этого рассматривается каждый дизъюнктивный член функции в отдельности, и в соответствующие ему клетки карты заносятся единицы.

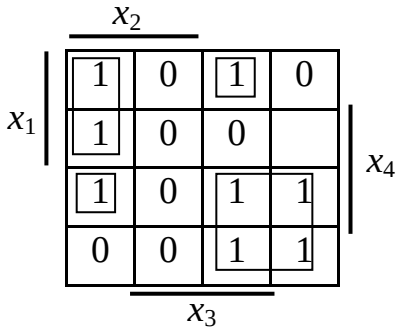



Рис. 27. Карта Вейча для $f_{\text{ДНФ}}$

Рассмотрим сказанное на примере функции

$$f_{\text{ДНФ}} = \overline{x}_1 \overline{x}_2 \vee x_1 x_2 \overline{x}_3 \vee x_1 \overline{x}_2 x_3 \overline{x}_4 \vee \overline{x}_1 x_2 \overline{x}_3 x_4.$$

Первому члену ДНФ поставлены в соответствие четыре клетки карты, второму – две клетки, третьему и четвертому − по одной клетке соответственно (рис. 27). Далее объединение единиц в контуры и выбор их минимального числа осуществляются рассмотренным выше методом.

Если выбраны самые большие контуры и использовано по возможности меньшее их число, то будет получена дизъюнктивная нормальная форма с меньшей ценой по Квайну. Дальнейшее упрощение можно получить за счет выполнения скобочных преобразований. Выражению с меньшим числом вхождений букв соответствует схема, имеющая меньшее число входов элементов, так что упрощение функций ведет к упрощению реализующих их схем.

Принимая во внимание клетки карт, содержащие нули, и поступая с ними так же, как мы поступали с клетками, содержащими единицы, можно получать конъюнктивные нормальные формы (рис. 28).

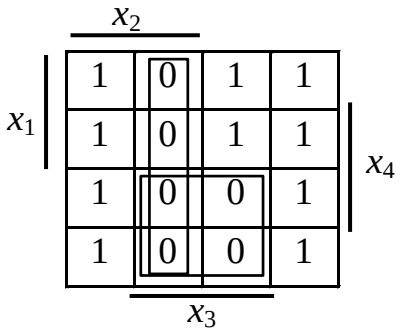



Рис. 28. Карта Вейча для $f_{\text{КНФ}}$

$f_{\text{КНФ}} = (\overline{x}_2 \vee \overline{x}_3)(x_1 \vee \overline{x}_3)$.

Если булева функция задана, например, таблицей истинности, то более удобной для графического представления функции является карта Карно. Карты Карно являются разновидностью карт Вейча.

Карта Карно является *координатным* способом представления булевых функций. Она представляет собой развертку на плоскости $n$-мерного единично-

го куба, т.е. клетки карты соответствуют координатам куба.

Переменные функции разбиваются на две группы так, что одна группа определяет координаты столбца карты, а другая − координаты строки. При таком способе построения каждая клетка определяется значениями переменных, соответствующих определенному двоичному набору. Внутри каждой клетки карты Карно ставится значение функции на данном наборе. Переменные в строках и столбцах располагаются так, чтобы соседние клетки карты Карно *различались только в одном разряде переменных,* т. е. были соседними. Поэтому значения переменных в столбцах и в строках карты образуют *r*-разрядный код Грея. Код Грея – двоичный код, в котором рядом стоящие наборы – соседние (их кодовое расстояние равно единице). В карте Карно каждой клетке соответствует код, состоящий из кода строки и кода столбца (рис. 29).

| $x_1x_2$ \ $x_3x_4$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0000 | 0001 | 0011 | 0010 |
| 01 | 0100 | 0101 | 0111 | 0110 |
| 11 | 1100 | 1101 | 1111 | 1110 |
| 10 | 1000 | 1001 | 1011 | 1010 |

Рис. 29. Структура карты Карно

На рис. 30 показано соответствие клеток карты Карно и строк таблицы истинности (см. рис. 30, *а*). При этом в карте (см. рис. 30, *б*) показаны координаты единичных и нулевых значений функции, а в карте (см. рис. 30, *в*) показано соответствие строк таблицы истинности и ячеек карты.

| № | $x_1$ $x_2$ $x_3$ | $f$ |
|---|---|---|
| 0 | 0 0 0 | 1 |
| 1 | 0 0 1 | 1 |
| 2 | 0 1 0 | 0 |
| 3 | 0 1 1 | 0 |
| 4 | 1 0 0 | 0 |
| 5 | 1 0 1 | 1 |
| 6 | 1 1 0 | 1 |
| 7 | 1 1 1 | 0 |

*а*

| $x_1$ \ $x_2x_3$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 000 | 001 | 011 | 010 |
| 1 | 100 | 101 | 111 | 110 |

*б*

| $x_1$ \ $x_2x_3$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 3 | 2 |
| 1 | 4 | 5 | 7 | 6 |

*в*

Рис. 30. Таблица истинности и карта Карно

Рассмотрим на примере (рис. 31) использование карты Карно для минимизации булевой функции в форме ДНФ и КНФ.

| $x_1x_2$ \ $x_3x_4$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 0 | 1 |
| 01 | 1 | 1 | 0 | 1 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 1 | 0 | 0 | 1 |

*а*

| $x_1x_2$ \ $x_3x_4$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 0 | 1 |
| 01 | 1 | 1 | 0 | 1 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 1 | 0 | 0 | 1 |

*б*

Рис. 31. Карта Карно функции четырех переменных

$f_{\text{МДНФ}} = \bar{x}_2\bar{x}_4 \vee \bar{x}_1\bar{x}_4 \vee \bar{x}_1 x_2 \bar{x}_3$ (для рис. 31, *а*);

$f_{\text{МКНФ}} = (x_2 \vee \bar{x}_4)\cdot(\bar{x}_1 \vee \bar{x}_2)\cdot(\bar{x}_3 \vee \bar{x}_4)$ (для рис. 31, *б*)

Особенностью изображения карт Карно для числа переменных более четырех является то, что «математически» соседние столбцы карты Карно пространственно оказываются разнесенными. Таким образом, карта Карно для пяти переменных представляет собой две карты четырех переменных, зеркально отображенные относительно центральной вертикальной линии (рис. 32).

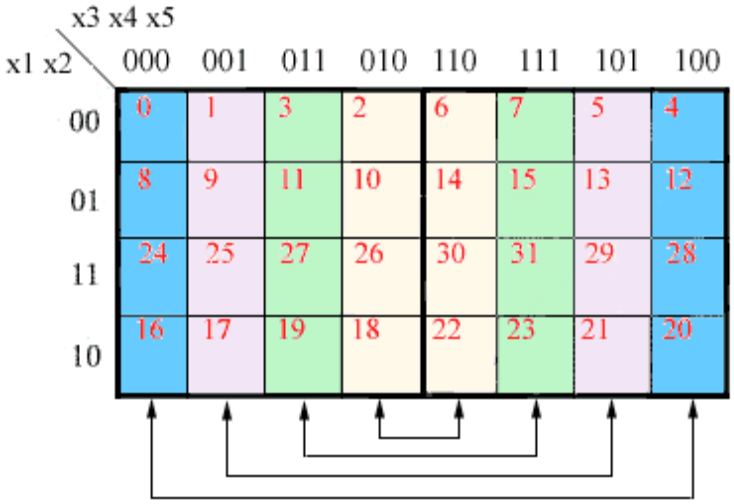



Рис. 32. Карта Карно функции пяти переменных

При этом столбцы одного цвета в правой и левой частях карты фактически оказываются соседними по переменной $x_3$ (соседние столбцы также указываются стрелками в нижней части карты). При выполнении склеиваний следует учитывать «соседство» указанных столбцов, особенно тех из них, которые пространственно разделены.

### 4.9.4. Минимизация конъюнктивных нормальных форм

Минимизация КНФ производится аналогично рассмотренным методам минимизации ДНФ булевых функций, поэтому остановимся лишь на основных положениях.

Напомним, что конституентой нуля называется набор, на котором функция принимает нулевое значение. Она выражается дизъюнкцией всех переменных функции взятых с инверсией. Например, набору 0110 соответствует конституента нуля $x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4$. Необходимо ввести понятия имплиценты и простой имплиценты [12].

*Имплицентой g* булевой функции *f* называется функция, принимающая значение нуль на подмножестве наборов на которых значения функции *f* равны 0 или не определены.

*Простой имплицентой* функции *f* называется элементарная дизъюнкция *p*, являющаяся имплицентой функции *f*, причем никакая ее собственная часть этой дизъюнкции имплицентой функции *f* не является.

Задачей минимизации КНФ является определение минимальной КНФ. Эта задача также решается в два этапа: поиск сокращенной КНФ (конъюнкция всех простых имплицент) и затем нахождение минимальной КНФ. Второй этап минимизации выполняется с помощью таблицы Квайна точно так же, как и при поиске минимальной ДНФ, так как возможны только два варианта: либо данная простая имплицента поглощает данную конституенту нуля, либо нет в соответствии с операцией поглощения:

$(A \vee x)\, A = A$.

Что касается первого этапа − поиска всех простых имплицент, то практически все методы минимизации ДНФ имеют свои аналоги для КНФ. Рассмотрим это подробнее.

Операция склеивания по Квайну:

$(A \vee x)(A \vee \overline{x}) = (A \vee x)(A \vee \overline{x})A$.

Операция склеивания по Блейку:

$(A \vee x)(B \vee \overline{x}) = (A \vee x)(B \vee \overline{x})(A \vee B)$.

Метод Нельсона в применении к задаче минимизации КНФ: раскрытие скобок в произвольной ДНФ функции и выполнение поглощений приводит к сокращенной КНФ. Предполагаются скобки в начале и конце каждого элементарного произведения исходной ДНФ $x_1\overline{x_2} \vee \overline{x_1}x_2$ и использование второго дистрибутивного закона. Например, функция, заданная минимальной ДНФ, дает возможность определить ее сокращенную КНФ:

$(x_1 \vee \overline{x_1})(x_1 \vee x_2)\,(\overline{x_1} \vee \overline{x_2})(x_2 \vee \overline{x_2}) = (x_1 \vee x_2)(\overline{x_1} \vee \overline{x_2})$.

По диаграмме Вейча поиск минимальной КНФ осуществляется аналогично, как и в случае ДНФ. Отличие состоит лишь в том, что в контур объединяются клетки соответствующие наборам переменных на которых функция принимает нулевое значение и переменные выписываются с инверсиями.

Например, для функции, заданной диаграммой (рис. 33), минимальной КНФ является:

$f_{\text{МКНФ}} = (\overline{x_1} \vee x_3)(x_2 \vee x_3)(\overline{x_1} \vee \overline{x_2} \vee \overline{x_4})$.

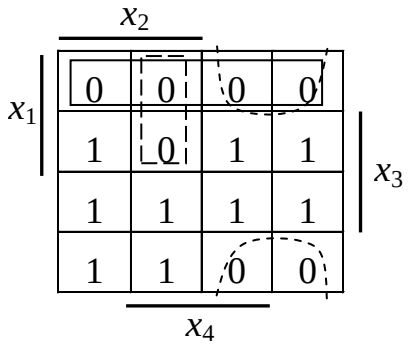



Рис. 33. Карта Вейча для поиска $f_{\text{МКНФ}}$

Для сравнения найдем минимальную ДНФ:

$f_{\text{МДНФ}} = \overline{x_1}x_2 \vee \overline{x_2}x_3 \vee x_3\overline{x_4}$.

В данном случае ДНФ оказалась проще. В общем случае о сравнительной сложности минимальных ДНФ и КНФ нельзя говорить заранее.

### 4.9.5. Минимизация не полностью определенных булевых функций

Если при синтезе логической схемы, реализующей некоторую булеву функцию $n$ переменных, окажется, что некоторые наборы из общего числа $2^n$ никогда не смогут появиться на входах схемы, то данная булева функция не определена на этих наборах. Тогда $2^n$ наборов переменных можно подразделить на три группы: множество наборов $L$, на которых функция принимает единичное значение; множество наборов $D$, на которых функция принимает нулевое значение; множество наборов $N$, на которых функция не определена (неопределенные наборы). Булева функция, содержащая неопределенные наборы, называется не полностью или частично определенной. Неопределенные наборы могут быть использованы для улучшения качества минимизации. При этом неопределенные наборы (при минимизации, например, картами Вейча, Карно) могут участвовать в образовании контуров как с единичными, так и с нулевыми наборами. Это приводит к формированию булевой функции с меньшей ценой по Квайну.

Рассмотрим пример. Пусть задана $f_{\text{СДНФ}} = \overline{x}_1 x_2 \overline{x}_3 \vee x_1 x_2 \overline{x}_3 \vee x_1 \overline{x}_2 x_3$ булевой функции картой Вейча. Звездочками на карте (рис. 34) отмечены наборы, на которых функция $f$ не определена. Если не учитывать неопределенные наборы, то минимальная форма (только на единичных наборах) будет иметь вид: $f_{\text{МДНФ}} = x_2 \overline{x}_3 \vee x_1 \overline{x}_2 x_3$. В случае если неопределенные наборы участвуют в образовании контуров, а следовательно, и $f_{\text{МДНФ}}$, то функция примет следующий вид: $f_{\text{МДНФ}} = x_2 \overline{x}_3 \vee x_1 \overline{x}_2$. Таким образом, реализация полученной $f_{\text{МДНФ}}$ логической схемой будет «дешевле» по сравнению с реализацией функции представленной в форме $f_{\text{СДНФ}}$.

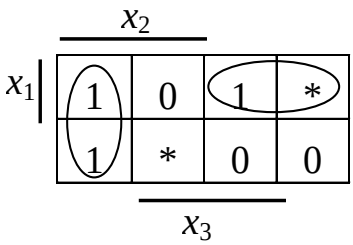



Рис. 34. Карта Вейча

Приведем примеры минимизации не полностью определенных булевых функций с использованием карт Вейча (рис. 35).

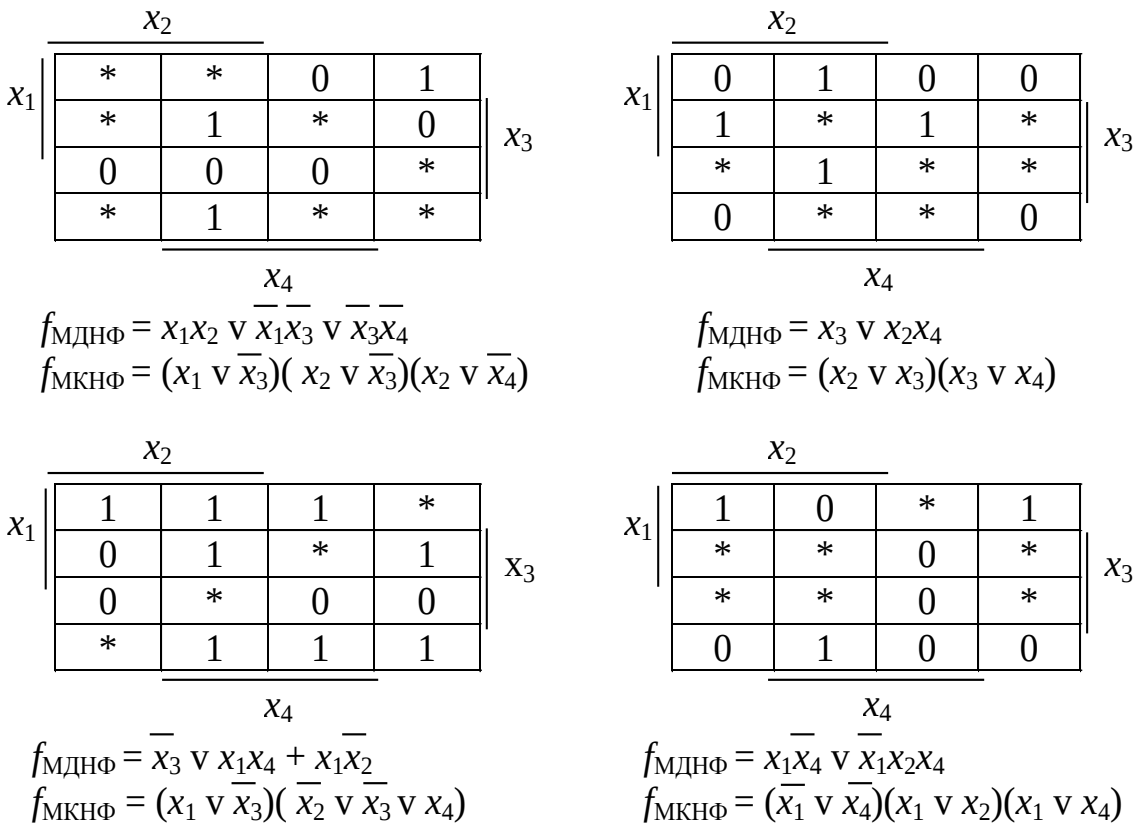



Рис. 35. Карта Вейча не полностью определенной функции

На рис. 36 приведен пример использования карты Карно для минимизации не полностью определенной булевой функции от четырех переменных в форме ДНФ и КНФ.

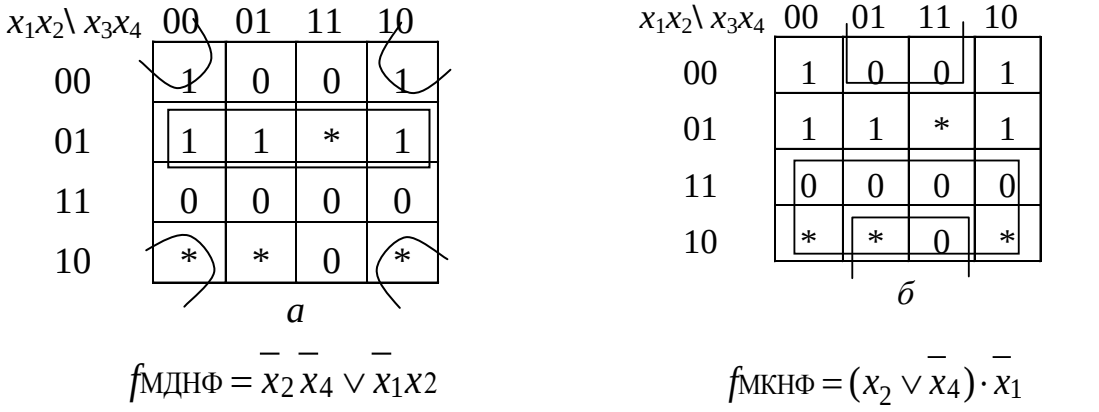



Рис. 36. Карта Карно не полностью определенной функции

Если рассматривать запись результатов минимизации в кубическом виде, то получим множество кубов, образующих единичное покрытие $C^1$ (соответствующее ДНФ), и множество кубов, образующих нулевое покрытие $C^0$ (соответствующее КНФ):

$$C^1 = \left\{ \begin{array}{l} x0x0 \\ 01xx \end{array} \right\}, \quad C^0 = \left\{ \begin{array}{l} x0x1 \\ 1xxx \end{array} \right\}.$$

### 4.9.6. Метод Квайна − Мак-Класки

Этот метод ориентирован на числовое задание булевой функции в виде кубического комплекса, состоящего из 0-кубов. Метод представляет собой формализованный на этапе нахождения простых импликант метод Квайна. Основной недостаток метода Квайна – необходимость выполнения полного сравнения (склеивания) всех минтермов на этапе нахождения простых импликант. В случае большого их количества сложность этого сравнения существенно возрастает.

Рассмотрим этапы метода Квайна − Мак-Класки.

На первом этапе, рассматриваемого метода, все исходные *n*-кубы разбиваются на непересекающиеся подгруппы по количеству единиц в кубе. Пусть, например, задана функция

$f_{\text{СДНФ}}(x_1x_2x_3x_4)$ = V (2, 3, 4, 6, 9, 10, 11, 12).

Сформируем кубический комплекс *K*, состоящий из кубов нулевой размерности:

*K* = (0010, 0011, 0100, 0110, 1001, 1010, 1011, 1100).

Выполним разбиение комплекса K на группы, получим

$$K_1^0 = \begin{Bmatrix} 0010 \\ 0100 \end{Bmatrix}, \qquad K_2^0 = \begin{Bmatrix} 0011 \\ 1100 \\ 0110 \\ 1001 \\ 1010 \end{Bmatrix}, \qquad K_3^0 = \{1011\}.$$

Попарное сравнение можно проводить только между соседними по номеру группами кубов ($K_1^i$ и $K_2^{i+1}$), так как кубы, порождающие новые кубы, должны иметь кодовое расстояние, равное единице. Если сравниваемые кубы различаются только в одном разряде, то в следующий столбец записывается код с символом *x* на месте этого разряда, т. е. производится операция полного склеивания, в результате которой формируется куб следующего ранга (ранг определяется количеством свободных координат *x* в нем). В результате сравнения кубов получим:

$$K_1^0 \text{ и } K_2^0 \Rightarrow K_1^1 = \begin{Bmatrix} 001x \\ 0x10 \\ x010 \\ x100 \\ 01x0 \end{Bmatrix}, \qquad K_2^0 \text{ и } K_3^0 \Rightarrow K_2^1 = \begin{Bmatrix} x011 \\ 10x1 \\ 101x \end{Bmatrix}.$$

После выполнения первого этапа метода простых импликант не выявлено. Полученные кубы первой размерности разобьем на группы кубов в зависимости от местоположения свободной координаты в кубе:

$$K_1^1 = \begin{Bmatrix} x011 \\ x100 \\ x010 \end{Bmatrix}, \qquad K_2^1 = \{0x10\}, \qquad K_3^1 = \begin{Bmatrix} 10x1 \\ 01x0 \end{Bmatrix}, \qquad K_4^1 = \begin{Bmatrix} 001x \\ 101x \end{Bmatrix}.$$

Далее выполняется сравнение кубов внутри каждой из групп. В результате могут быть получены кубы второй размерности, которые, в свою очередь, аналогично кубам первой размерности будут объединены в группы по совпадению свободных координат и вновь будет выполнено сравнение. На каждом шаге сравнения выявляются кубы, не участвовавшие в образовании новых кубов – простые импликанты, и исключаются из дальнейшего упрощения. Для рассматриваемого примера сравнение в группах $K_1^1 \ \ldots \ K_4^1$ привело к образованию двух новых кубов $x01x$ и $x01x$ и кубов, не образовавших новых {$x100$, $0x10$, $10x1$, $01x0$}:

$$K_1^2 = \{x01x\}, \qquad K_2^2 = \{x01x\}.$$

Дальнейшее сравнение не приводит к формированию новых кубов $K_1^2 = K_2^2 = \{x01x\}$. Таким образом, получено множество простых импликант

$f_{\text{сокр.ДНФ}}$ = {$x100$, $0x10$, $10x1$, $01x0$, $x01x$}.

Далее аналогично методу Квайна строится импликантная таблица (табл. 15). Формирование минимального покрытия сводится к выявлению обязательных простых импликант и построению на их основе тупиковых форм.

Из таблицы следует, что простые импликанты $x100$, $10x1$, $x01x$ являются существенными (обязательными), они образуют ядро функции, т. е. будут обязательно входить во все тупиковые ДНФ. Оставшиеся две простые импликанты ($0x10$ и $01x0$) не являются обязательными. Они обе покрывают один непокрытый минтерм 0110.

Таблица 15

| Простые импликанты | | Минтермы | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 0010 | 0100 | 0011 | 1100 | 0110 | 1001 | 1010 | 1011 |
| *A* | $x100$ | | * | | ***** | | | | |
| *B* | $0x10$ | * | | | | * | | | |
| *C* | $10x1$ | | | | | | ***** | | * |
| *D* | $01x0$ | | * | | | * | | | |
| *E* | $x01x$ | * | | ***** | | | | * | * |

Следовательно образуются следующие две тупиковые формы:

$f_{\text{МДНФ}}$ = {$x100$, $10x1$, $x01x$, $0x10$} – 1-я тупиковая форма (*ACEB*);

$f_{\text{МДНФ}}$ = {$x100$, $10x1$, $x01x$, $01x0$} – 2-я тупиковая форма (*ACED*).

Из всех тупиковых форм выбирается та, у которой наименьшее число вхождений переменных (букв).

*Замечание*. Если функция не полностью определена, наборы, на которых она не определена, могут участвовать в склеивании, но в импликантную таблицу не вносятся.

Для получения МКНФ выполняется формирование кубических комплексов для макстермов, формируется таблица имплицент, а тупиковые формы получаются в виде конъюнкции всех существенных имплицент и оставшихся простых имплицент из набора.

Рассмотрим еще один пример.

$f_{\text{СДНФ}}(x_1x_2x_3x_4) = \text{V}\,(0, 1, 2, 3, 5, 7, 8, 10, 11, 12, 13)$.

Как и ранее сформируем кубический комплекс $K$ и выполним разбиение его на группы:

$$K_0^0 = \{0000\}, \qquad K_1^0 = \begin{Bmatrix} 0001 \\ 0010 \\ 1000 \end{Bmatrix}, \qquad K_2^0 = \begin{Bmatrix} 0011 \\ 0101 \\ 1010 \\ 1100 \end{Bmatrix}, \qquad K_3^0 = \begin{Bmatrix} 0111 \\ 1011 \\ 1101 \end{Bmatrix}.$$

В результате попарного сравнения кубов соседних комплексов получим:

$$K_0^0 \text{ и } K_1^0 \Rightarrow K_1^1 = \begin{Bmatrix} 000x \\ 00x0 \\ x000 \end{Bmatrix}, \; K_1^0 \text{ и } K_2^0 \Rightarrow K_2^1 = \begin{Bmatrix} 00x1 \\ 0x01 \\ 001x \\ x010 \\ 10x0 \\ 1x00 \end{Bmatrix}, \; K_2^0 \text{ и } K_3^0 \Rightarrow K_2^1 = \begin{Bmatrix} 0x11 \\ x011 \\ 01x1 \\ x101 \\ 101x \\ 110x \end{Bmatrix}.$$

В результате сравнения кубов на первом шаге простых импликант не выявлено. Полученные кубы разобьем на группы в зависимости от местоположения свободной координаты в кубе:

$$K_1^1 = \begin{Bmatrix} x000 \\ x010 \\ x011 \\ x101 \end{Bmatrix}, \qquad K_2^1 = \begin{Bmatrix} 0x01 \\ 1x00 \\ 0x11 \end{Bmatrix}, \qquad K_3^1 = \begin{Bmatrix} 00x0 \\ 00x1 \\ 10x0 \\ 01x1 \end{Bmatrix}, \qquad K_4^1 = \begin{Bmatrix} 000x \\ 001x \\ 101x \\ 110x \end{Bmatrix}.$$

Далее выполняется сравнение кубов внутри каждой из полученных групп. В результате выполненного сравнения получены кубы второй размерности {00*xx*, *x*0*x*0, 0*xx*1, *x*01*x*} и выделены кубы, не образовавшие новых кубов (простые импликанты) {1*x*00, *x*101, 110*x*}. Из кубов второй размерности сформируем группы:

$$K_1^2 = \{00xx\}, \qquad K_2^2 = \{x0x0\}, \qquad K_3^2 = \{0xx1\}, \qquad K_4^2 = \{x01x\}.$$

Дальнейшее сравнение кубов внутри комплексов не приводит к формированию новых кубов. Таким образом, получено множество простых импликант:

$f_{\text{сокр.ДНФ}}$ = {1*x*00, *x*101, 110*x*, 00*xx*, *x*0*x*0, 0*xx*1, *x*01*x*}.

Как и в предыдущем примере построим импликантную таблицу (табл. 16) из кубов, образующих сокращенную ДНФ.

Таблица 16

| Простые импликанты | | Минтермы | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0000 | 0001 | 0010 | 0011 | 0101 | 0111 | 1000 | 1010 | 1011 | 1100 | 1101 |
| *A* | 1*x*00 | | | | | | | * | | | * | |
| *B* | *x*101 | | | | | * | | | | | | * |
| *C* | 110*x* | | | | | | | | | | * | * |
| *D* | 00*xx* | * | * | * | * | | | | | | | |
| *E* | *x*0*x*0 | * | | * | | | | * | * | | | |
| *F* | 0*xx*1 | | * | | * | * | * | | | | | |
| *G* | *x*01*x* | | | * | * | | | | * | * | | |

Столбцы с минтермами 0111, 1011 имеют по одной метке, следовательно, простые импликанты *F* и *G* являются существенными. Вычеркиваются столбцы, которые имеют метку в строках *F* или *G*.

Записываем формулу (функцию) покрытия оставшихся минтермов простыми импликантами по методу С. Петрика [12].

Функция покрытия строится следующим образом. Соединяются знаком дизъюнкции и заключаются в скобки имена (символы) тех простых импликант, которые соответствуют строкам, содержащим метки в рассматриваемом столбце. Это выполняется для каждого невычеркнутого столбца, затем все скобки соединяются знаком конъюнкции, поскольку рассматриваемые простые импликанты покрывают в своей совокупности оставшиеся минтермы функции. Полученная функция преобразуется путем использования закона поглощения булевой алгебры и записывается в виде ДНФ. Каждый ее терм соответствует неизбыточному набору импликант, покрывающему оставшиеся минтермы функции.

Совокупность импликант ядра и каждого из полученных наборов соответствует одной тупиковой ДНФ функции.

(*D* v *E*) (*A* v *E*) (*A* v *C*) (*B* v *C*) = (*DA* v *DE* v *AE* v *E*) (*AB* v *AC* v *CB* v *C*) =

= (*DA* v *E*) (*AB* v *C*) = *DAB* v *DAC* v *EAB* v *EC*

$f_{\text{МДНФ}}$ = {0*xx*1, *x*01*x*, 00*xx*, 1*x*00, *x*101} – 1-я тупиковая форма (*FGDAB*);

$f_{\text{МДНФ}}$ = {0*xx*1, *x*01*x*, 00*xx*, 10*x*0, 110*x*} – 2-я тупиковая форма (*FGDAC*);

$f_{\text{МДНФ}}$ = {0*xx*1, *x*01*x*, *x*0*x*0, 1*x*00, *x*101} – 3-я тупиковая форма (*FGEAB*);

$f_{\text{МДНФ}}$ = {0*xx*1, *x*01*x*, *x*0*x*0, 110*x*} – 4-я тупиковая форма (*FGEC*) .

4-я тупиковая форма является кратчайшей ДНФ. Она может быть упрощена. Можно вынести за скобки $\overline{x}_2$, тогда получим скобочную форму записи функции

$$y = \overline{x}_1 x_4 \vee \overline{x}_2 (x_3 \vee \overline{x}_4) \vee x_1 x_2 \overline{x}_3.$$

Запись функции в скобочной форме является более короткой, чем МДНФ, и выходит из класса нормальных форм.

### 4.9.7. Алгоритм извлечения (метод Рота)

Метод Рота [3, 14] предназначен для минимизации булевой функции, заданной в форме *произвольного кубического покрытия*, что позволяет упростить процесс подготовки выражения для минимизации. Достоинство алгоритма Рота – полная формализация действий на всех этапах минимизации функции.

**Специальные логические операции алгоритма Рота: *, ∩, #**

Реализация алгоритма извлечения осуществляется на основе специальных логических операций, которые позволяют полностью формализовать процесс получения минимальной формы.

**Операция умножения кубов (*).** Операция умножения кубов $a = a_1a_2...a_n$ и $b = b_1b_2...b_n$ обозначается как $c = a*b$ и служит для образования $r$-куба, противоположные $(r - 1)$ грани которого содержатся в кубах $a$ и $b$. Предварительные координаты куба $c = c_1c_2...c_n$ определяются в соответствии с таблицей, приведенной ниже.

| $a_i$ \ $b_i$ | * | 0 | 1 | $x$ |
|---|---|---|---|---|
| | 0 | 0 | $y$ | 0 |
| | 1 | $y$ | 1 | 1 |
| | $x$ | 0 | 1 | $x$ |

Окончательно координаты куба $c$ формируются:

$$a*b = \begin{cases} m(a_1*b_1)\ m(a_2*b_2)\ ...\ m(a_n*b_n), \text{ если } a_i*b_i = y \text{ только для одного } i, \\ \varnothing, \text{ если } a_i*b_i = y \text{ для } i \geq 2, \end{cases}$$

где $m(a_i*b_i)$ – окончательная $i$-я координата куба $c$; $m(0) = 0$; $m(1) = 1$; $m(y) = x$; $y$ – условное обозначение того, что координаты $a_i$ и $b_i$ противоположны.

Эта операция соответствует операции склеивания: образуется новый $r$-куб, если кодовое расстояние двух исходных кубов равно единице.

Рассмотрим некоторые примеры, графическая интерпретация которых приведена на рис. 37.

1. 011
   *001
   $0y1 \Rightarrow 0x1$ – ребро покрывает две вершины

2. $11x$
   $*01x$
   $y1x \Rightarrow x1x$ – грань

3. $0x1$
   $*1x0$
   $yxy \Rightarrow \varnothing$ – новый куб не порождается (координаты имеют значение $y$).

4. $x1x$
   $*011$
   011

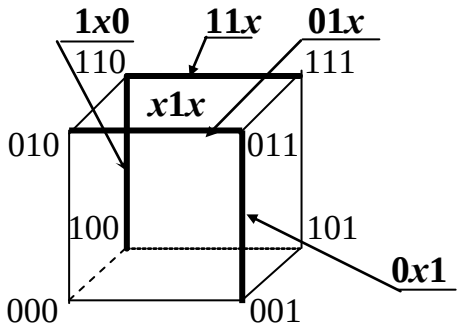



Рис. 37. Графическая интерпретация операции умножения кубов (*)

**Операция пересечения кубов (∩).** Операция пересечения кубов $a = a_1a_2...a_n$ и $b = b_1b_2...b_n$ обозначается как $c = a \cap b$ и служит для выделения куба $c = c_1c_2...c_n$, являющегося общей частью кубов $a$ и $b$. Предварительные координаты куба $c = c_1c_2...c_n$ определяются согласно следующей таблице.

| $a_i$ \ $b_i$ | 0 | 1 | $x$ |
|---|---|---|---|
| 0 | 0 | ∅ | 0 |
| 1 | ∅ | 1 | 1 |
| $x$ | 0 | 1 | $x$ |

Окончательно координаты куба $c$ формируются:

$$a \cap b = \begin{cases} m(a_1 \cap b_1)\, m(a_2 \cap b_2)\ \ldots\ m(a_n \cap b_n), \\ \varnothing, \text{ если существует такое } i, \text{ для которого } a_i \cap b_i = \varnothing, \end{cases}$$

где $m(a_i * b_i)$ – окончательная $i$-я координата куба $c$; $m(0) = 0$; $m(1) = 1$; $m(x) = x$.

Рассмотрим примеры и их графическую интерпретацию (рис. 38).

1. $\begin{array}{l} 100 \\ \cap\ \underline{001} \end{array}$
   $\varnothing 0 \varnothing \Rightarrow \varnothing$ − общей части у кубов нет

2. $\begin{array}{l} 1x0 \\ \cap\ \underline{10x} \end{array}$
   $100 \Rightarrow 100$ − общая часть кубов (ребер) − вершина

3. $\begin{array}{l} x1x \\ \cap\ \underline{0xx} \end{array}$
   $01x \Rightarrow 01x$ − общая часть кубов (граней) – ребро

4. $\begin{array}{l} 0xx \\ \cap\ \underline{0x0} \end{array}$
   $0x0$ (совпадает с операцией умножения кубов (*))

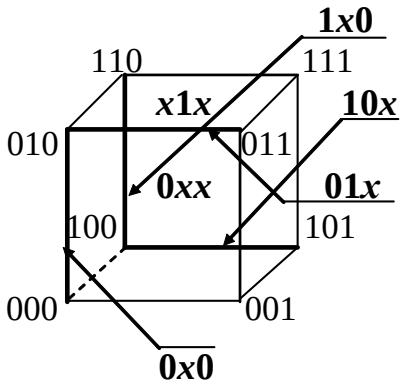



Рис. 38. Графическая интерпретация операции пересечения кубов (∩)

**Операция вычитания кубов (#).** Операция вычитания из куба $a = a_1a_2...a_n$ куба $b = b_1b_2...b_n$ обозначается как $c=a \# b$ и служит для удаления из куба $a$ общей части кубов $a$ и $b$. Предварительные координаты куба $c = c_1c_2...c_n$ определяются согласно следующей таблице.

| $a_i$ \ $b_i$ | 0 | 1 | x |
|---|---|---|---|
| 0 | z | y | z |
| 1 | y | z | z |
| x | 1 | 0 | z |

Окончательно координаты куба $c$ формируются:

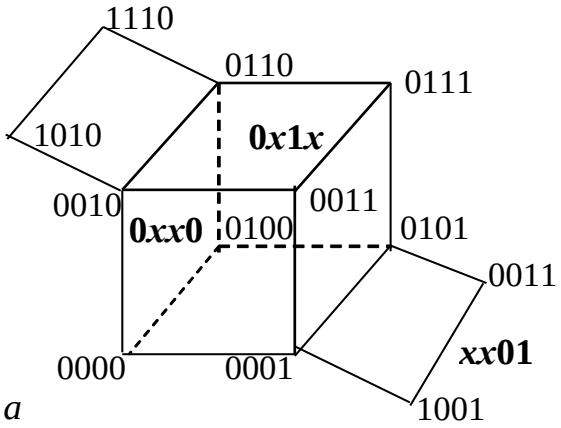
$$c = a \# b = \begin{cases} \varnothing, \text{ если для любого } i \ \ a_i \# b_i = z, \\ a, \text{ если существует } i, \text{ такое что } a_i \# b_i = y, \\ \bigcup_{i=1} a_1a_2...a_{i-1}\,\alpha_i\,a_{i+1}...a_n, \text{ если } \alpha_i \text{ равно 0 или 1 для одного или нескольких } i, \end{cases}$$

где $z$ означает, что координаты совпадают, а $y$, как для операции *, означает, что координаты $a_i$ и $b_i$ противоположны.

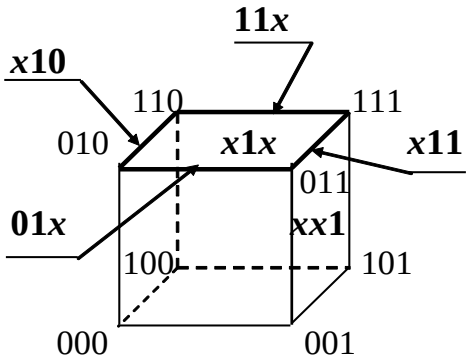
По этим $\alpha_i$-координатам производится объединение (U) кубов, получаемых в результате замены в кубе $a$ символа $x$ на соответствующее значение (0,1) координаты $\alpha_i$. Рассмотрим примеры выполнения операции # и их графическую интерпретацию (рис. 39).

1. $x1x \ \#\ x11$
   $zz0 \Rightarrow x10$ – ребро
2. $x1x \ \#\ 110$
   $0z1 \Rightarrow \begin{Bmatrix} 01x \\ x11 \end{Bmatrix}$ – два ребра
3. $xx1 \ \#\ x10$
   $z0y \Rightarrow xx1$ – грань
4. $x11 \ \#\ xx1$
   $zzz \Rightarrow \varnothing$ – пусто

5. $\begin{matrix} 0xxx \\ \#\ \underline{xx01} \\ zz10 \end{matrix} \Rightarrow \left\{ \begin{matrix} 0x1x \\ 0xx0 \end{matrix} \right\}$ – две грани



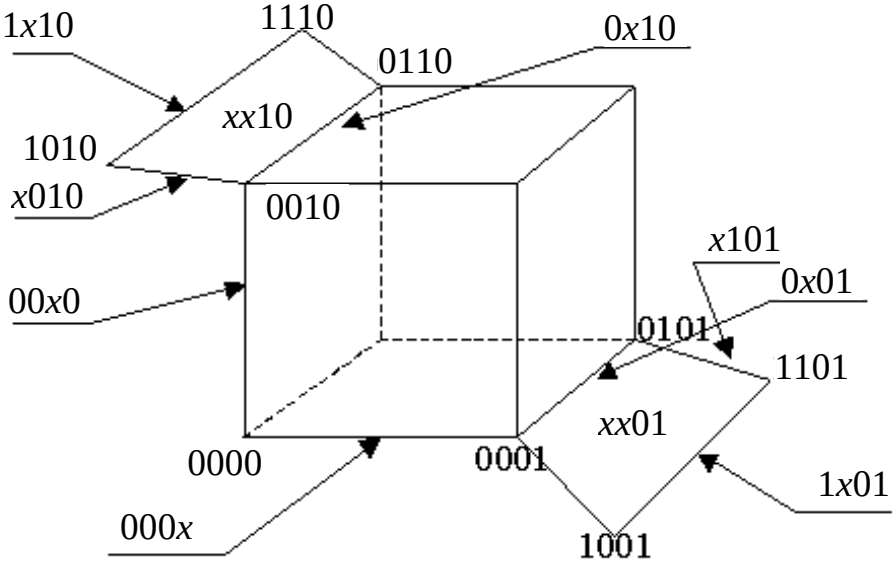

*а*




*б*

Рис. 39. Графическая интерпретация операции вычитания кубов (#)

Далее рассмотрим метод Рота на примере минимизации булевой функции, заданной покрытием $C_0$ (рис. 40).




$$C_0 = \left\{ \begin{matrix} L = \left\{ \begin{matrix} x010 \\ 0x10 \\ 0000 \\ 0x01 \end{matrix} \right. \\ N = \left\{ \begin{matrix} 1110 \\ 1x01 \end{matrix} \right. \end{matrix} \right.$$

Рис. 40. Графическое задание исходного покрытия

Исходное покрытие $C_0$ задано объединением множеств кубов $L$ и $N$. Кубы комплекса $N$ − это наборы, на которых функция не определена и которые включаются в покрытие ради возможного дополнительного упрощения комплекса $L$. Минимальное покрытие $C_{\min}$ комплексов $L$ и $N$, называется $K$-покрытием $L$.

Общий метод построения минимального $K$-покрытия называется алгоритмом извлечения и состоит в следующем:

- нахождении множества $Z$ простых импликант комплекса $K$;
- выделении $L$-экстремалей на множестве $Z$;
- применении алгоритма ветвления при отсутствии $L$-экстремалей;
- нахождении абсолютно минимального покрытия из некоторого множества избыточных покрытий.

**1. Нахождение множества простых импликант**

Преобразование исходного покрытия $C_0$ комплекса $K$ в множество простых импликант $Z$ осуществляется с помощью операции умножения кубов. В результате первого шага ($C_0$*$C_0$) (табл. 17) предусматривается выявление как новых кубов $C_y$ (первой и более высокой размерности), так и кубов, которые не образуют новых кубов (включаются в множество $Z_0$).

Таблица 17

| $C_0$*$C_0$ | $x$010 | 0$x$10 | 0000 | 0$x$01 | 1110 | 1$x$01 |
|---|---|---|---|---|---|---|
| $x$010 | – | | | | | |
| 0$x$10 | 0010 | – | | | | |
| 0000 | 00$y$0 | 00$y$0 | – | | | |
| 0$x$01 | Ø | Ø | 000$y$ | – | | |
| 1110 | 1$y$10 | $y$110 | Ø | Ø | – | |
| 1$x$01 | Ø | Ø | Ø | $yx$01 | Ø | – |
| $A_1$ | 00$x$0<br>1$x$10 | $x$110 | 000$x$ | $xx$01 | | |

Из полученных новых кубов образуется множество $A_1$. Также формируется множество $B_1 = C_0 - Z_0$. Для следующего шага получения множества $Z$ формируется множество $C_1 = A_1 \cup B_1$. Для уменьшения мощности множества кубов $C_1$ выполним операцию поглощения (удаления) кубов, образующих множество $C_1$, кубами из множества $A_1$ ($A_1 \subseteq C_1$). Для рассматриваемого примера получим:

$$A_1 = \begin{Bmatrix} 00x0 \\ 1x10 \\ x110 \\ 000x \\ xx01 \end{Bmatrix}, \quad Z_0 = \text{Ø}.$$

Далее сформируем новое множество кубов $C_1 = A_1 \cup B_1$.

$$C_1 = \left\{\begin{matrix} 00x0 \\ 1x10 \\ 000x \\ x110 \\ xx01 \\ x010 \\ 0x10 \\ 0000 \\ 0x01 \\ 1110 \\ 1x01 \end{matrix}\right\}$$ (до выполнения операции поглощения кубами множества $A$ кубов множества $B$)

$$C_1 = \left\{\begin{matrix} 00x0 \\ 1x10 \\ 000x \\ x110 \\ xx01 \\ x010 \\ 0x10 \end{matrix}\right\}$$ (после выполнения операции поглощения)

$$C_1 * C_1 = (A_1 \cup B_1) * (A_1 \cup B_1) = (A_1 * A_1) \cup (A_1 * B_1) \cup (B_1 * A_1) \cup (B_1 * B_1) = (A_1 * A_1) \cup (A_1 * B_1).$$

Среди кубов $C_0$, возможно, находятся такие кубы, которые с кубами множества $A_1$ могут дать новые кубы или оказаться простыми импликантами после второго шага ($C_1 * C_1$). При формировании таблицы для выполнения операции $C_1 * C_1$ (табл. 18) следует учесть, что $B_1 * B_1$ уже выполнялось на шаге $C_0 * C_0$. Следовательно:

Таблица 18

| $C_1 * C_1$ | $00x0$ | $1x10$ | $000x$ | $x110$ | $xx01$ |
|---|---|---|---|---|---|
| $00x0$ | – | | | | |
| $1x10$ | $y010$ | – | | | |
| $000x$ | 0000 | Ø | – | | |
| $x110$ | $0y10$ | 1110 | Ø | – | |
| $xx01$ | $000y$ | Ø | 0001 | Ø | – |
| $x010$ | 0010 | 1010 | $00y0$ | $xy10$ | Ø |
| $0x10$ | 0010 | $yx10$ | $00y0$ | 0110 | Ø |
| $A_2$ | Ø | $xx10$ | Ø | $xx10$ | Ø |

В результате выполнения умножения $C_1 * C_1$ получим:

$A_2 = \{xx10\}$,

$$Z_1 = \left\{\begin{matrix} 00x0 \\ 000x \end{matrix}\right\}.$$

Необходимо отметить, что куб $xx01$ не дал нового куба. Но это куб второй размерности, поэтому новые кубы он может дать на третьем шаге ($C_2 * C_2$) (табл. 19). Таким образом его не следует включать в число кубов, образующих множество $Z_1$.

$$B_2 = \begin{Bmatrix} 1x10 \\ x110 \\ xx01 \\ x010 \\ 0x10 \end{Bmatrix}, \quad C_2 = A_2 \cup B_2 = \begin{Bmatrix} xx10 \\ 1x10 \\ x110 \\ x010 \\ 0x10 \\ xx01 \end{Bmatrix} = \begin{Bmatrix} xx10 \\ xx01 \end{Bmatrix}.$$

Таблица 19

| $C_2 * C_2$ | $xx10$ |
|---|---|
| $xx10$ | – |
| $xx01$ | Ø |
| $A_3$ | Ø |

Из таблицы следует, что $A_3 = \text{Ø}$. Таким образом, новых кубов при выполнении операции $C_2 * C_2$ не было получено.

$$Z_2 = \begin{Bmatrix} xx10 \\ xx01 \end{Bmatrix}, \quad B_3 = C_2 \setminus Z_2 = \text{Ø}, \quad C_3 = A_3 \cup B_3 = \text{Ø}.$$

На этом процесс выявления простых импликант окончен. Таким образом сформировано множество простых импликант

$$Z = \bigcup_{i=0}^{2} Z_i = Z_0 \cup Z_1 \cup Z_2 = \begin{Bmatrix} 00x0 \\ 000x \\ xx01 \\ xx10 \end{Bmatrix}.$$

Необходимо выяснить, не содержатся ли в этом множестве «лишние» простые импликанты.

**2. Определение *L*-экстремалей**

Множество $Z$ может быть избыточным. Прежде всего необходимо выявить обязательные простые импликанты, называемые в алгоритме извлечения $L$-экстремалями. $L$-экстремаль – это куб, который (и только он) покрывает некоторую вершину из множества $L$, не покрываемую никаким другим кубом из множества $Z$.

Для определения $L$-экстремалей воспользуемся операциями вычитания (#) (табл. 20) и пересечения ($\cap$) кубов (табл. 21). В табл. 20 $z \subseteq Z$ – некоторая простая импликанта, из которой вычитаются остальные $Z - z$.

Таблица 20

| z#(Z − z) | 00*x*0 | 000*x* | *xx*01 | *xx*10 |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 00*x*0 | – | *zzz*1<br>0001 | 11*zy*<br>*xx*01 | 11*zz*<br>1*x*10<br>*x*110 |

Окончание таблицы 20

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 000*x* | *zz*1*z*<br>0010 | – | 11*zz*<br>1*x*01<br>*x*101 | *y*1*yz*<br>1*x*10<br>1*yyz*<br>*x*110 |
| *xx*01 | *zzyy*<br>0010 | *zzzz*<br>Ø | – | *zzyy*<br>1*x*10<br>*zzyy*<br>*x*110 |
| *xx*10 | *zzzz*<br>Ø | Ø | *zzyy*<br>1*x*01<br>*zzyy*<br>*x*101 | – |
| Остаток | Ø | Ø | 1*x*01, *x*101 | 1*x*10, *x*110 |

Таким образом, из таблицы получено множество *L*-экстремалей:

$$E = \begin{Bmatrix} xx01 \\ xx10 \end{Bmatrix}.$$

Если результат вычисления Ø хотя бы в одном, любом случае, то это значит, что среди простых импликант есть кубы, которые покрывают *уменьшаемый*, а следовательно, этот уменьшаемый не может быть *L*-экстремалью.

Если же полученный результат не Ø, то в противоположность предыдущему утверждению *уменьшаемый* куб оказывается кубом большей размерности по отношению к другим простым импликантам.

Что касается простых импликант, «удаленных» от *уменьшаемой*, то они с ней дают координаты *y* и, таким образом, остается *уменьшаемый* куб при вычитании этих «*удаленных»* кубов.

После выявления *L*-экстремалей следует выяснить, не являются ли некоторые из них простыми импликантами, остатки которых покрывают только некоторое подмножество кубов комплекса *N*, которое нет необходимости покрывать, вводя в минимальное покрытие соответствующие наборы. Для этого необходимо выполнить операцию пересечения остатков, полученных при выполнении операции $z\#(Z - z)$ с кубами из комплекса *L*. Во множестве *E* необходимо оставить только те кубы, остатки от которых пересекаются с кубами из комплекса *L*.

Таблица 21

| $z\#(Z-z)\cap L$ | 1*x*01 | *x*101 | 1*x*10 | *x*110 |
|---|---|---|---|---|
| *x*010 | Ø | Ø | 1010 | Ø |
| 0*x*10 | Ø | Ø | 1010 | Ø |
| 0000 | Ø | Ø | Ø | 0110 |
| 0*x*01 | Ø | 1101 | Ø | Ø |

Из таблицы видно, что куб 1*x*01 не пересекается с кубами комплекса *L*. Однако куб *x*101 имеет с кубом 0*x*01 (из комплекса *L*) общую вершину

0101. Оба куба (1*x*01, *x*101) входят в куб более высокой размерности *xx*01 (*L*-экстремаль). Таким образом, куб 1*x*01, образованный на комплексе *N*, позволил уменьшить цену схемы. Выясним далее, какие из вершин комплекса *L* не покрываются *L*-экстремалями. Для этого из каждого куба комплекса *L* вычтем (#) элементы множества *E* (табл. 22). В результате вычитания получим $L_1 = L\#E$.

Таблица 22

| *L*#*E* | *x*010 | 0*x*10 | 0000 | 0*x*01 |
|---|---|---|---|---|
| *xx*01 | *zzyy*<br>*x*010 | *zzyy*<br>0*x*10 | *zzzy*<br>0000 | *zzzz*<br>Ø |
| *xx*10 | *zzzz*<br>Ø | *zzzz*<br>Ø | *zzyz*<br>0000 | Ø |
| | Ø | Ø | 0000 | Ø |

Из таблицы видно, что $L_1 = \{0000\}$. Однако не покрытые *L*-экстремалями кубы должны быть покрыты другими импликантами из множества

$$\hat{Z} = Z \setminus E = \begin{Bmatrix} 00x0 \\ 000x \\ xx01 \\ xx10 \end{Bmatrix} \setminus \begin{Bmatrix} xx01 \\ xx10 \end{Bmatrix} = \begin{Bmatrix} 00x0 \\ 000x \end{Bmatrix}.$$

Теперь из полученного множества $\hat{Z}$ надо выбрать минимальное число кубов с минимальной ценой (максимальной размерностью), чтобы покрыть непокрытые *L*-экстремалями элементы комплекса *L*. Выбор так называемого не максимального куба осуществляется с помощью операции частичного упорядочивания кубов (табл. 23). Введем обозначения кубов *a* и *b* (кубов может быть и больше)

$$\hat{Z} = \begin{Bmatrix} 00x0 \\ 000x \end{Bmatrix} \begin{matrix} - \text{ куб } a, \\ - \text{ куб } b. \end{matrix}$$

Куб *a* будет не максимален по отношению к кубу *b*, если выполняются одновременно два условия:

1) $C^a \geq C^b$, где $C^a$ – цена куба *a*;

2) $a \cap L_1 \subseteq b \cap L_1$, куб *b* покрывает не меньше кубов чем куб *a*.

Таблица 23

| | $\hat{Z} \cap L_1$ | 0000 |
|---|---|---|
| *a* | 00*x*0 | 0000 |
| *b* | 000*x* | 0000 |

В результате получаем, что $C^a = C^b$ (пересечение обоих кубов дало в результате 0-куб). Следовательно, кубы *a* и *b* равноценны и для покрытия вершины 0000 можно выбрать любой из них в качестве экстремали второго порядка:

$E_2^{'} = \{000x\}$ или $E_2^{''} = \{00x0\}$.

Следовательно, могут быть получены две тупиковые формы.

$$f^1_{\text{МДНФ}} = C_{\min} = E \bigcup E_2^{'} = \left\{ \begin{matrix} xx10 \\ xx01 \\ 000x \end{matrix} \right\}$$ – первая тупиковая форма,

$$f^2_{\text{МДНФ}} = C_{\min} = E \bigcup E_2^{''} = \left\{ \begin{matrix} xx10 \\ xx01 \\ 00x0 \end{matrix} \right\}$$ – вторая тупиковая форма.

Аналитически полученные тупиковые формы могут быть записаны следующим образом

$f^1_{\text{МДНФ}} = x_3\overline{x}_4 \vee \overline{x}_3 x_4 \vee \overline{x}_1\overline{x}_2\overline{x}_3$,

$f^2_{\text{МДНФ}} = x_3\overline{x}_4 \vee \overline{x}_3 x_4 \vee \overline{x}_1\overline{x}_2\overline{x}_4$.

Схемная реализация первой тупиковой формы булевой функции приведена на рис. 41.

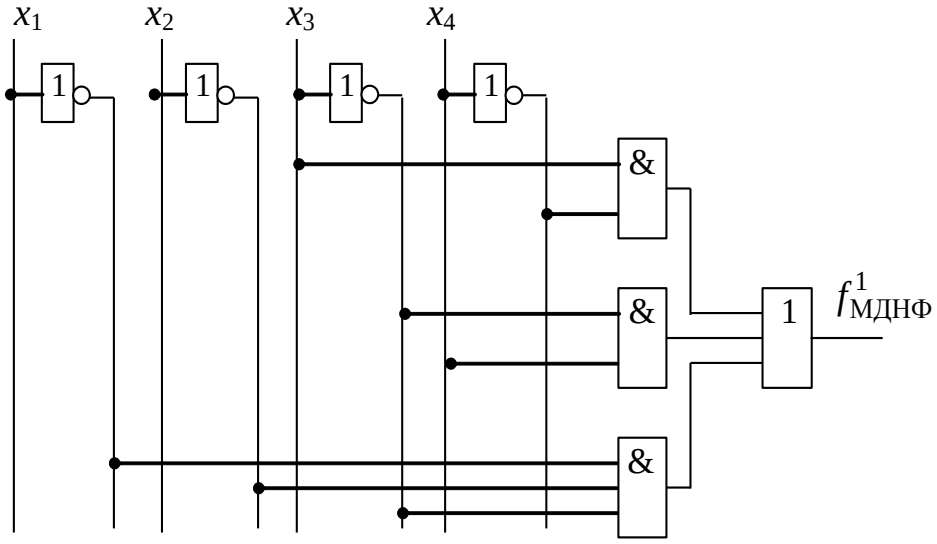



Рис. 41. Схемная реализация $f^1_{\text{МДНФ}}$.

### 4.9.8. Минимизация булевых функций методом преобразования логических выражений

Рассмотрим подход к упрощению булевых функций, заключающийся в применении к ней скобочных преобразований. Пусть имеется функция

$f = x_1x_3x_4x_6 \vee x_2x_3x_4x_6 \vee x_5x_6 \vee x_7$.

Применим к ней скобочные преобразования, в результате чего получим функцию $f = ((x_1 \vee x_2)\, x_3x_4 \vee x_5)\, x_6 \vee x_7$.

Из выражений видно, что цена схемы до минимизации была равна 14, после стала равна 11. Таким образом, общая стоимость схемы сократилась, однако исходной функции соответствовала схема, имеющая два уровня элементов, а преобразованной − пять уровней. Следовательно полученная схема будет работать примерно в 2,5 раза медленнее исходной. Таким образом, уменьшение затрат оборудования в общем случае приводит к понижению быстродействия схем.

### 4.9.9. Минимизация систем булевых функций

Существует два подхода в минимизации систем булевых функций:

- минимизация каждой функции в отдельности;
- совместная минимизация функций системы.

Кратко остановимся на первом. Если произвести минимизацию булевых функций, входящих в систему, независимо друг от друга, то общая схема будет состоять, как правило, из изолированных подсхем. Ее можно иногда упростить за счет объединения участков подсхем, реализующих одинаковые термы, входящие в несколько булевых функций системы.

Пусть в результате минимизации функций получены следующие МДНФ:

$f_1 = x_1\overline{x}_3 \vee x_1\overline{x}_2 \vee \overline{x}_1 x_3,$

$f_1 = x_1\overline{x}_2 \vee \overline{x}_1 x_3 \vee \overline{x}_1 x_2.$

На рис. 42 показана реализация системы функций без учета общих частей (термов). Аппаратные затраты по критерию Квайна без учета инверсий для данной реализации составляют $C$=18.

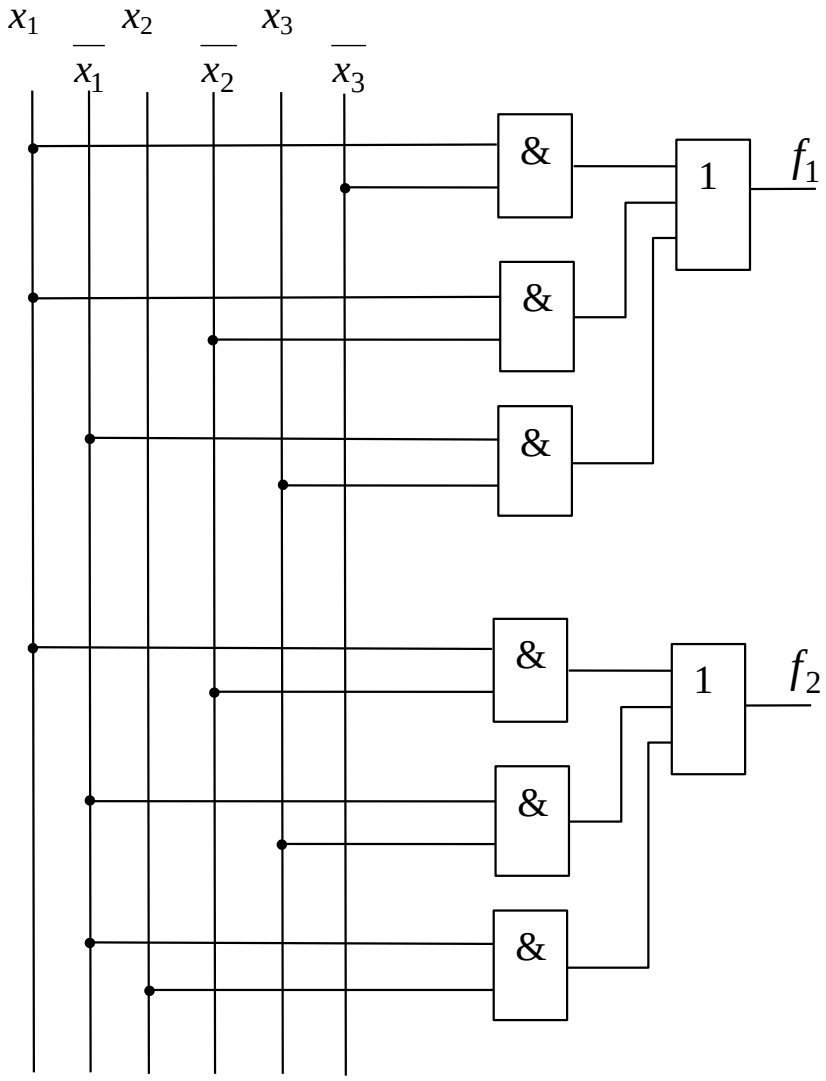



Рис. 42. Реализация системы функций без учета общих частей

На рис. 43 показана реализация системы функций с объединением общих частей $x_1\overline{x}_2$, $\overline{x}_1x_3$. Стоимость схемы в этом случае составляет $C = 14$.

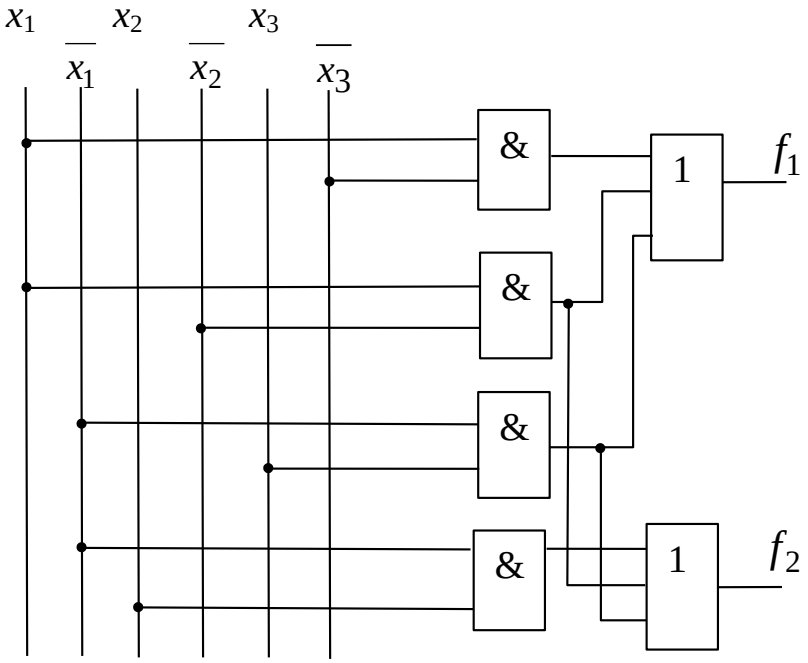



Рис. 43. Реализация системы функций с объединением общих частей

### 4.10. Применение правил и законов алгебры логики к синтезу некоторых цифровых устройств

Остановимся несколько более подробно на применении правил и законов булевой алгебры на примере синтеза некоторых цифровых устройств, являющихся узлами вычислительной техники. К числу таких относятся сумматоры и вычитатели.

Сумматор – узел вычислительного устройства, предназначенный для выполнения арифметической операции сложения двух или нескольких двоичных чисел. При арифметическом сложении выполняются и другие дополнительные операции: учет знаков чисел, выравнивание порядков слагаемых и тому подобное. Указанные операции выполняются в арифметическо-логических устройствах (АЛУ) или процессорных элементах, ядром которых являются сумматоры.

#### 4.10.1. Синтез одноразрядного комбинационного полусумматора

Полусумматор (рис. 44) имеет два входа *x* и *y* для разрядов двух слагаемых и два выхода: С – сумма, П – перенос. Таким образом, это устройство предназначено для сложения разрядов двух чисел без учета переноса из предыдущего разряда. Обозначением полусумматора служат буквы *HS* (*half sum* — полусумма).

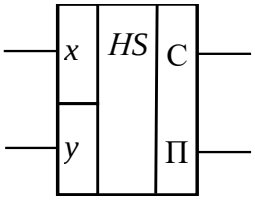



Рис. 44. УГО полусумматора

Таблица истинности, отражающая алгоритм работы полусумматора, имеет следующий вид (табл. 24).

Таблица 24

| $x$ | $y$ | С | П |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Из таблицы следует что функции П и С имеют вид:

$$П = x\,y,$$

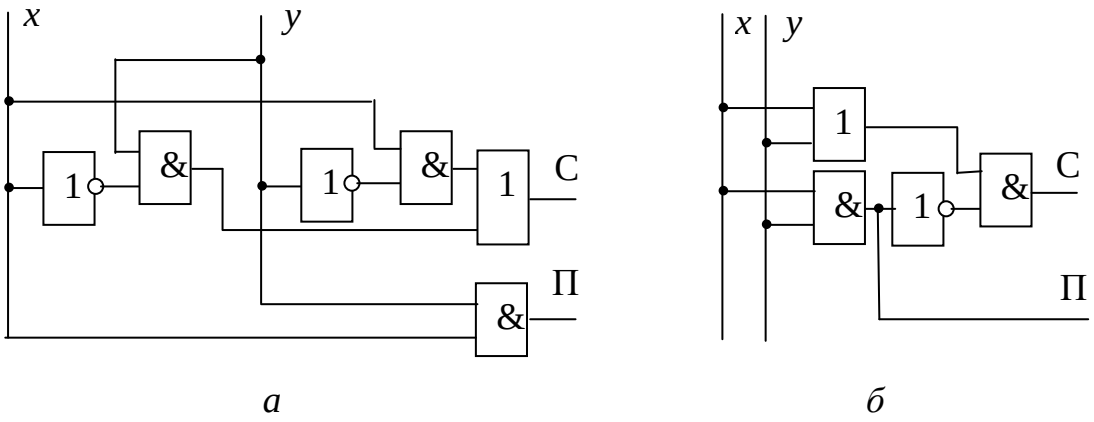
$$С = \bar{x}\,y \vee x\,\bar{y}.$$

Логическая схема, соответствующая записанной системе булевых функций представлена на рис.45.

Данная схема может быть упрощена, если функция С будет записана на нулевых наборах и в ее записи использована функция П. Запись одной функции с участием другой назовем *совместной минимизацей*.

$$\bar{С} = \bar{x}\,\bar{y} \vee x\,y$$

$$\bar{\bar{С}} = С = \overline{\bar{x}\,\bar{y} \vee x\,y} = (x \vee y)\overline{x\,y} = (x \vee y) \cdot \bar{П}$$




*а* *б*

Рис. 45 Логические схемы полусумматора:
*а* – до упрощения; *б* – после упрощения

### 4.10.2. Синтез одноразрядного полного комбинационного сумматора

Полный одноразрядный двоичный сумматор (рис. 46) имеет три входа (*x*, *y* − для разрядов двух слагаемых и *z* − для переноса из предыдущего (младшего) разряда) и два выхода (С − сумма, П − перенос в следующий (старший) разряд). Обозначением полного двоичного сумматора служат буквы *SM*.

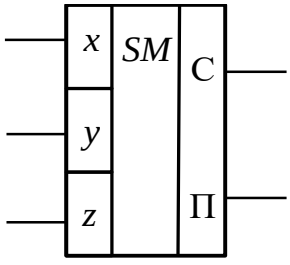



Рис. 46. УГО сумматора

Пусть имеется два числа:

$A = a_1 a_2 \ldots a_{i-1} a_i a_{i+1} \ldots a_n$,

$B = b_1 b_2 \ldots b_{i-1} b_i b_{i+1} \ldots b_n$.

В зависимости от значений переменных $a_i$, $b_i$, $z_i$ (перенос в $i$-й разряд) формируется значение булевых функций $C_i$, и $П_i$. Введем следующие обозначения:

$a_i \Rightarrow x$ $\quad$ $C_i \Rightarrow C$, где $C_i$ – значение суммы в разряде $i$;

$b_i \Rightarrow y$ $\quad$ $П_i \Rightarrow П$, где $П_i$ – значение переноса из разряда $i$.

$z_i \Rightarrow z$

Таблица истинности, отражающая алгоритм работы сумматора, имеет следующий вид (табл. 25).

Таблица 25

| $x$ | $y$ | $z$ | C | П |
|---|---|---|---|---|
| 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 |

Выполним преобразование функций П и С:

$$П = \bar{x}yz \vee x\bar{y}z \vee xy\bar{z} \vee xyz =$$

$$= \bar{x}yz \vee x\bar{y}z \vee xy\bar{z} \vee xyz \vee xyz \vee xyz =$$

$$= yz \cdot (\bar{x} \vee x) \vee xz(\bar{y} \vee y) \vee xy(\bar{z} \vee z).$$

$$П = yz \vee xz \vee xy.$$

$$C = \bar{x}\bar{y}z \vee \bar{x}y\bar{z} \vee x\bar{y}\bar{z} \vee xyz \vee$$

$$\vee \bar{x}\bar{y}x \vee \bar{x}x\bar{z} \vee y\bar{y}\bar{z} \quad \Leftarrow \text{ Логические нули}$$

$$\vee \bar{x}\bar{y}y \vee \bar{x}z\bar{z} \vee z\bar{y}\bar{z} =$$

$$= (x \vee y \vee z)(\bar{x}\bar{y} \vee \bar{x}\bar{z} \vee \bar{y}\bar{z}) \vee xyz,$$

но $\bar{П} = \overline{yz \vee xz \vee xy} = (\bar{x} \vee \bar{y})(\bar{x} \vee \bar{z})(\bar{y} \vee \bar{z}) =$

$$= \bar{x}\bar{y} \vee \bar{x}\bar{z} \vee \bar{y}\bar{z}.$$

Следовательно, с учетом результата преобразования функции П функция С будет иметь вид:

$$C = (x \vee y \vee z) \cdot \bar{П} \vee xyz.$$

Таким образом, логическая схема синтезированного одноразрядного полного комбинационного сумматора имеет следующий вид (рис. 47).

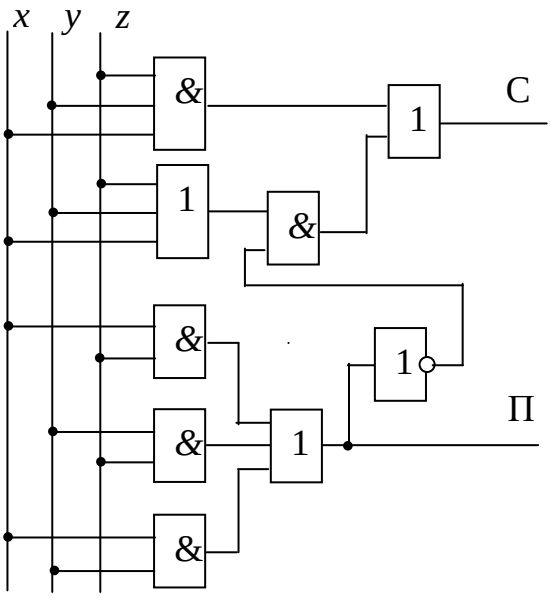



Рис. 47 Логическая схема полного комбинационного сумматора

### 4.10.3. Синтез одноразрядного полного комбинационного сумматора на двух полусумматорах

Согласно рассмотренному выше материалу функция суммирования для полного комбинационного сумматора имеет вид

$$C = \bar{x}\,\bar{y}z \vee \bar{x}\,y\,\bar{z} \vee x\,\bar{y}\,\bar{z} \vee x\,y\,z = \overline{(x\,y \vee x\,\bar{y})}\,\bar{z} \vee (\bar{x}\,\bar{y} \vee x\,y)\,z\,.$$

Введем обозначение $M = x \oplus y$, тогда

$$C = M\,\bar{z} + \overline{M}\,z = M \oplus z = x \oplus y \oplus z\,.$$

Аналогично можно выполнить преобразование функции переноса П:

$$П = \bar{x}\,yz \vee x\,\bar{y}\,z \vee x\,y\,\bar{z} \vee x\,y\,z = (\bar{x}\,y \vee x\,\bar{y})\,z \vee x\,y(\bar{z} \vee z) = (x \oplus y)z \vee xy\,.$$

Схемная реализация полного сумматора на двух полусумматорах и элементе логического ИЛИ будет иметь следующий вид (рис. 48).

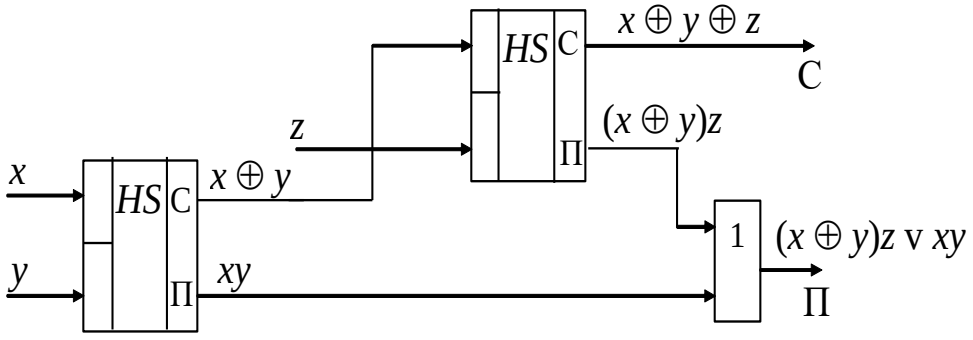



Рис. 48. Схема сумматора на двух полусумматорах и ИЛИ

### 4.10.4. Синтез одноразрядного комбинационного вычитателя

Пусть имеется два числа:

$A = a_1 a_2 \ldots a_{i-1} a_i a_{i+1} \ldots a_n$,

$B = b_1 b_2 \ldots b_{i-1} b_i b_{i+1} \ldots b_n$.

В зависимости от значений переменных $a_i$, $b_i$ и $z_i$ ($z_i$ – заем из $i$-го разряда в $i$ + 1-й разряд) формируется значение булевых функций $\text{Р}_i$ и $\text{З}_i$. Введем следующие обозначения:

$a_i \Rightarrow x$ $\quad$ $\text{Р}_i \Rightarrow$ Р, где $\text{Р}_i$ – значение разности в разряде $i$,

$b_i \Rightarrow y$ $\quad$ $\text{З}_i \Rightarrow$ З, где $\text{З}_i$ – значение заема в $i$-й разряд из $i$ – 1-го.

$z_i \Rightarrow z$

Таблица истинности, соответствующая устройству, выполняющему операцию вычитания, имеет следующий вид (табл. 26).

Таблица 26

| $x$ | $y$ | $z$ | Р | З |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

$\text{З} = \bar{x}\,\bar{y}z \vee \bar{x}\,y\,\bar{z} \vee \bar{x}\,y\,z \vee x\,y\,z$.

Выполним склеивания 1 и 3, 2 и 3, 3 и 4 наборов:

$\text{З} = \bar{x}\,z \vee \bar{x}\,y \vee y\,z = \bar{x}(z \vee y) \vee y\,z$.

$\text{П} = yz \vee x\,z \vee x\,y = x\,(y \vee z) \vee yz$.

$\text{Р} = \bar{x}\,\bar{y}z \vee \bar{x}\,y\,\bar{z} \vee x\,\bar{y}\bar{z} \vee x\,y\,z = \text{С}$.

Как видно, функции разности Р и суммы С совпадают (функция С была получена ранее).

Схемная реализация вычитателя может быть выполна самостоятельно.

### 4.10.5. Объединенная схема одноразрядного комбинационного сумматора-вычитателя

Для системы булевых функций С, Р, П и З, полученных выше, может быть построена объединенная логическая схема (рис. 49) на элементах И, ИЛИ и НЕ, выполняющая обе операции (сложения и вычитания).

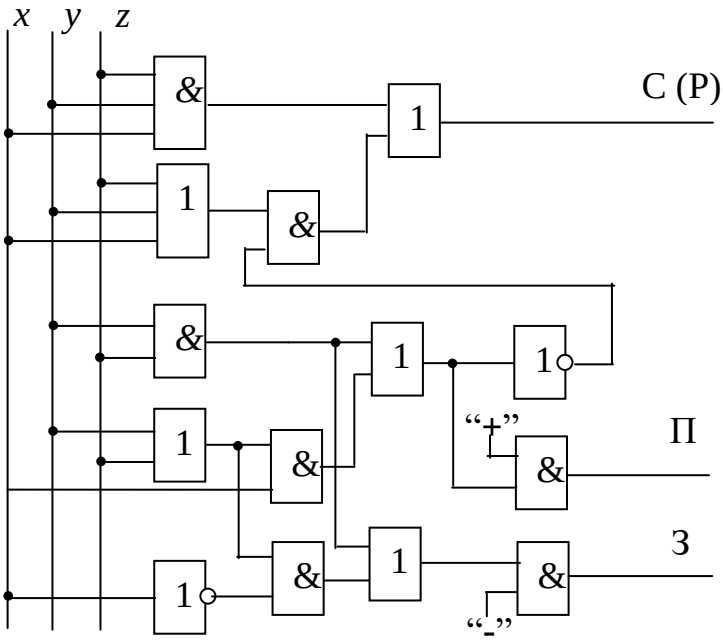



Рис. 49 Логическая схема полного комбинационного сумматора-вычитателя

### 4.10.6. Триггер со счетным входом как полный одноразрядный сумматор

Триггером называется устройство, имеющее два устойчивых состояния и способное под действием входного сигнала переходить из одного устойчивого состояния в другое. Триггер – это простейший цифровой автомат с памятью, способный хранить один бит информации.

Триггер со счетным входом (*T*-триггер) может быть рассмотрен как полный сумматор, работающий в три такта. В основе работы этого устройства лежит операция «сумма по модулю 2».

**Первый такт**

1. $\tau(t-1) = 0$ – триггер находится в исходном состоянии.
2. $T = x$ первое слагаемое подается на вход триггера

$$\tau(t) = \overline{T}\,\tau(t-1) \vee T\,\overline{\tau}(t-1) = \overline{x}\cdot 0 \vee x\cdot 1 = x.$$

После первого такта содержимое триггера соответствует входному сигналу.

**Второй такт**. Во втором такте на вход триггера подается второе слагаемое *y*:

$$\tau(t+1) = \overline{T}\,\tau(t) \vee T\,\overline{\tau}(t) = \overline{y}\cdot x \vee y\cdot \overline{x} = x \oplus y,$$

на выходе триггера формируется сумма по модулю 2 слагаемых $x$ и $y$.

**Третий такт**. В третьем такте на вход триггера поступает значение, соответствующее третьему слагаемому *z*.

$$\tau(t+2) = \overline{T}\,\tau(t+1) \vee T\,\overline{\tau}(t+1) = \overline{y}\cdot x \vee y\cdot \overline{x} = \overline{z}(x \oplus y) \vee z(\overline{x \oplus y}) =$$
$$= \overline{x}\,\overline{y}z \vee x\,\overline{y}\,\overline{z} \vee xyz \vee \overline{x}\,\overline{y}z.$$

Из полученной функции видно, что на выходе *T*-триггера получена полная сумма трех слагаемых и, следовательно, триггер со счетным входом являет-

ся полным сумматором, работающим в три такта.

## 4.11. Стандартные функциональные узлы вычислительной техники

### 4.11.1. Шифраторы и дешифраторы

Шифратор (кодер) (от английского слова *encoder*) – это комбинационное цифровое устройство, преобразующее позиционный *n*-разрядный код в *k*-разрядный *r*-ичный код (двоичный, троичный и т. д.). В общем случае единичный сигнал на одном из входов преобразуется в *k*-разрядный двоичный код. Один из вариантов его применения – в устройствах ввода информации (пультах управления) для преобразования десятичных чисел в двоичную систему счисления.

Предположим, на пульте десять клавиш от нуля до девяти. При нажатии любой из них на вход шифратора подается единичный сигнал ($x_0$, ..., $x_9$). На выходе шифратора должен появиться двоичный код этого десятичного числа. Как видно из таблицы истинности (табл. 27), в этом случае нужен преобразователь с десятью входами и четырьмя выходами.

Таблица 27

| № | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |

Функциональная схема и условно графическое изображение шифратора приведены на рис. 50

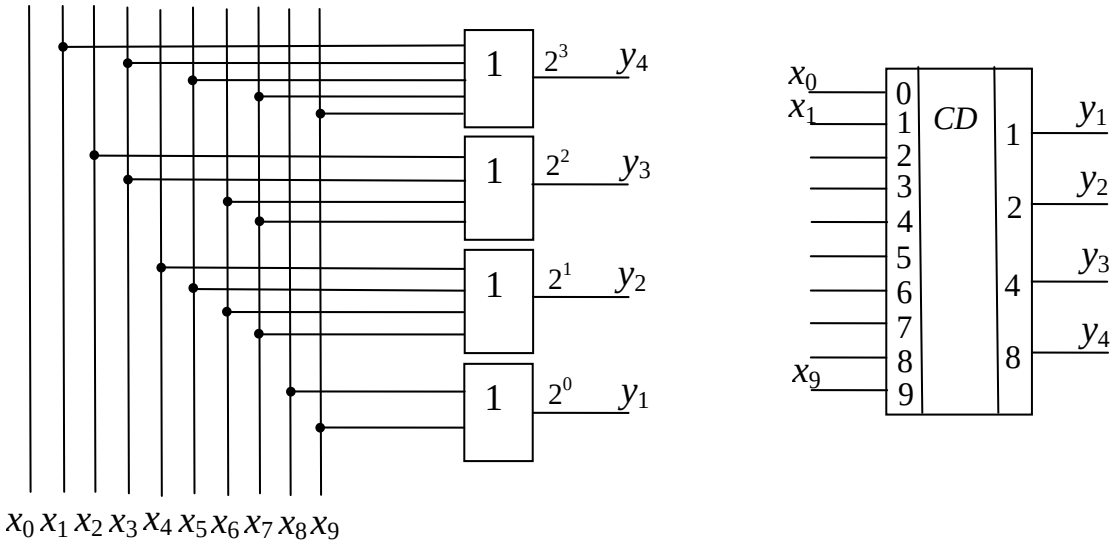



Рис. 50. Структура и УГО шифратора

Дешифратор (декодер) (от английского слова *decoder*) – это комбинационная логическая схема, преобразующая поступающий на ее входы двоичный позиционный код в активный сигнал только на одном из выходов (унитарный код). Обычно указанный в схеме номер вывода дешифратора соответствует десятичному эквиваленту двоичного кода, подаваемого на вход дешифратора в качестве входных переменных. Иначе говоря, при подаче на вход устройства двоичного кода на выходе дешифратора появится сигнал на том выходе, номер которого соответствует десятичному эквиваленту двоичного кода. Отсюда следует, что в любой момент времени выходной сигнал будет иметь место только на одном выходе дешифратора. В зависимости от типа дешифратора, этот сигнал может иметь как уровень логической единицы (при этом на всех остальных выходах уровень логического нуля), так и уровень логического нуля (при этом на всех остальных выходах уровень логической единицы).

В дешифраторах каждой выходной функции соответствует только один минтерм, а количество функций определяется количеством разрядов двоичного числа. Если дешифратор реализует все минтермы входных переменных, то он называется полным дешифратором (в качестве примера неполного дешифратора можно привести дешифратор двоично-десятичных чисел).

Рассмотрим пример синтеза дешифратора (полного), в короторм количество разрядов двоичного числа составляет 3, количество выходов – 8. Таблица истинности, соответствующая устройству представлена в табл. 28.

Таблица 28

| $x_1x_2x_3$ | $y_1y_2y_3y_4y_5y_6y_7y_8$ |
|---|---|
| 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 |

Из таблицы следует, что для реализации полного дешифратора (рис. 51) на $m$ входов (переменных) потребуются $n = 2^m$ элементов конъюнкции (количество входов каждого элемента равно $m$).

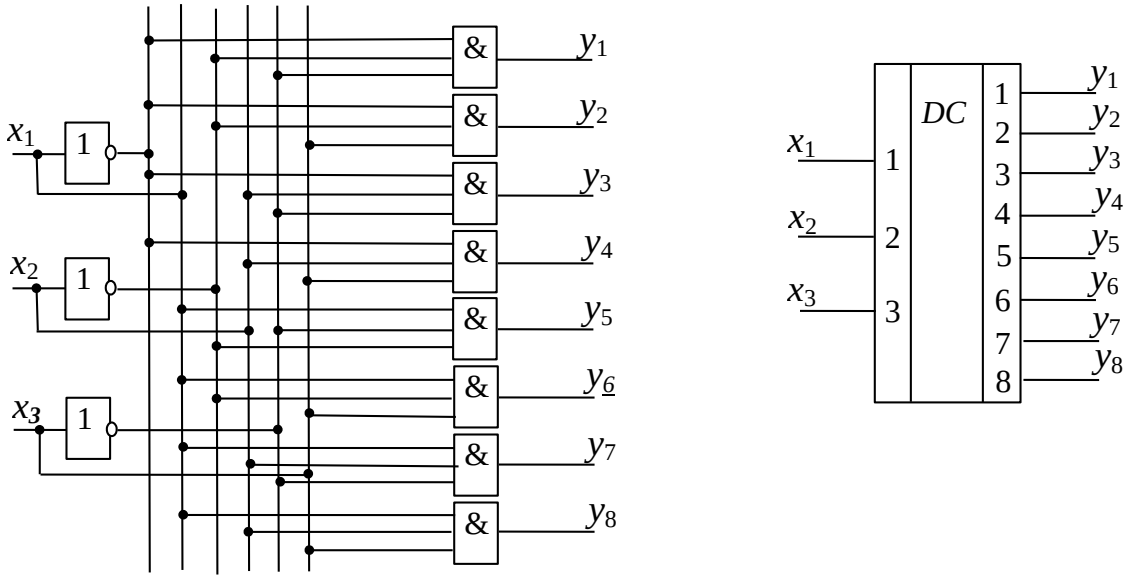



Рис. 51. Структура и УГО дешифратора

### 4.11.2. Мультиплексоры и демультиплексоры

Мультиплексором (от английского слова *multiplex* − многократный) называется комбинационное цифровое устройство, способное коммутировать (передавать) информацию с нескольких входов на один выход. Мультиплексор имеет несколько информационных входов, один или более управляющих (адресных) входов и один выход. Выбор информационного входа осуществляется подачей соответствующей комбинации (номера этого входа) на управляющие входы мультиплексора.

Мультиплексор представляет собой устройство с $m$ информационными, $n$ управляющими входами и одним выходом. Функционально мультиплексор состоит из $m$ элементов конъюнкции, выходы которых объединены с помощью элемента ИЛИ с $m$ входами. На один вход каждого из элементов конъюнкции подается один из информационных сигналов, а на другие входы элементов конъюнкции подаются сигналы, соответствующие номеру логического элемента И, сигнал с которого должен быть пропущен на выход мультиплексора. Таблица истинности, описывающая работу мультиплексора «1 из 4», имеющего четыре информационных ($x_1x_2x_3x_4$) и два управляющих входа ($A_0A_1$), представлена в табл. 29:

Таблица 29

| $A_0A_1$ | $x_1\,x_2\,x_3\,x_4$ | $F$ |
|---|---|---|
| 0 0 | 0 $x$ $x$ $x$ | 0 |
| 0 0 | 1 $x$ $x$ $x$ | 1 |
| 0 1 | $x$ 0 $x$ $x$ | 0 |
| 0 1 | $x$ 1 $x$ $x$ | 1 |
| 1 0 | $x$ $x$ 0 $x$ | 0 |
| 1 0 | $x$ $x$ 1 $x$ | 1 |
| 1 1 | $x$ $x$ $x$ 0 | 0 |
| 1 1 | $x$ $x$ $x$ 1 | 1 |

Функциональная схема и УГО мультиплексора приведена на рис. 52.

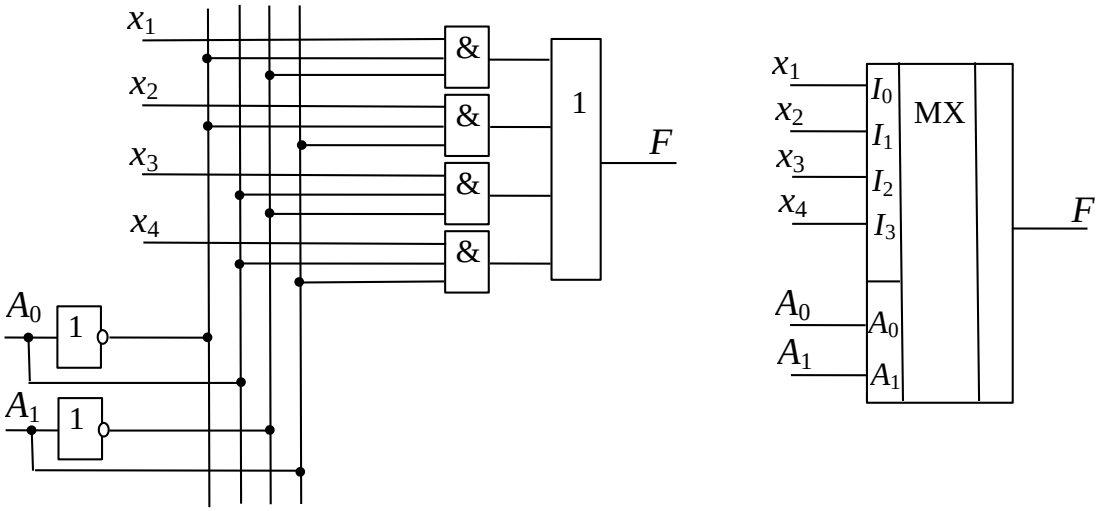



Рис. 52. Структура и УГО мультиплексора

Для обеспечения возможности синхронной работы мультиплексора с другими узлами целесообразно ввести дополнительный вход, на который будет подаваться сигнал разрешения (запрета) работы мультиплексора (стробирующий сигнал).

Демультиплексор выполняет функцию, обратную мультиплексору, т. е. в соответствии с принятым адресным значением, направляет информацию с единственного входа *D* на один из выходов, номер которого соответствует значению адреса. При этом на остальных выходах будут логические нули (единицы). Принцип работы демультиплексора «из 1 в 4» иллюстрируется таблицей истинности (табл. 30):

Таблица 30

| $A_0A_1$ | $D$ | $F_1F_2F_3F_4$ |
|---|---|---|
| 0 0 | 0 | 0 0 0 0 |
| 0 0 | 1 | 1 0 0 0 |
| 0 1 | 0 | 0 0 0 0 |
| 0 1 | 1 | 0 1 0 0 |
| 1 0 | 0 | 0 0 0 0 |
| 1 0 | 1 | 0 0 1 0 |
| 1 1 | 0 | 0 0 0 0 |
| 1 1 | 1 | 0 0 0 1 |

Функциональная схема и УГО демультиплексора приведена на рис. 53.

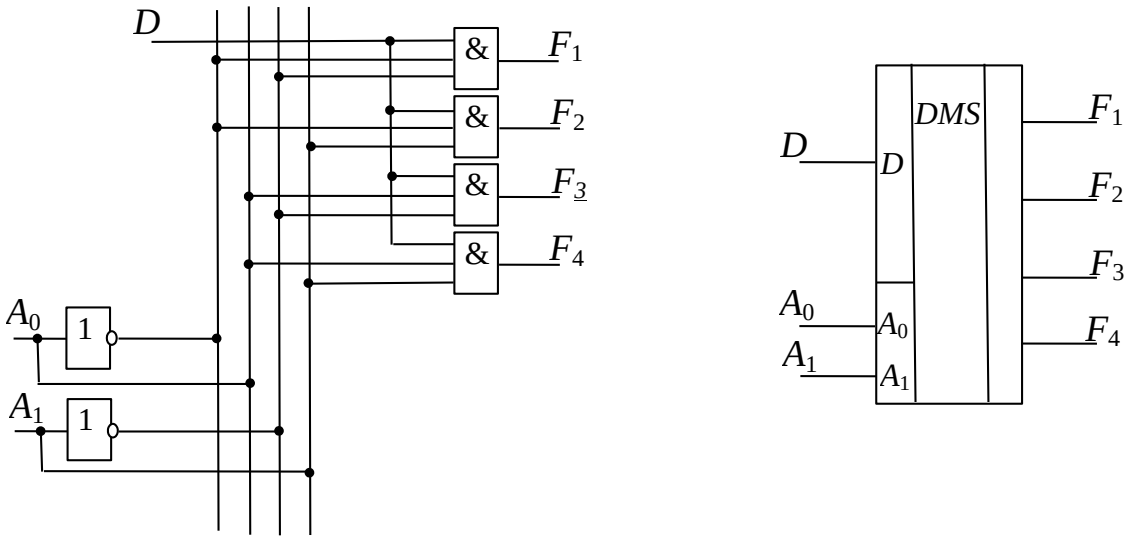



Рис. 53. Структура и УГО демультиплексора

Демультиплексоры эффективно используются для преобразования последовательного кода в параллельный. Совместное использование мультиплексора с демультиплексором обеспечивает последовательную передачу информации с преобразованием параллельного кода в последовательный и последовательного в параллельный. В таком применении в качестве управляющих сигналов используются выходы двоичного счетчика.

### 4.11.3. Регистры

Регистрами называют цифровые устройства, предназначенные для временного хранения информации, которая подается на них в виде многоразрядных двоичных чисел. При этом каждому разряду числа, записанного в регистр, соответствует свой разряд регистра. Основой любого регистра является элемент памяти – триггер. Количество триггеров, размещенных параллельно или объединенных последовательно, определяет разрядность регистров.

В регистрах используются *RS, JK* и *D*-триггеры. Для обеспечения управления записью информации в триггеры и ее считыванием используются комбинационные схемы, определяющие алгоритм управления регистрами.

Регистры могут классифицироваться по разным признакам, но основным является способ ввода и вывода информации. Исходя из этого они разделяются на следующие типы:

- параллельные (накопительные или регистры памяти);
- последовательные (регистры сдвига);
- последовательно-параллельные или комбинированные регистры.

#### 4.11.3.1. Параллельные регистры

Самыми простыми структурами являются параллельные регистры. Триггеры, используемые в таких регистрах должны быть синхронными для обеспечения управления схемой. В регистрах параллельного действия запись числа

осуществляется параллельным кодом, т. е. во все разряды регистра значения записываются одновременно.

На рис. 54 изображена простейшая структура записывающего трехразрядного регистра, построенного на основе синхронизированного *D*-триггера.

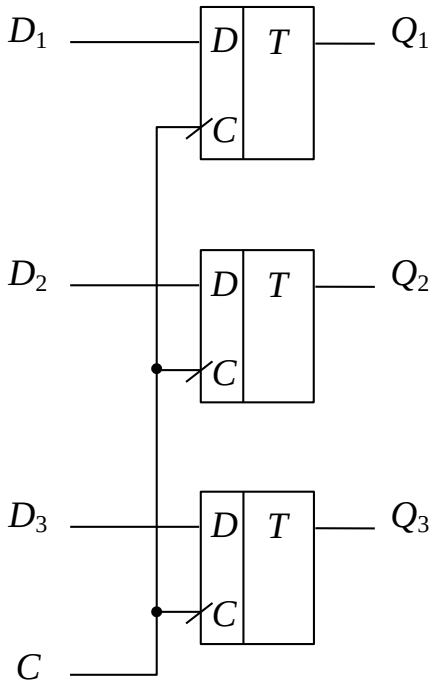



Рис. 54. Параллельный регистр на базе синхронизированных *D*-триггеров

Одновременно во все разряды регистра записываются данные, поступающие на его входы.

### 4.11.3.2. Последовательные регистры

Последовательные регистры могут быть однонаправленными или двунаправленными (реверсными).

В ЭВМ на ряду с параллельным используется также последовательный способ представления (передачи) двоичной информации. Для этого и используются регистры последовательного действия, основу которых составляют регистры сдвига. Регистр сдвига осуществляет операцию сдвига записанного в него двоичного числа влево или вправо на один или несколько разрядов при подаче специального управляющего сигнала «сдвиг». Регистр должен работать следующим образом: в момент прихода синхронизирующего сигнала *C* число сдвигается в регистре вверх на один разряд, а в освобождающиеся слева разряды вводятся разряды числа, поступившего в последовательном коде на его вход.

На рис. 55 изображен однонаправленный последовательный трехразрядный регистр сдвига. Как и параллельный регистр, он также построен на основе *D*-триггеров.

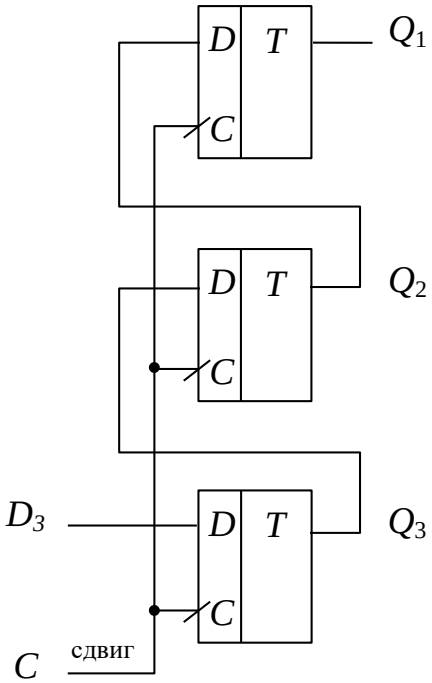



Рис. 55. Последовательный однонаправленный регистр на базе синхронизированных *D*-триггеров

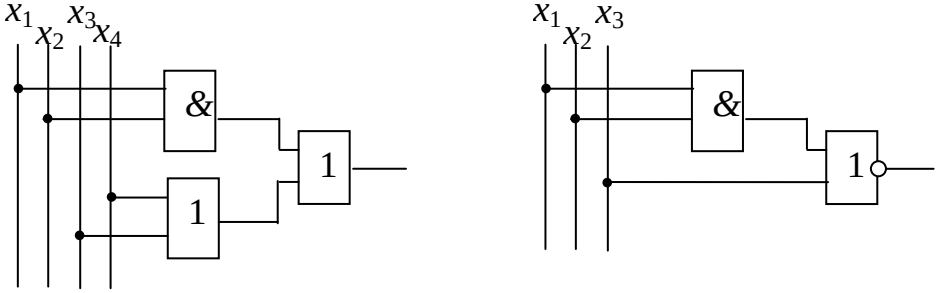
Поскольку для позиционных систем счисления вес разряда определяется его положением в числе, то сдвиг числа на один разряд влево в разрядной сетке соответствует умножению числа на основание системы счисления, а вправо – соответственно делению.

**Контрольные вопросы и задания**

1. Определите понятие «булева алгебра».
2. Назовите три основные операции булевой алгебры.
3. Дайте определение функционально полного базиса.
4. Дано логическое выражение. Нарисуйте для него логическую схему и составте таблицу истинности.

а) $\overline{x_1 (x_2 \vee x_3) \; x_4}$, б) $\overline{(x_1 \vee x_2)(x_3 \vee x_4)}$,
в) $\overline{(x_1 \vee x_2)} \; \overline{(x_3 \vee x_4)}$, г) $\overline{x_1 x_2 x_3}$,
д) $x_1 x_2 \vee \overline{x_1 x_3}$, е) $x_4 \; (x_1 x_2 x_3 \vee \overline{x}_2 \overline{x}_3)$.

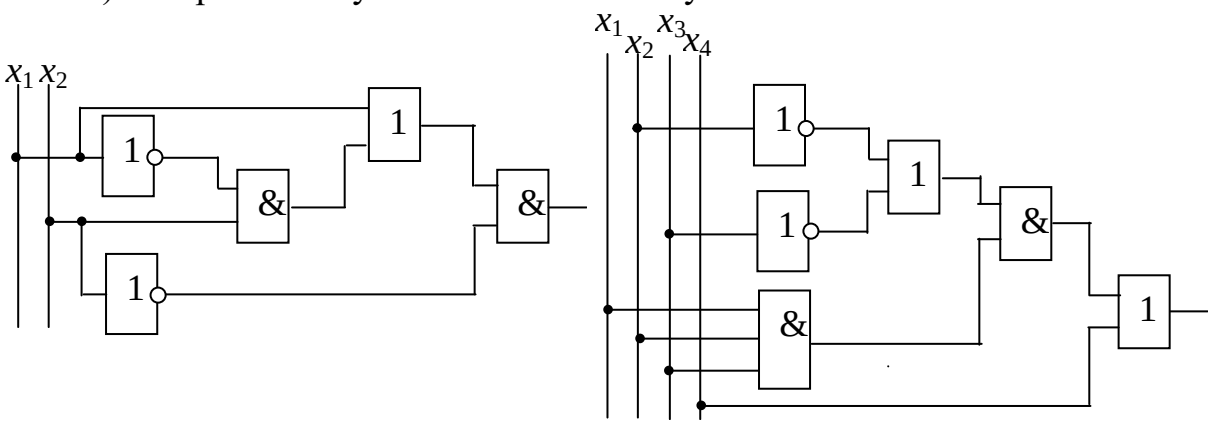
5. Для данных логических схем напишите логические выражения.




6. Для данных логических схем необходимо:
   а) написать логическое выражение;

б) упростить выражение;

в) построить новую логическию схему.




7. Выполните минимизацию булевой функции f на единичных наборах (метод минимизации любой), заданной в числовой форме:

а) $f^1(x_1x_2x_3x_4) = \vee$ 0, 3, 4, 8, 11, 12, 15;

б) $f^1(x_1x_2x_3x_4) = \vee$ 1, 5, 7, 8, 9, 10, 13, 15;

в) $f^1(x_1x_2x_3x_4) = \vee$ 1, 3, 5, 7, 10, 11, 15;

г) $f^1(x_1x_2x_3x_4) = \vee$ 0, 1, 2, 3, 7, 12, 13, 15;

д) $f^1(x_1x_2x_3x_4) = \vee$ 0, 1, 3, 4, 7, 8, 9, 11, 12.

При этой форме записи функция принимает единичное значение на перечисленных наборах таблицы истинности. Запишите минимальную функцию в форме ДНФ, КНФ.

## 5. Введение в теорию конечных автоматов

### 5.1. Основные понятия теории автоматов

Все рассмотренные выше устройства относятся к классу комбинационных схем, т. е. дискретных устройств без памяти. Наряду с ними в цифровой технике широкое распространение получили последовательностные автоматы, или, иначе, комбинационные схемы, объединенные с элементами памяти.

Под термином *автомат* можно понимать некоторое реально существующее устройство, функционирующее на основании как сигналов о состоянии внешней среды, так и внутренних сигналов о состоянии самого автомата. В этом плане ЭВМ может быть рассмотрена как цифровой автомат. Под цифровым автоматом понимается устройство, предназначенное для преобразования цифровой информации. С другой стороны, под термином *автомат* можно понимать математическую модель некоторого устройства. Общая теория автоматов подразделяется на две части: *абстрактную* и *структурную*. Различие между ними состоит в том, что абстрактная теория абстрагируется от структуры как самого автомата, так и входных и выходных сигналов. В абстрактной теории анализируются переходы автомата под воздействием абстрактных входных слов и формируемые на этих переходах абстрактные выходные слова.

В структурной теории рассматривается структура как самого автомата, так и его входных и выходных сигналов, способы построения автоматов из элементарных автоматов, способы кодирования входных и выходных сигналов, состояний автомата.

В соответствии с этим принято различать две модели автоматов: абстрактную и структурную. Абстрактная модель применяется при теоретическом рассмотрении автоматов. Структурная модель служит для построения схемы автомата из логических элементов и триггеров и предназначена для выполнения функции управления.

*Абстрактный автомат* – это математическая модель цифрового автомата, задаваемая шестикомпонентным вектором $S = (A, Z, W, \delta, \lambda, a_1)$, где $A = \{a_1, \ldots, a_m\}$ – множество внутренних состояний абстрактного автомата; $Z = \{z_1, \ldots, z_k\}$ и $W = \{w_1, \ldots, w_l\}$ – соответственно множества входных и выходных абстрактных букв; $\delta$ − функция переходов; $\lambda$ − функция выходов; $a_1$ – начальное состояние автомата. Абстрактный автомат может быть представлен как устройство с одним входом и одним выходом (рис. 56), на которые подаются абстрактные входные слова и формируются абстрактные выходные слова.

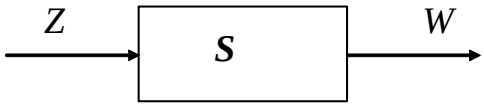



Рис. 56. Абстрактный автомат

Понятие *состояние автомата* используется для описания систем, выходы которых зависят не только от входных сигналов, но и от предыстории, т. е. информации о том, что происходило с автоматом в предыдущий интервал времени. Состояние автомата позволяет устранить время как явную переменную и выразить выходные сигналы как функцию состояний и входных сигналов.

По виду функции выходов все множество автоматов можно подразделить на два класса: автоматы Мили и автоматы Мура.

*Автоматами Мура (Moore),* или автоматами первого типа, называют автоматы, для которых буква выходного алфавита $w(t)$ не зависит явно от буквы входного алфавита $z(t)$, а определяется только состоянием автомата в момент времени $t$. Закон функционирования автомата Мура может быть описан системой уравнений

$$\begin{cases} a(t+1) = \delta(a(t), z(t)); \\ w(t) = \lambda(a(t)). \end{cases}$$

К автоматам второго типа, или *автоматам Мили (Mealy)*, относятся автоматы, поведение которых может быть описано системой уравнений

$$\begin{cases} a(t+1) = \delta(a(t), z(t)); \\ w(t) = \lambda(a(t), z(t)). \end{cases}$$

Следовательно, в отличие от автомата Мура для автомата Мили буква выходного алфавита $w(t)$ зависит не только от текущего состояния, но и от буквы входного алфавита.

Между моделями автоматов Мили и Мура существует соответствие, позволяющее преобразовать закон функционирования одного из них в другой.

*Совмещенная модель автомата (С-автомат).* Абстрактный *С*-автомат – математическая модель дискретного устройства, определяемого вектором $S = (A, Z, W, U, \delta, \lambda_1, \lambda_2, a_1)$, где $A$, $Z$, $\delta$ и $a_1$ определены выше, а $W = \{w_1, \ldots, w_l\}$ и $U = \{u_1, \ldots, u_l\}$ – выходной абстрактный алфавит автомата Мили и Мура соответственно, $\lambda_1$ и $\lambda_2$ − функции выходов. Абстрактный *С*-автомат может быть представлен как устройство с одним входом, на который поступают слова из входного алфавита $Z$, и двумя выходами (рис. 57), на которых формируются абстрактные выходные слова из выходных алфавитов $W$ и $U$.

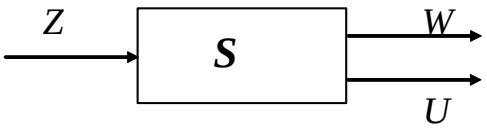



Рис. 57. Абстрактный *С*-автомат

Отличие *С*-автомата от моделей автоматов Мили и Мура заключается в том, что он одновременно реализует две функции выходов $\lambda_1$ и $\lambda_2$, каждая из которых характерна для одной из двух моделей.

$$\begin{cases} a(t+1) = \delta(a_m(t), z(t)) ; \\ w(t) = \lambda_1(a_m(t), z(t)) ; \\ u(t) = \lambda_2(a_m(t)) . \end{cases}$$

Автомат $S$ называется *конечным*, если конечны множества $A$, $Z$ и $W$, и *детерминированным*, если, находясь в некотором состоянии, он не может перейти более чем в одно состояние под действием одной и той же буквы входного алфавита. Если в автомате выделено начальное состояние $a_1$, то он называется *инициальным*. Состояние $a_s$ называется *устойчивым,* если для любого $z_k \in Z$, такого что $a_s = \delta(a_m, z_k)$, $a_s = \delta(a_s, z_k)$, т. е. если автомат из состояния $a_m$ перешел в состояние $a_s$ под действием буквы входного алфавита $z_k$, то выйти из этого состояния он может только под действием другой входной буквы. Автомат $S$ является *асинхронным,* если каждое его состояние устойчиво, иначе *синхронным*. Автомат называется *полностью определенным*, если область определения функции $\delta$ совпадает с множеством пар $(a_m, z_k)$, а функции $\lambda$ для автомата Мили − с множеством пар $(a_m, z_k)$, Мура − с множеством $a_m$. У *частично* определенного автомата функции $\delta$ и $\lambda$ определены не для всех пар.

### 5.2. Способы задания автоматов

Закон функционирования автоматов может быть задан в виде систем уравнений, таблиц, матриц и графов. Под законом функционирования понимается совокупность правил, описывающих переходы автомата в новое состояние и формирование выходных символов в соответствии с последовательностью входных символов.

В зависимости от типа автомата при табличном задании закона функционирования автомата используются либо таблицы переходов и выходов (автомат Мили), либо совмещенная таблица переходов и выходов (автомат Мура) [19]. С помощью табл. 31 и 32 (таблицы переходов и таблицы выходов соответственно) задан закон функционирования абстрактного автомата Мили, для которого

$A = \{a_1, a_2, a_3, a_4\}$, $Z = \{z_1, z_2, z_3\}$, $W = \{w_1, w_2, w_3, w_4, w_5\}$.

Таблица 31

| $\delta$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ |
|---|---|---|---|---|
| $z_1$ | $a_2$ | $a_2$ | − | $a_4$ |
| $z_2$ | $a_4$ | − | $a_1$ | $a_3$ |
| $z_3$ | $a_3$ | $a_3$ | $a_4$ | − |

$a(t + 1) = \delta(a(t), z(t))$

Таблица 32

| $\lambda$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ |
|---|---|---|---|---|
| $z_1$ | $w_1$ | $w_2$ | − | $w_3$ |
| $z_2$ | $w_2$ | − | $w_4$ | $w_5$ |
| $z_3$ | $w_3$ | $w_1$ | $w_3$ | − |

$w(t) = \lambda(a(t), z(t))$

Строки таблиц отмечены буквами входного алфавита (элементы множества $Z$), а столбцы − состояниями (элементы множества $A$). Буквы входного алфавита и состояния, которыми помечены строки и столбцы, относятся к моменту времени $t$. В табл. 31 (таблице переходов) на пересечении строки $z_i(t)$ и столбца $a_m(t)$ ставится состояние $a_s(t + 1) = \delta(a_m(t), z_i(t))$. В табл. 32 (таблице

выходов) на пересечении строки $z_i(t)$ и столбца $a_m(t)$ ставится буква выходного алфавита $w(t) = \lambda(a_m(t), z_i(t))$, соответствующий переходу из состояния $a_m$ в состояние $a_s$. Таким образом, по таблицам переходов и выходов можно проследить последовательность работы автомата. Так, например, в начальный момент времени $t = 0$ автомат, находясь в состоянии $a_1$ (первый столбец), под действием буквы входного алфавита $z_1$ может перейти в состояние $a_2$, при этом буква выходного алфавита не формируется; под действием буквы $z_2$ – в состояние $a_4$ с формированием буквы $w_2$; под действием буквы $z_3$ – в состояние $a_3$ с формированием буквы $w_3$. Далее если на вход автомата, установленного в исходное состояние $a_m \in A$, в моменты времени $t = 1, 2, \ldots, n$ подается некоторая последовательность букв входного алфавита (входных символов) $z_i \in Z$, то на выходе автомата будут последовательно формироваться буквы выходного алфавита (выходные символы) $w_j \in W$, при этом автомат будет переключаться в состояния $a_s \in A$. Следовательно, с помощью таблиц переходов и выходов можно получить выходную реакцию автомата на любое входное слово.

Как отмечалось выше, для автомата Мура буква выходного алфавита не зависит от буквы входного, а определяется только текущим состоянием автомата. Это позволяет для автомата Мура объединить обе таблицы (переходов и выходов) в одну *совмещенную таблицу*. В совмещенной таблице (табл. 33) переходов и выходов каждый столбец отмечается состоянием $a_m \in A$ и буквой выходного алфавита $w(t) = \lambda(a(t))$, соответствующим этому состоянию.

Таблица 33

| $\lambda$ | $w_1$ | $w_2$ | – | $w_3$ |
|---|---|---|---|---|
| $\delta$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ |
| $z_1$ | $a_2$ | $a_2$ | – | $a_4$ |
| $z_2$ | $a_4$ | – | $a_1$ | $a_3$ |
| $z_3$ | $a_3$ | $a_3$ | $a_4$ | – |

$$a(t + 1) = \delta(a(t), z(t))$$
$$w(t) = \lambda(a(t))$$

Другим наглядным способом описания закона функционирования автомата является представление его в виде графа. Граф автомата – ориентированный граф, вершины которого соответствуют состояниям, а дуги – переходам между ними. Дуга, направленная из вершины $a_m$ в вершину $a_s$, соответствует переходу из состояния $a_m$ в $a_s$. В начале дуги записывается буква входного алфавита $z_i$, влияющий на переход $a_s=\delta(a_m, z_i)$, а буква выходного алфавита $w_j$ записывается в конце дуги (для автомата Мили) или рядом с вершиной (для автомата Мура). На рис. 58 приведен граф автомата Мили, соответствующий закону функционирования, описанному выше (см. табл. 31 и 32), а на рис. 59 приведен граф автомата Мура (см. табл. 33).

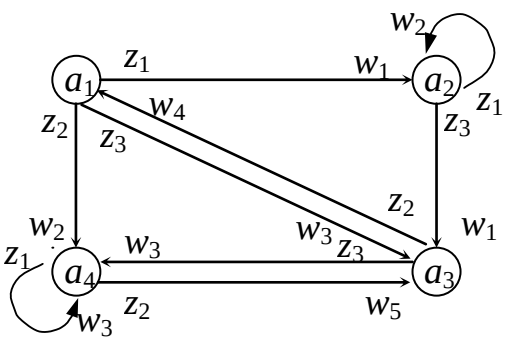



Рис. 58. Граф автомата Мили

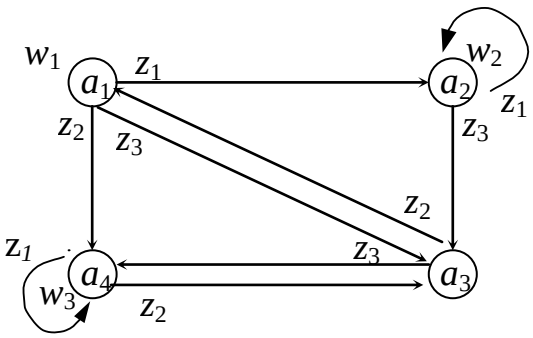



Рис. 59. Граф автомата Мура

### 5.3. Структурный автомат

В отличие от абстрактного автомата, имеющего один вход и один выход, структурный автомат имеет $L$ входов и $N$ выходов. На входы структурного автомата поступают наборы входных двоичных переменных из множества $X = \{x_1, x_2, \ldots, x_L\}$, а на выходах формируются выходные двоичные сигналы из множества $Y = \{y_1, y_2, \ldots, y_N\}$. Очевидно, что $L = ]\log_2|Z|[$ и $N = ]\log_2|W|[$ [3].

Структурная модель автомата представляет собой две взаимосвязанные части: комбинационную схему (КС) и блок памяти. Комбинационная часть автомата кроме сигналов из множества $Y$ формирует также двоичные сигналы, подаваемые на входы элементов памяти $D = \{d_1, d_2, \ldots, d_R\}$. Эти сигналы управляют кодом состояния автомата и называются *функциями возбуждения элементов памяти*. Сигналы, формируемые на выходах элементов памяти $T = \{\tau_1, \tau_2, \ldots, \tau_R\}$, являются кодом состояния автомата, они подаются на входы комбинационной схемы наряду с входными переменными и называются *переменными обратной связи*. Структурная схема автомата изображена на рис. 60.

Любому переходу в абстрактном автомате из состояния $a_m$ в состояние $a_s$ под действием буквы входного алфавита $z_i$ с формированием буквы выходного алфавита $w_j$ соответствует переход в стуктурном автомате из состояния $a_m$ с кодом $\tau_1, \ldots, \tau_R$ в состояние $a_s$ с кодом $d_1, \ldots, d_R$ ($R = ]\log_2|A|[$) под действием набора входных сигналов $x_1, \ldots, x_L$ с формированием выходного набора $y_1, \ldots, y_N$.

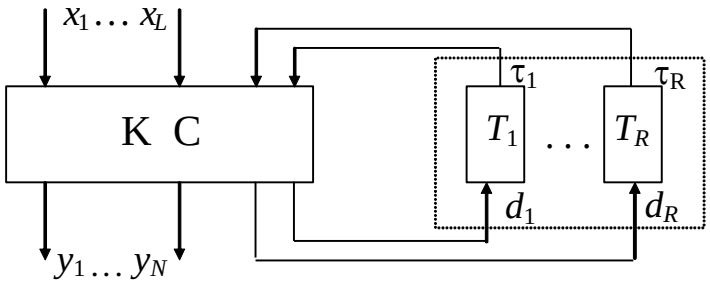



Рис. 60. Структурный автомат

### 5.4. Память автомата

В качестве элементов памяти автомата используются простейшие схемы, предназначенные для приема, хранения и передачи одного бита информации − триггеры. Триггер – простейший цифровой автомат с памятью (изобретен в 1918 году Бонч-Бруевичем, руководителем Нижегородской лаборатории связи). Триггер имеет два устойчивых состояния, одно из которых соответствует двоичной единице, а другое – двоичному нулю. Термин триггер происходит от английского слова *trigger* – защёлка, спусковой крючок. Для обозначения этой схемы в английском языке чаще употребляется термин *flip-flop*, что в переводе означает «хлопанье». Это звукоподражательное название электронной схемы указывает на её способность почти мгновенно переходить («перебрасываться») из одного электрического состояния в другое и наоборот. Триггер имеет один или более входов и два выхода (прямой и инверсный). Выходные сигналы триггера зависят только от его состояния и изменяются только при смене состояния триггера. Таким образом, триггеры являются элементарными автоматами Мура (элементарными, так как они имеют только два устойчивых состояния). В основе любого триггера находится регенеративное кольцо из двух инверторов.

Триггеры можно классифицировать по следующим признакам:

1) по способу записи информации: несинхронизируемые (асинхронные) и синхронизируемые (синхронные) триггеры. У асинхронных триггеров запись информации происходит под действием информационных сигналов, у синхронных кроме информационных на вход должны быть поданы разрешающие сигналы;

2) по способу синхронизации: синхронные триггеры со статическим управлением записью, синхронные двухступенчатые триггеры, синхронные триггеры с динамическим управлением записью;

3) по способу организации логических связей: триггеры с раздельной установкой состояния (*RS*-триггеры), триггеры со счетным входом (*T*-триггеры), универсальные триггеры с раздельной установкой состояний (*JK*-триггеры), триггеры с приемом информации по одному входу (*D*-триггеры), комбинированные триггеры (*RST*-, *JKRS*-, *DRS*-триггеры и т. д.), триггеры со сложной входной логикой.

Приняты следующие изображения входов триггеров:

− *S* − раздельный вход установки триггера в единичное состояние по прямому выходу;

− *R* − раздельный вход сброса триггера в нулевое состояние по прямому выходу;

− *J* и *K* − назначение аналогично входам *S* и *R*;

− *D* − информационный вход; используется для приема информации, записываемой в триггер;

− *T* − счетный вход;

− *C* − вход синхронизации (разрешения записи в триггер).

***D*-триггер**. Триггером типа *D* (от английского *Delay* − задержка) называется триггер с двумя устойчивыми состояниями равновесия и одним информационным входом *D*. На практике наиболее часто используется

синхронный *D*-триггер с входом *C*. На рис. 61 приведена схема одноступенчатого синхронного *D*-триггера на элементах И-НЕ и его условно графическое обозначение. Значения, поступающие на вход *D*, записываются на выход *Q*, т. е. триггер работает как повторитель. Принцип работы синхронного *D*-триггера основан на том, что сигнал на выходе после переключения равен сигналу на входе *D* до переключения.

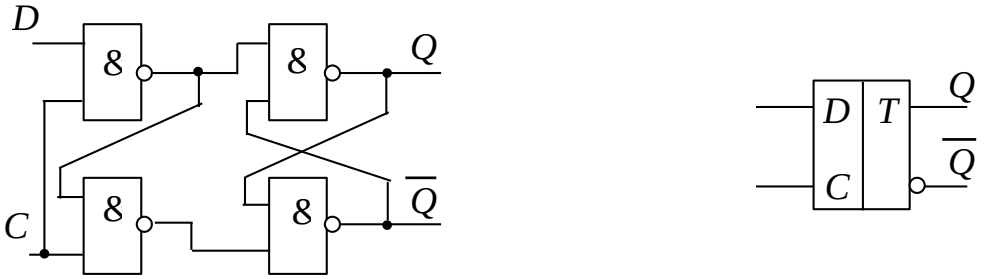



Рис. 61. *D*-триггер и УГО *D*-триггера

В табл. 34 приведена информация о работе *D*-триггера. Переключение состояний выполняется по формуле $Q_{t+1} = Q_t \overline{C} \vee DC$.

Таблица 34

| $Q_t$ | *D* | |
|---|---|---|
| | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 0 | 1 |

***T*-триггер** работает следующим образом: если на вход *T* подается сигнал «0», триггер не изменяет свое внутреннее состояние, если на вход *T* подается сигнал «1», триггер инвертирует свое внутреннее состояние. Но для установки триггера в начальное состояние одного *T*-входа недостаточно. *T*-триггер реализуется с установочным входом *R* (входом асинхронной установки). В таком исполнении он называется *RT*-триггером (или просто *T*-триггером). Когда на вход *R* подается сигнал «1», триггер устанавливается в нуль, когда на вход R подается сигнал «0», триггер выполняет свою обычную функцию (рис. 62).

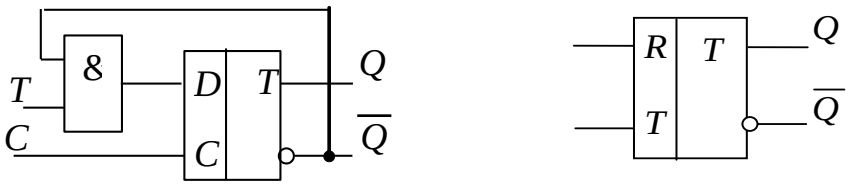



Рис. 62. *T*-триггер на основе *D*-триггера и УГО *T*-триггера

Принцип работы *T*-триггера основан на том, что единичный сигнал на входе изменяет содержимое триггера на противоположное. На рис. 62 приведена схема *T*-триггера на элементах И-НЕ и его условное изображение.

В табл. 35 приведена информация о работе *T*-триггера. Переключение состояний выполняется по формуле $Q_{t+1} = Q_t \oplus T$.

Таблица 35

| $Q_t$ | $T$ | |
|---|---|---|
| | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

***RS*-триггеры**. Триггером типа *RS* называется триггер с двумя устойчивыми состояниями равновесия и двумя информационными входами. Вход *S* (*Set*) служит для установки триггера в «1», вход *R* (*Reset*) для установки в «0». Одновременная подача двух активных сигналов на входы *R* и *S* запрещена, т. е. *R* и *S* не могут одновременно быть равны единице. Подача двух нулей на входы триггера сохраняет его внутреннее состояние. Активным значением сигнала на входе является уровень «1». Вход в этом случае считается прямым. Если активным значением сигнала на входе является нуль, то такой вход считается инверсным.

Асинхронные *RS*-триггеры являются простейшими триггерами. Такие триггеры строятся на логических элементах: 2ИЛИ-НЕ – триггер с прямыми входами (рис. 63) или 2И-НЕ – триггер с инверсными входами. Выход каждого из логических элементов подключен к одному из входов другого элемента, что обеспечивает нахождение триггера в одном из двух устойчивых состояний.

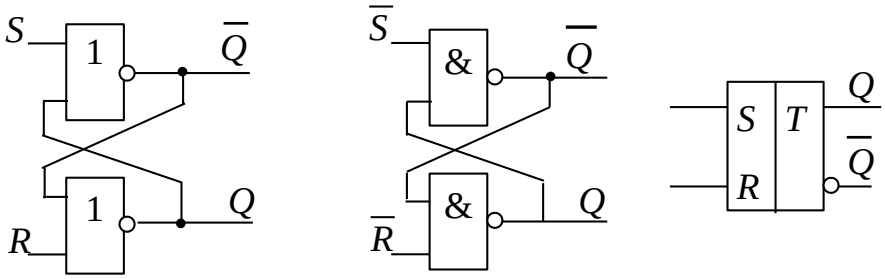



Рис. 63. *RS*-триггер и УГО *RS*-триггера

Информация из табл. 36 определяет переходы *RS*-триггера.

Таблица 36

| $Q_t$ | $R\ S$ | | | |
|---|---|---|---|---|
| | 00 | 01 | 10 | 11 |
| 0 | 0 | 1 | 0 | *x* |
| 1 | 1 | 1 | 0 | *x* |

Возможны следующие режимы работы *RS*-триггера:

1) $S = 0$, $R = 0$ – режим хранения информации (значение триггера не изменяется);

2) $S = 0$, $R = 1$ – режим сброса (триггер всегда устанавливается в 0);

3) $S = 1$, $R = 0$ – режим записи логической единицы (триггер устанавливается в 1);

4) $S = 1$, $R = 1$ – запрещенная комбинация (значение триггера неопределенное).

Переключение состояний выполняется по формуле $Q_{t+1} = Q_t \overline{R}$ V $S$. Иначе эта функция может быть найдена по карте Карно (рис. 64), которая строится

по таблице переходов триггера (см. табл. 36).

| $R\,S \setminus Q_t$ | 0 | 1 |
|---|---|---|
| 00 | 0 | 1 |
| 01 | 1 | 1 |
| 11 | * | * |
| 10 | 0 | 0 |

Рис. 64. Карта Карно для функции переходов *RS*-триггера

Еще одним способом описания триггеров является граф переходов (рис. 65). Вершинам соответствуют состояния триггеров, а дугам – переходы между состояниями. Состояние определяется значением выхода *Q*. Когда $Q = 0$, считается, что триггер находится в состоянии $a_0$, когда $Q = 1$, считается, что триггер находится в состоянии $a_1$. На дугах записываются условия того или иного переходов.

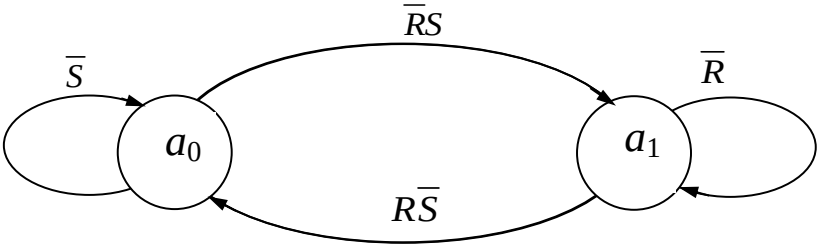



Рис. 65. Граф переходов *RS*-триггера

***JK*-триггеры.** Триггером типа *JK* называется триггер с двумя устойчивыми состояниями равновесия и двумя информационными входами. Вход *J* (*Jarк*) служит для установки триггера в «1», вход *K* (*Kill*) для установки в «0». Активным значением сигнала на входе является уровень «1». Одновременная подача двух активных сигналов на входы *K* и *J* не запрещена, при этом на выходе появляется инверсное значение состояния триггера *Q*. Подача двух нулей на входы триггера сохраняет его внутреннее состояние.

Асинхронный *JK*-триггер строится на базе *RS*-триггера. Простейший *JK*-триггер можно получить из *RS*-триггера, если ввести дополнительные обратные связи с выходов триггера на входы, которые позволяют устранить неопределенность в таблице состояний. Логическая схема и УГО *JK*-триггера приведены на рис. 66.

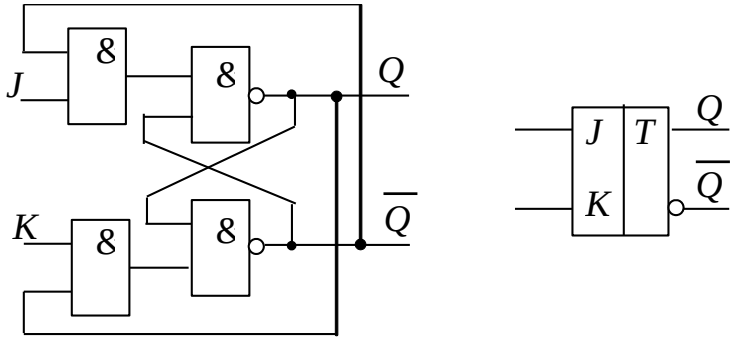



Рис. 66. *JK*-триггер и УГО *JK*-триггера

Табл. 37 определяет переходы *JK*-триггера которые выполняются согласно логической формулы $Q_{t+1} = \overline{Q}_t J \vee Q_t \overline{K}$.

Таблица 37

| $Q_t$ | $J\ K$ | | | |
|---|---|---|---|---|
| | 00 | 01 | 10 | 11 |
| 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 |

Возможны следующие режимы работы *RS*-триггера:
*J* = 0, *K* = 0 – режим хранения информации (значение триггера не изменяется);
*J* = 0, *K* = 1 – режим сброса (триггер устанавливается в 0);
*J* = 1, *K* = 0 – режим записи логической единицы (триггер устанавливается в 1);
*J* = 1, *K* = 1 – режим инверсии содержимого триггера.

Как и для *RS* –триггера функция переключения *JK*-триггера может быть получена по карте Карно (рис. 67), которая строится по таблице переходов триггера (см. табл.37).

| $J\ K \setminus Q_t$ | 0 | 1 |
|---|---|---|
| 00 | 0 | 1 |
| 01 | 1 | 1 |
| 11 | 1 | 0 |
| 10 | 0 | 0 |

Рис. 67. Карта Карно для функции переходов *JK*-триггера

Граф переходов *JK*-триггера представлен на рис. 68.

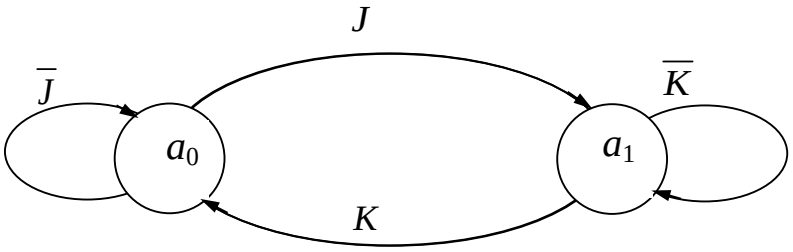



Рис. 68. Граф переходов *JK* триггера

В заключение можно отметить, что JK-триггер является универсальным триггером. Универсальность его состоит в том, что он может выполнять функции *RS*-, *T*- и *D*-триггеров. Для получения *D*-триггера *K* вход соединяется со входом *J* через инвертор. *T*-триггер получается из *JK*-триггера путем объединения входов *J* и *K* в один, называемый *T*-входом. Если *JK*-триггер предварительно установлен в 0 и на вход не подается комбинация 11, то он работает как *RS*-триггер.

### 5.5. Канонический метод структурного синтеза автомата

В общем случае, задача синтеза структурного автомата сводится к построению КС, имеющей (как правило) несколько выходов. Канонический метод синтеза разработан В.М. Глушковым [12]. Описание этого метода приводится в [3, 17]. Суть этого метода состоит в сведении задачи синтеза автоматов к построению системы булевых функций, описывающих поведение проектируемого автомата. При каноническом методе предполагается, что каждая выходная функция реализуется своей схемой, совокупность которых и дает требуемую КС. Поэтому синтез сложной КС с $n$ выходами заменяется синтезом $n$ схем с одним выходом.

Синтез цифровых устройств выполняется в два этапа:

1) этап абстрактного синтеза;
2) этап структурного синтеза.

На этапе абстрактного синтеза, для перехода от абстрактного автомата к структурному требуется:

1) поставить в соответствие каждой букве входного алфавита $Z = \{z_1, z_2, \ldots, z_k\}$ совокупность двоичных сигналов из множества $X = \{x_1, x_2, \ldots, x_L\}$, т. е. *закодировать буквы входного алфавита* абстрактного автомата. Значение $L$, определяющее количество двоичных переменных для кодирования абстрактных букв входного алфавита, вычисляется следующим образом: $L = ]\log_2|Z|[$, где $|Z|$ − мощность множества $Z$ (число различных элементов множества $Z$), $]n[$ − означает ближайшее целое число, большее либо равное $n$;

2) поставить в соответствие каждой букве выходного алфавита $W = \{w_1, w_2, \ldots, w_l\}$ совокупность двоичных выходных сигналов из множества $Y = \{y_1, y_2, \ldots, y_N\}$, т. е. *закодировать буквы выходного алфавита* абстрактного автомата. Значение $N$ вычисляется следующим образом: $N = ]\log_2|W|[$;

3) поставить в соответствие каждому состоянию абстрактного автомата $A = \{a_1, a_2, \ldots, a_m\}$ совокупность состояний элементов памяти $T = \{\tau_1, \tau_2, \ldots, \tau_R\}$, т. е. *закодировать состояния* абстрактного автомата. Количество элементов памяти определяется условием $R = ]log_2|A|[$.

На этапе структурного синтеза необходимо:

1) по результату абстрактного синтеза составить систему уравнений для функций $y_1, y_2, \ldots, y_N, d_1, d_2, \ldots, d_R$, в соответствии с которой будет построена комбинационная часть структурной схемы автомата. Подлежащие реализации булевы функции представляются в виде СДНФ;

2) при использовании методов минимизации определяются МДНФ или МКНФ этих функций;

3) полученные МДНФ (МКНФ) представляют в заданном базисе;

4) по представлению функции в заданном базисе строят комбинационную схему структурного автомата.

Полученная таким образом система булевых функций называется *канонической*.

Необходимо отметить, что подлежащая реализации булева функция может быть задана не на всех возможных наборах переменных. На тех наборах, где функция не определена, ее доопределяют так, чтобы в результате минимизации получить более простую МДНФ или МКНФ. При этом упростится и сама КС. Кроме того, довольно часто с целью получения еще более простого представления функции МДНФ, выполняется ее запись в скобочной форме, т. е. выносятся за скобки общие части импликант МДНФ.

### 5.5.1. Канонический метод структурного синтеза автомата Мили

Рассмотрим пример структурного синтеза автомата Мили, блок памяти которого построим на *T*-триггерах. Исходные данные для выполнения синтеза структурной схемы заданы таблично: таблицы переходов (табл. 38) и выходов (табл. 39). Описание работы *T*-триггера приведено в табл. 35.

Определяем вначале общее количество входов, выходов и элементов памяти автомата:

$L = ]\log_2|Z|[ = ]\log_2 4[ = 2;$
$N = ]\log_2|W|[ = ]\log_2 6[ = 3;$
$R = ]log_2|A|[ = ]\log_2 4[ = 2.$

Таблица 38

| $\delta$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ |
|---|---|---|---|---|
| $z_1$ | $a_2$ | $a_2$ | – | $a_4$ |
| $z_2$ | $a_4$ | – | $a_2$ | $a_2$ |
| $z_3$ | $a_3$ | $a_3$ | $a_4$ | $a_3$ |
| $z_4$ | – | $a_4$ | – | $a_1$ |

$a(t+1) = \delta(a(t), z(t))$

Таблица 39

| $\lambda$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ |
|---|---|---|---|---|
| $z_1$ | $w_1$ | $w_2$ | – | $w_5$ |
| $z_2$ | $w_2$ | – | $w_4$ | $w_6$ |
| $z_3$ | $w_3$ | $w_1$ | $w_3$ | $w_5$ |
| $z_4$ | – | $w_2$ | – | $w_1$ |

$w(t) = \lambda(a(t), z(t))$

Структурная схема автомата изображена на рис. 69.

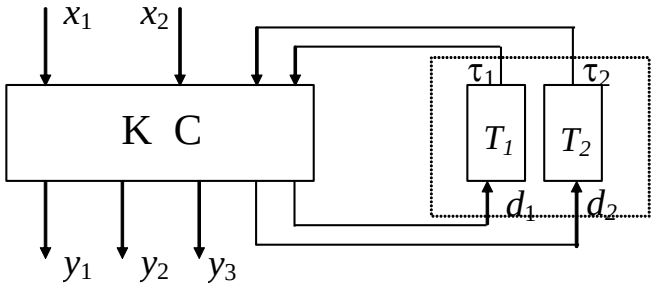



Рис. 69. Структурная схема автомата

На основании полученных значений построим таблицы (табл. 40–42) и выполним кодирование входного, выходного алфавита и внутренних состояний автомата (табл. 38–39).

Таблица 40

| Z \ X | $x_1$ | $x_2$ |
|---|---|---|
| $z_1$ | 0 | 0 |
| $z_2$ | 0 | 1 |
| $z_3$ | 1 | 0 |
| $z_4$ | 1 | 1 |

Таблица 41

| W \ Y | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|
| $w_1$ | 0 | 0 | 0 |
| $w_2$ | 0 | 0 | 1 |
| $w_3$ | 0 | 1 | 0 |
| $w_4$ | 0 | 1 | 1 |
| $w_5$ | 1 | 0 | 0 |
| $w_6$ | 1 | 0 | 1 |

Таблица 42

| A \ T | $\tau_1$ | $\tau_2$ |
|---|---|---|
| $a_1$ | 0 | 0 |
| $a_2$ | 0 | 1 |
| $a_3$ | 1 | 0 |
| $a_4$ | 1 | 1 |

По результатам кодирования строим таблицы переходов и выходов структурного автомата (табл. 43 и 44, соответственно переходов и выходов). Они получаются путем занесения соответствующих значений из табл. 40–42 в исходные таблицы (см. табл. 38, 39).

Таблица 43

| $\tau_1\tau_2$<br>$x_1x_2$ | 00<br>$a_1$ | 01<br>$a_2$ | 10<br>$a_3$ | 11<br>$a_4$ |
|---|---|---|---|---|
| 00<br>$z_1$ | 01 | 01 | – | 11 |
| 01<br>$z_2$ | 11 | – | 01 | 01 |
| 10<br>$z_3$ | 10 | 10 | 11 | 10 |
| 11<br>$z_4$ | – | 11 | – | 00 |

Таблица 44

| $\tau_1\tau_2$<br>$x_1x_2$ | 00<br>$a_1$ | 01<br>$a_2$ | 10<br>$a_3$ | 11<br>$a_4$ |
|---|---|---|---|---|
| 00<br>$z_1$ | 000<br>$w_1$ | 001<br>$w_2$ | – | 100<br>$w_5$ |
| 01<br>$z_2$ | 001<br>$w_2$ | – | 011<br>$w_4$ | 101<br>$w_6$ |
| 10<br>$z_3$ | 010<br>$w_3$ | 000<br>$w_1$ | 010<br>$w_3$ | 100<br>$w_5$ |
| 11<br>$z_4$ | – | 001<br>$w_2$ | – | 000<br>$w_1$ |

$y_1y_2y_3$ $y_1y_2y_3$

На основании табл. 43 и используя таблицу переходов $T$-триггера (см. табл. 35) построим табл. 45 − таблицу функций возбуждения элементов памяти.

Если $i$-й триггер на некотором переходе переключается из состояния 0 в состояние 1 или наоборот, то $d_i = 1$, в противном случае (т. е. если $i$-й триггер не переключается) $d_i = 0$. Например, рассмотрим переход из состояния 10 в состояние 11 (см. табл. 43, 4-й столбец, 3-я строка). Первый триггер, установленный в 1, не меняет своего значения, поэтому функция возбуждения элемента памяти для него $d_1 = 0$. Второй триггер изменяет свое значение с 0 на 1, следовательно, $d_2 = 1$. Остальные клетки табл. 44 заполняются аналогично.

Таблица 45

| $\tau_1\tau_2$<br>$x_1x_2$ | 00<br>$a_1$ | 01<br>$a_2$ | 10<br>$a_3$ | 11<br>$a_4$ |
|---|---|---|---|---|
| 00<br>$z_1$ | 01 | 00 | – | 00 |
| 01<br>$z_2$ | 11 | – | 11 | 10 |
| 10<br>$z_3$ | 10 | 11 | 01 | 01 |
| 11<br>$z_4$ | – | 10 | – | 11 |

$d_1d_2$ $d_1d_2$

На основании полученных табл. 44 и 45, которые можно рассматривать как таблицы истинности, запишем систему булевых функций для построения комбинационной схемы автомата. Функции представлены в форме СДНФ:

$$d_1 = \bar{\tau}_1\bar{\tau}_2\bar{x}_1 x_2 \vee \bar{\tau}_1\bar{\tau}_2 x_1\bar{x}_2 \vee \bar{\tau}_1\tau_2 x_1 x_2 \vee \tau_1\bar{\tau}_2\bar{x}_1 x_2 \vee \tau_1\tau_2\bar{x}_1 x_2 \vee \tau_1\tau_2 x_1 x_2;$$

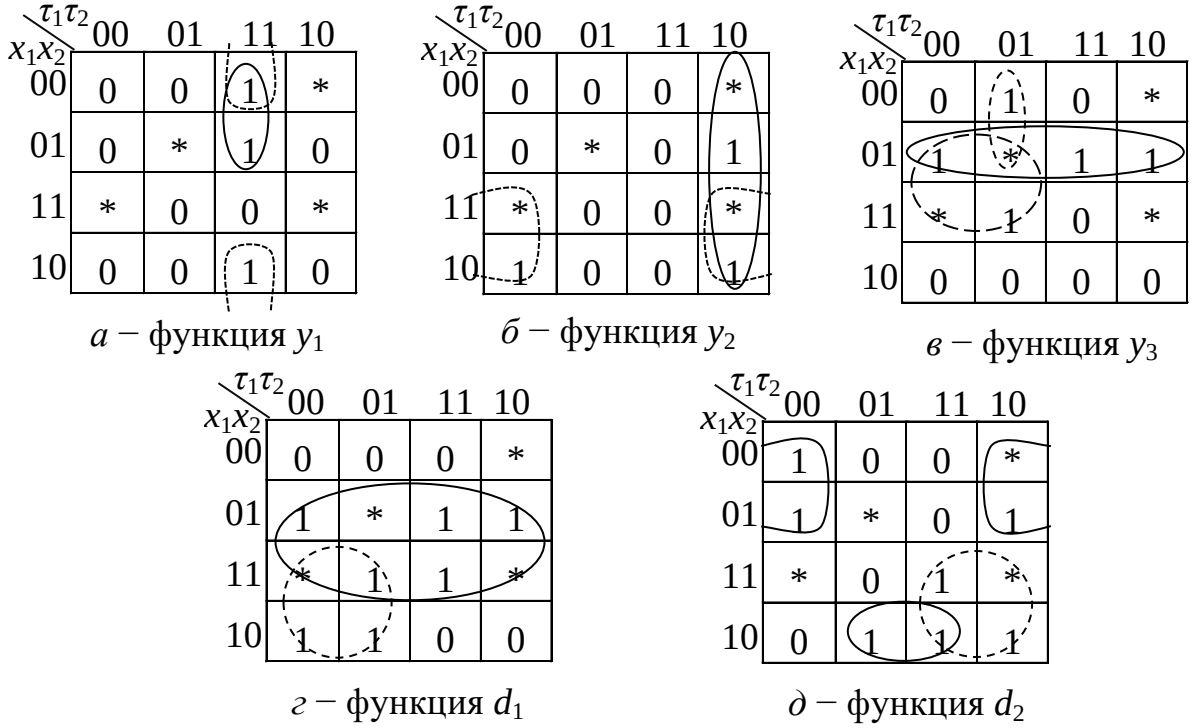
$$d_2 = \bar{\tau}_1\bar{\tau}_2\bar{x}_1\bar{x}_2 \vee \bar{\tau}_1\bar{\tau}_2 x_1 x_2 \vee \bar{\tau}_1\tau_2 x_1\bar{x}_2 \vee \tau_1\bar{\tau}_2\bar{x}_1 x_2 \vee \tau_1\bar{\tau}_2 x_1\bar{x}_2 \vee \tau_1\tau_2 x_1\bar{x}_2 \vee \tau_1\tau_2 x_1 x_2;$$

$$y_1 = \tau_1\tau_2\bar{x}_1\bar{x}_2 \vee \tau_1\tau_2\bar{x}_1 x_2 \vee \tau_1\tau_2 x_1\bar{x}_2;$$

$$y_2 = \bar{\tau}_1\bar{\tau}_2 x_1\bar{x}_2 \vee \tau_1\bar{\tau}_2\bar{x}_1 x_2 \vee \tau_1\bar{\tau}_2 x_1\bar{x}_2;$$

$$y_3 = \bar{\tau}_1\bar{\tau}_2\bar{x}_1 x_2 \vee \bar{\tau}_1\tau_2\bar{x}_1\bar{x}_2 \vee \bar{\tau}_1\tau_2 x_1 x_2 \vee \tau_1\bar{\tau}_2\bar{x}_1 x_2 \vee \tau_1\tau_2\bar{x}_1 x_2.$$

Для упрощения комбинационной схемы выполним минимизацию каждой из булевых функций. Для этого используем метод минимизирующих карт Карно. Обратим внимание на то, что в некоторых клетках табл. 44 и 45 есть прочерки. Это значит, например, что в состоянии $a_1$ ($\bar{\tau}_1\bar{\tau}_2$) на вход автомата не может поступить значение $x_1x_2$. Следовательно функции на наборе $\bar{\tau}_1\bar{\tau}_2 x_1 x_2$, будут неопределены. На рис. 70 изображены пять карт Карно для минимизации каждой из пяти ($y_1$, $y_2$, $y_3$, $d_1$ и $d_2$) булевых функций. В клетках карт, соответствующих наборам, на которых функции не определены, стоят символы звездочки «*».




*а* – функция $y_1$ *б* – функция $y_2$ *в* – функция $y_3$

*г* – функция $d_1$ *д* – функция $d_2$

Рис. 70. Карты Карно для минимизации булевых функций

По результатам минимизации запишем систему минимальных функций:

$$d_1 = x_2 \vee \bar{\tau}_1 x_1;$$

$$d_2 = \bar{\tau}_2 \bar{x}_1 \vee \tau_1 x_1 \vee \tau_2 x_1 \bar{x}_2;$$

$$y_1 = \tau_1 \tau_2 \bar{x}_1 \vee \tau_1 \tau_2 \bar{x}_2;$$

$$y_2 = \tau_1 \bar{\tau}_2 \vee \bar{\tau}_2 x_1;$$

$$y_3 = \bar{x}_1 x_2 \vee \bar{\tau}_1 x_2 \vee \bar{\tau}_1 \tau_2 \bar{x}_1.$$

На рис. 71 изображена логическая схема, построенная на основании полученной системы булевых функций. При построении схемы использованы элементы И и ИЛИ.

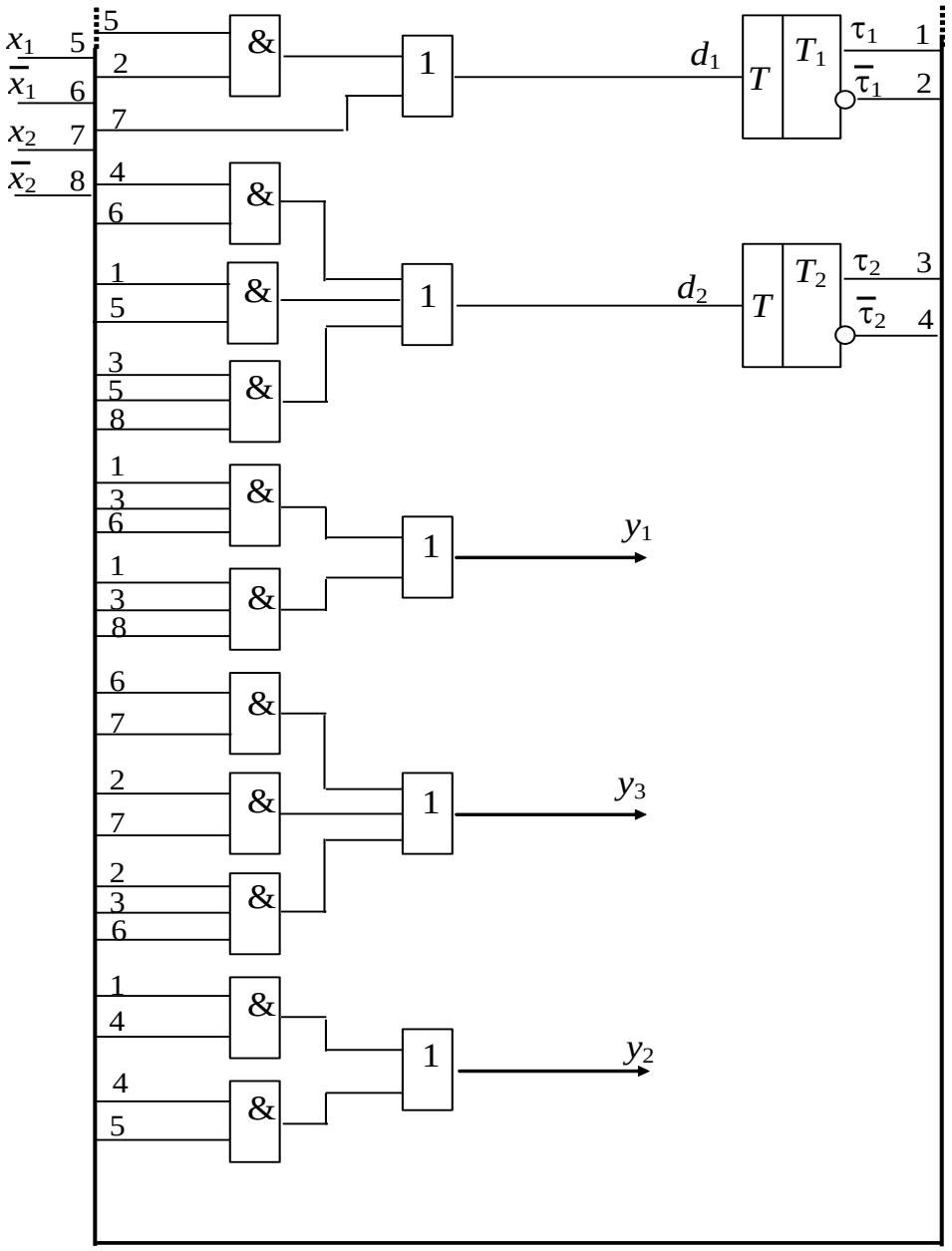



Рис. 71. Логическая схема автомата Мили

### 5.5.2. Канонический метод структурного синтеза автомата Мура

Далее рассмотрим канонический метод на примере синтеза структурного автомата Мура, блок памяти которого построим на *RS*-триггерах. Исходные данные для выполнения синтеза структурной схемы заданы совмещенной таблицей переходов − выходов (табл. 46). Закон функционирования *RS*-триггера приведен в табл. 36.

Таблица 46

| $\lambda$ | $w_1$ | $w_2$ | $w_3$ | $w_3$ |
|---|---|---|---|---|
| $\delta$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ |
| $z_1$ | $a_2$ | $a_3$ | $a_2$ | – |
| $z_2$ | $a_4$ | – | $a_3$ | $a_2$ |
| $z_3$ | – | $a_2$ | $a_4$ | $a_1$ |
| $z_4$ | $a_3$ | $a_3$ | – | $a_4$ |

Как и в предыдущем примере вначале определим общее количество входов, выходов и элементов памяти структурного автомата: $L = 2$, $N = 2$, $R = 2$.

Структурная схема автомата изображена на рис. 72.

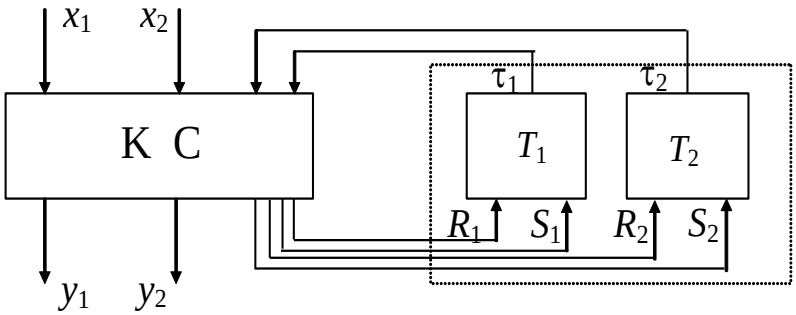



Рис. 72. Структурная схема автомата Мура

Как и при синтезе автомата Мили выполним кодирование входного, выходного алфавита и внутренних состояний автомата (табл. 47−49).

Таблица 47

| $Z$ \ $X$ | $x_1$ | $x_2$ |
|---|---|---|
| $z_1$ | 0 | 0 |
| $z_2$ | 0 | 1 |
| $z_3$ | 1 | 0 |
| $z_4$ | 1 | 1 |

Таблица 48

| $W$ \ $Y$ | $y_1$ | $y_2$ |
|---|---|---|
| $w_1$ | 0 | 0 |
| $w_2$ | 0 | 1 |
| $w_3$ | 1 | 0 |
| $w_4$ | 1 | 1 |

Таблица 49

| $A$ \ $T$ | $\tau_1$ | $\tau_2$ |
|---|---|---|
| $a_1$ | 0 | 0 |
| $a_2$ | 0 | 1 |
| $a_3$ | 1 | 0 |
| $a_4$ | 1 | 1 |

Результаты кодирования занесем в исходную таблицу переходов − выходов (см. табл. 46) и построим табл. 50. Внесем изменения в значения кодов состояний переходов в соответствии с принципом работы *RS*-триггера (см. табл. 35). Построим новую, перекодированную, таблицу переходов (табл. 51). При ее построении учитываем следующее. Для переключения *i*-го триггера на некотором переходе из

состояния 0 в состояние 1 на его входы (*RS*) надо подать код 01, а в состояние 0 соответсвенно -0. Для переключения из состояния 1 в состояние 1 на триггер подается код 0-, а в состояние 0 соответсвенно 10. Например, рассмотрим переход из состояния 10 в состояние 11 (см. табл. 50, 4-й столбец, 5-я строка). Первый триггер не изменяет своего значения. На входы первого триггера (установленного в 1) подается 0-. Второй триггер должен изменить свое значение с 0 на 1. На его входы для этого подается 01. Остальные клетки табл. 51 заполняются аналогично.

Таблица 50

| $y_1y_2$ | 00 | 01 | 01 | 10 |
|---|---|---|---|---|
| | $w_1$ | $w_2$ | $w_2$ | $w_3$ |
| $\tau_1\tau_2$ $x_1x_2$ | 00 $a_1$ | 01 $a_2$ | 10 $a_3$ | 11 $a_4$ |
| 00 $z_1$ | 01 | 10 | 01 | – |
| 01 $z_2$ | 11 | – | 10 | 01 |
| 10 $z_3$ | – | 01 | 11 | 00 |
| 11 $z_4$ | 10 | 10 | – | 11 |

Таблица 51

| $y_1y_2$ | 00 | 01 | 10 | 10 |
|---|---|---|---|---|
| | $w_1$ | $w_2$ | $w_3$ | $w_3$ |
| $\tau_1\tau_2$ $x_1x_2$ | 00 $a_1$ | 01 $a_2$ | 10 $a_3$ | 11 $a_4$ |
| 00 $z_1$ | -001 | 0110 | 1001 | – |
| 01 $z_2$ | 0101 | – | 0--0 | 100- |
| 10 $z_3$ | – | -00- | 0-01 | 1010 |
| 11 $z_4$ | 01-0 | 0110 | – | 0-0- |

На основании табл. 51 запишем систему булевых функций для построения комбинационной схемы автомата:

$$R_1 = \bar{\tau}_1\bar{\tau}_2\bar{x}_1\bar{x}_2 \vee \tau_1\tau_2\bar{x}_1x_2 \vee \tau_1\tau_2x_1\bar{x}_2;$$

$$S_1 = \bar{\tau}_1\bar{\tau}_2\bar{x}_1x_2 \vee \bar{\tau}_1\bar{\tau}_2x_1x_2 \vee \bar{\tau}_1\tau_2\bar{x}_1\bar{x}_2 \vee \bar{\tau}_1\tau_2x_1x_2;$$

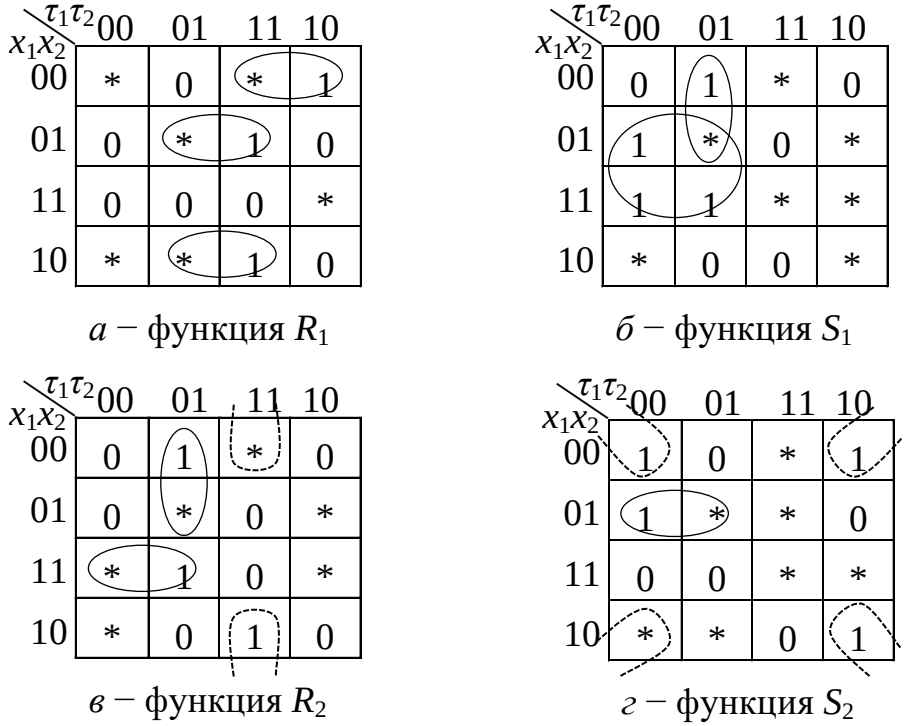
$$R_2 = \bar{\tau}_1\tau_2\bar{x}_1\bar{x}_2 \vee \bar{\tau}_1\tau_2x_1x_2 \vee \tau_1\tau_2x_1\bar{x}_2;$$

$$S_2 = \bar{\tau}_1\bar{\tau}_2\bar{x}_1\bar{x}_2 \vee \bar{\tau}_1\bar{\tau}_2x_1x_2 \vee \tau_1\bar{\tau}_2\bar{x}_1\bar{x}_2 \vee \tau_1\bar{\tau}_2x_1\bar{x}_2;$$

$$y_1 = \tau_1\bar{\tau}_2 \vee \tau_1\tau_2;$$

$$y_2 = \bar{\tau}_1\tau_2.$$

Для упрощения комбинационной схемы выполним минимизацию каждой из булевых функций. Для этого используем метод минимизирующих карт Карно. На рис. 73 изображены четыре карты Карно для минимизации каждой из шести ($y_1$, $y_2$, $R_1$, $S_1$, $R_2$ и $S_2$) булевых функций. Как и в примере синтеза автомата Мили, в карты Карно внесены символы * в клетки, соответствующие наборам входных значений, на которых функции не определены.




*а* − функция $R_1$ *б* − функция $S_1$

*в* − функция $R_2$ *г* − функция $S_2$

Рис. 73. Карты Карно для минимизации булевых функций

По результатам минимизации запишем систему минимальных функций:

$$R_1 = \tau_1 \bar{x}_1 \bar{x}_2 \vee \tau_2 \bar{x}_1 x_2 \vee \tau_2 x_1 \bar{x}_2;$$

$$S_1 = \bar{\tau}_1 x_2 \vee \bar{\tau}_1 \tau_2 \bar{x}_1;$$

$$R_2 = \bar{\tau}_1 \tau_2 \bar{x}_1 \vee \bar{\tau}_1 x_1 x_2 \vee \tau_1 \tau_2 \bar{x}_2;$$

$$S_2 = \bar{\tau}_1 \bar{x}_1 x_2 \vee \bar{\tau}_2 \bar{x}_2;$$

$$y_1 = \tau_1;$$

$$y_2 = \bar{\tau}_1 \tau_2 .$$

На рис. 74 изображена логическая схема, построенная на основании полученной системы булевых функций. При построении схемы использованы элементы И и ИЛИ.

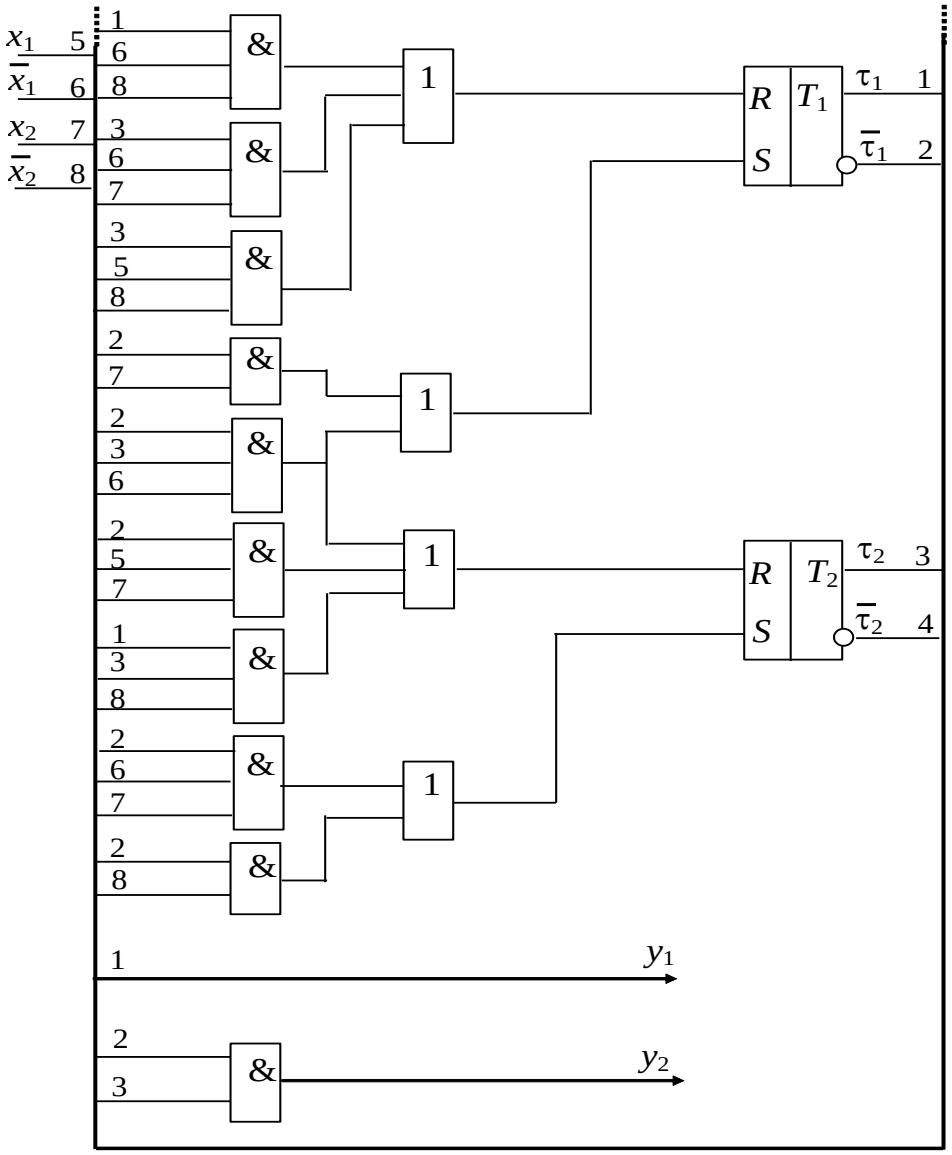



Рис. 74. Логическая схема автомата Мура

**5.6. Понятие операционного и управляющих автоматов.**

Как показал академик В. М. Глушков [12] в любом устройстве обработки цифровой информации можно выделить два основных блока – операционный автомат (ОА) и управляющий автомат (УА). В общем случае вычислительное устройство, имеет структуру, представленную на рис. 75.

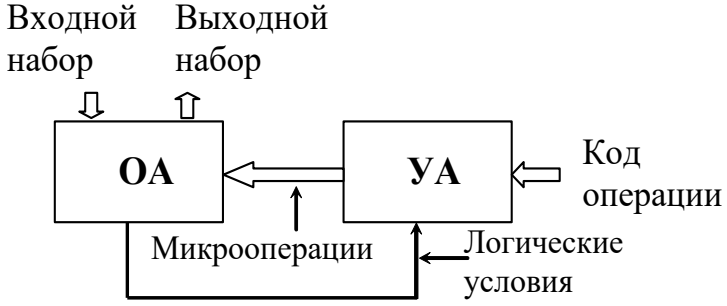



Рис. 75. Структура вычислительного устройства

Операционный автомат служит для хранения информации, выполнения набора микроопераций и вычисления значений логических условий, т. е. ОА предназначен для выполнения действий над информацией. Микрооперации, выполняемые ОА, задаются множеством управляющих сигналов $Y = \{y_1, ..., y_N\}$, с каждым из которых отождествляется определенная микрооперация.

Значения логических условий, вычисляемых в ОА, определяются множеством сигналов $X = \{x_1, ..., x_L\}$, соответствующих определенным логическим условиям.

УА генерирует последовательность управляющих сигналов, определяемую микропрограммой в соответствии с логическими условиями. Иначе говоря, УА задает порядок выполнения действий в ОА.

Таким образом любое устройство вычислительной техники является композицией ОА и УА. При этом ОА, реализуя действия над информацией, является исполнительной частью вычислительного устройства, работой которого управляет УА, генерирующий необходимые последовательности управляющих сигналов в определенном порядке и времени.

Функция УА – это операторная схема алгоритма (микропрограммы). Операторная схема алгоритма часто представляется в виде граф-схемы алгоритма (ГСА) [17, 19]. ГСА определяет вычислительный процесс последовательно во времени, устанавливая порядок проверки логических условий $X$ и порядок следования микрокоманд $Y$.

### 5.7. Принцип микропрограммного управления

Устройства вычислительной техники перерабатывают информацию, выполняя над ней какие-либо операции. Функцией ОА является выполнение заданного множества операций над входными наборами с целью формирования выходных наборов, которые представляют собой результаты операций.

Функциональная и структурная организация ОА базируется на принципе микропрограммного управления [17, 19], который состоит в следующем:

1. Любая операция, реализуемая устройством, рассматривается как сложное действие, которое разделяется на последовательность элементарных действий, выполняемых за один такт его работы. Эти элементарные действия называются ***микрооперациями***.

2. Для управления порядком следования микроопераций используются ***логические условия***. Проверка значений логических условий в каждом такте работы автомата позволяет определить группу выполняемых микроопераций.

3. Совокупность операций, выполняемых за один такт работы автомата, называется ***микрокомандой***.

4. Процесс выполнения микрокоманд в устройстве описывается в форме алгоритма, который представляется в терминах микроопераций (микрокоманд) и логических условий и называется ***микропрограммой***. Микропрограмма определяет порядок проверки логических условий и выполняемых микроопераций (микрокоманд) для получения требуемых результатов.

5. Принцип, согласно которому алгоритм работы некоторого устройства описывается в перечисленных выше терминах, называется принципом микропрограммного управления.

Таким образом, из принципа микропрограммного управления следует, что структура и порядок функционирования операционных устройств определяется алгоритмом выполнения операции.

К элементарным действиям над словами информации (микрооперациям) относятся: *передача информации из одного регистра в другой, взятие обратного кода, сдвиг* и т. д.

Конечный автомат, алгоритм работы которого может быть описан на основе принципа микропрограммного управления, называется микропрограммным автоматом (МПА).

### 5.8. Граф-схема алгоритма

Для записи микропрограмм в компактной форме используются специализированные языки. Одним из способов графического представления микропрограммы является ГСА [17, 19]. ГСА представляет собой ориентированный связный граф. Кроме наглядности это дает возможность использовать для анализа и преобразования микропрограмм эффективные методы теории графов. При графическом описании отдельные функции алгоритмов (микрооперации) описываются в виде условных графических изображений – вершин. ГСА может содержать вершины четырех типов: начальную, операторную, условную и конечную (рис. 76):

− начальная вершина (см. рис. 76,*а*), имеет один выход, входов не имеет; обозначает начало микропрограммы;

− операторная вершина (см. рис. 76,*б*) имеет один вход и один выход; внутри операторной вершины записывается одна микрокоманда;

− условная вершина (см. рис. 76,*в*) имеет один вход и два выхода; внутри условной вершины записывается логическое условие, осуществляющее выбор направления дальнейшего выполнения микропрограммы;

− конечная вершина (см. рис. 76,*г*) имеет один вход, выходов не имеет; обозначает конец микропрограммы.

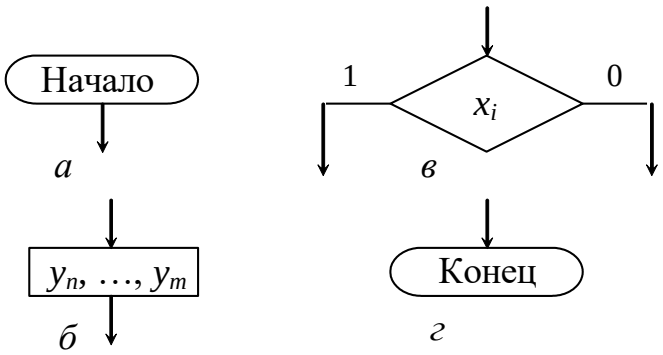



Рис. 76. Типы вершин ГСА

Пример ГСА представлен на рис. 77 и 78. Если внутри вершин в явном виде записаны микрооперации и логические условия, то ГСА называется содержа-

тельной (см. рис. 77). Если же каждую микрооперацию обозначить символами $y_i$, а логические условия через $x_i$, то ГСА будет кодированная (см. рис. 78).

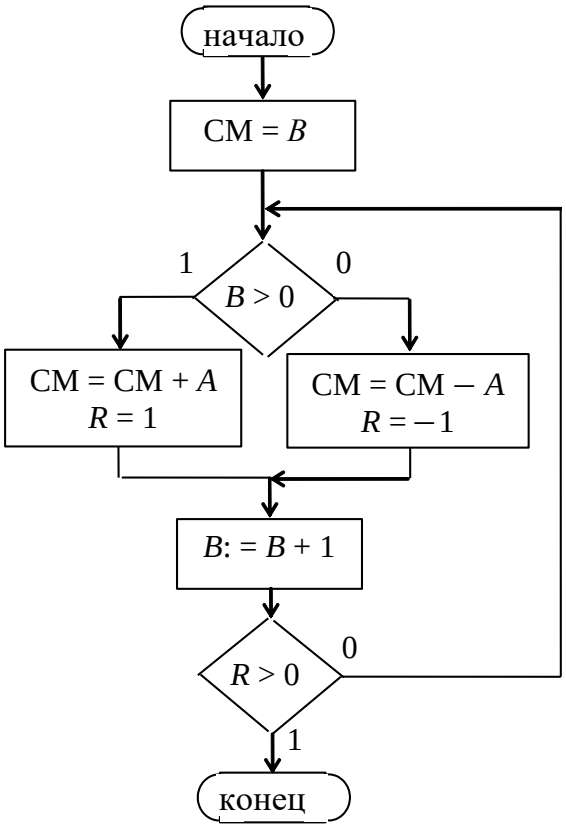



Рис. 77. Содержательная ГСА

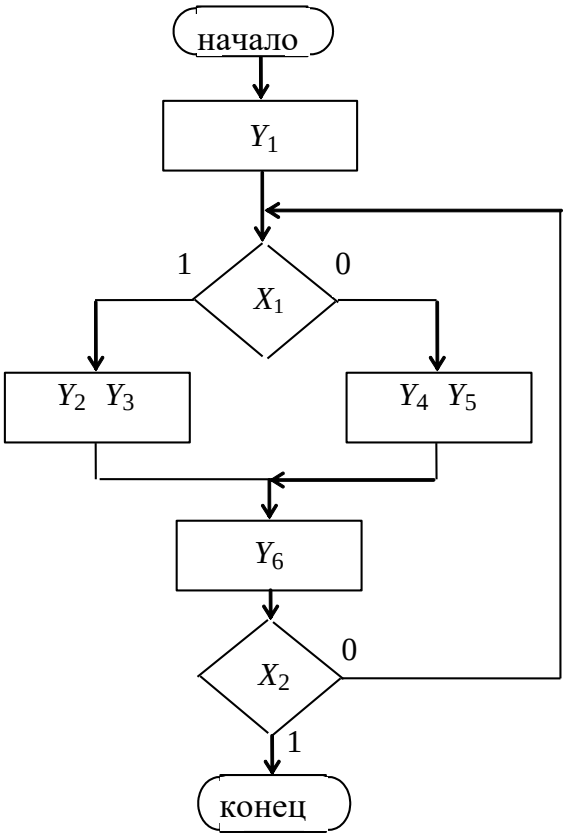



Рис. 78. Кодированная ГСА

ГСА должна удовлетворять следующим основным требованиям:

- в ГСА имеются одна начальная и одна конечная вершины;
- входы и выходы вершин соединяются с помощью дуг;
- каждая вершина должна лежать на одном из путей следования из начальной вершины в конечную;
- один из выходов условной вершины может соединяться с ее входом (ждущая вершина);
- в каждой условной вершине записывается одно из логических условий $x_i$ (допускается запись одинаковых условий в различных вершинах);
- в каждой операторной вершине записывается микрокоманда (допускается пустая микрокроманда и повтор микрокоманды в различных вершинах).

### 5.9. Синтез МПА по ГСА

МПА может быть синтезирован по ГСА, описывающей микропрограмму работы проектируемого дискретного устройства [17, 19].

Алгоритм синтеза МПА по ГСА состоит в следующем:

1) разметка ГСА для синтеза МПА Мили (Мура);
2) кодирование внутренних состояний;
3) построение структурной таблицы по отмеченной ГСА;
4) формирование таблиц истинности или системы булевых функций;

5) построение логической схемы автомата;
6) конец алгоритма.

Как отмечалось выше, известны два класса автоматов: Мили и Мура. В качестве примера рассмотрим синтез микропрограммного автомата, управляющего операционным автоматом, реализующим операцию деления чисел в дополнительных кодах.

### 5.9.1. Синтез МПА Мили по ГСА

ГСА, соответствующая алгоритму деления, изображена на рис. 79. Описание алгоритма деления чисел в дополнительном коде приведено в п 2.21.2.

На приведенном рисунке в изображении содержательной ГСА использованы следующие обозначения: См – сумматор, Зн См – знак сумматора, Чт – частное, Ст – счетчик тактов, Дт – делитель, Зн Дт – знак делителя, $L_1$См ($L_1$Чт) – сдвиг влево на один разряд в сумматоре (в частном), ТП – переполнение.

В последующих тактах Зн См может быть равен нулю. Это означает, что остаток $A_i >$ Дт, но Ст уже содержит ненулевое значение, и алгоритм выполняется по стрелке 4. Если Зн См равен единице, то остаток отрицательный и деление будет выполняться в направлении стрелки 3.

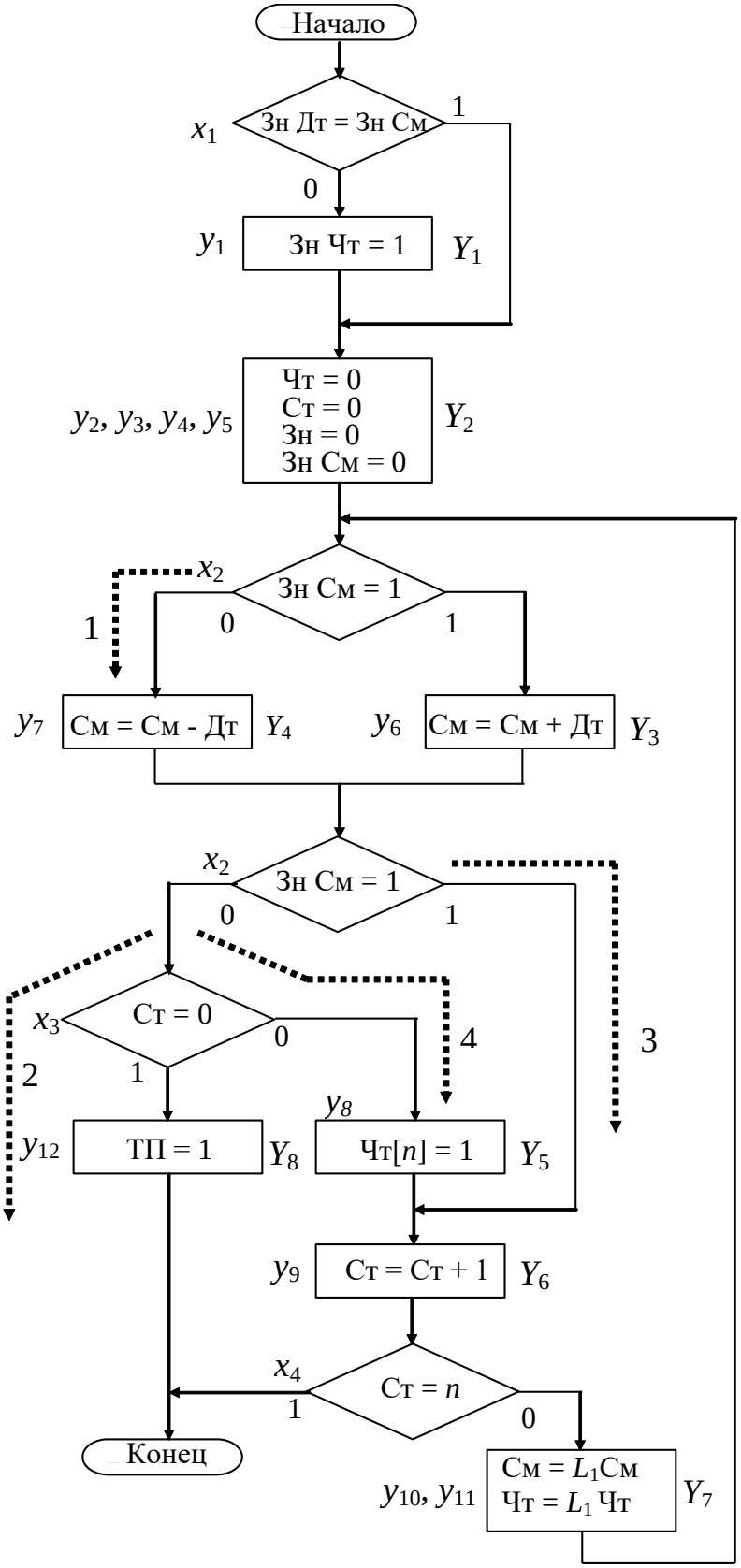



Рис. 79. ГСА выполнения операции деления чисел

Для получения графа автомата Мили исходная ГСА отмечается метками. Каждой метке на ГСА ставится во взаимно однозначное соответствие состояние

автомата. Отметка ГСА при синтезе автомата Мили метками состоит в следующем:

- выход начальной и вход конечной вершин отмечаются меткой $a_1$;
- входы всех вершин, следующих за операторными, отмечаются метками $a_2$, …, $a_m$;
- одной меткой может быть отмечен только один вход.

На рис. 80 приведена ГСА, отмеченная метками.

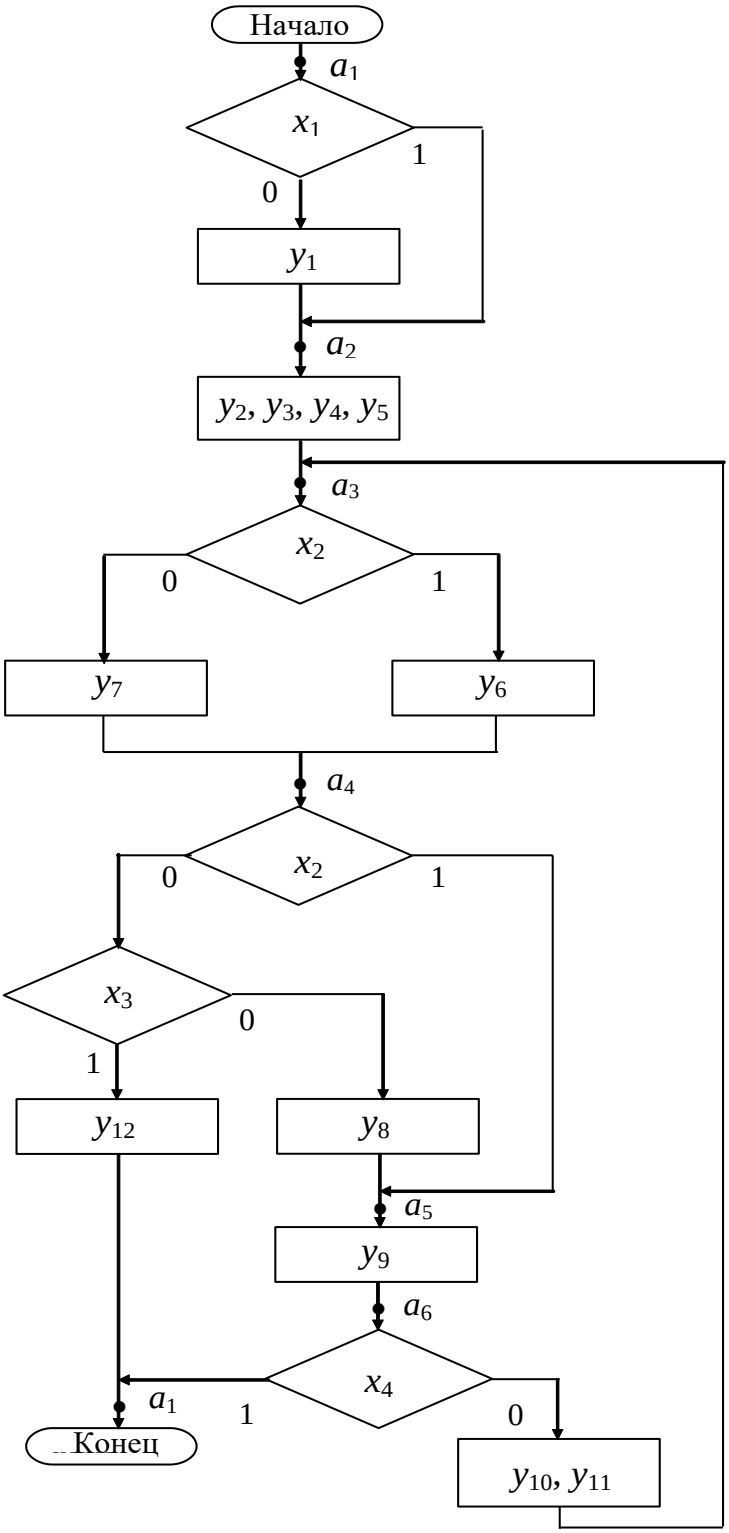



Рис. 80. Отмеченная ГСА при синтезе МПА Мили

Кодирование состояний автомата может быть выполнено, как и ранее (т. е. каждому состоянию ставится в соответствие двоичный эквивалент десятичного номера состояния). Для нахождения всевозможных переходов автомата на отмеченной ГСА отыскиваются все пути вида

$a_m x_i, ..., x_j Y_k\, a_s$ или $a_m X(a_m, a_s) Y_k\, a_s$.

При достаточно большом числе состояний и переходов удобным является представление автомата структурной таблицей, содержащей всю необходимую для синтеза информацию. Структурная таблица может быть прямой или обратной. В *прямой* таблице (табл. 52) вначале записываются все переходы из со стояния $a_1$, затем из состояния $a_2$ и т. д. В *обратной* таблице сначала записываются все переходы в состояние $a_1$, затем в $a_2$ и т. д.

Таблица 52

| Исходное состояние | Код исходного состояния | Состояние перехода | Код состояния перехода | Входной сигнал | Выходной сигнал | Функции возбуждения |
|---|---|---|---|---|---|---|
| $a_m$ | $K(a_m)$ | $a_s$ | $K(a_s)$ | $X(a_m, a_s)$ | $Y(a_m, a_s)$ | $F(a_m, a_s)$ |
| $a_1$ | 000 | $a_2$ | 001 | $\overline{x}_1$ | $y_1$ | $S_3$ |
| | | $a_3$ | 010 | $x_1$ | $y_2\, y_3\, y_4\, y_5$ | $S_2$ |
| $a_2$ | 001 | $a_3$ | 010 | 1 | $y_2\, y_3\, y_4\, y_5$ | $S_2\, R_3$ |
| $a_3$ | 010 | $a_4$ | 011 | $x_2$ | $y_6$ | $S_3$ |
| | | $a_4$ | 011 | $\overline{x}_2$ | $y_7$ | $S_3$ |
| $a_4$ | 011 | $a_1$ | 000 | $\overline{x}_2 x_3$ | $y_{12}$ | $R_2\, R_3$ |
| | | $a_5$ | 100 | $\overline{x}_2 \overline{x}_3$ | $y_8$ | $S_1\, R_2\, R_3$ |
| | | $a_6$ | 101 | $x_2$ | $y_9$ | $S_1\, R_2$ |
| $a_5$ | 100 | $a_6$ | 101 | 1 | $y_9$ | $S_3$ |
| $a_6$ | 101 | $a_1$ | 000 | $x_4$ | – | $R_1\, R_3$ |
| | | $a_3$ | 010 | $\overline{x}_4$ | $y_{10}\ y_{11}$ | $R_1\, S_2\, R_3$ |

Для реализации блока памяти синтезируемого МПА Мили использованы *RS*-триггеры. В последнем столбце $F(a_m, a_s)$ структурной таблицы отмечены те функции возбуждения, которые приводят к изменению содержимого каждого из элементов памяти на соответствующем переходе.

Для построения схемы, реализующей синтезируемый МПА Мили, удобно результаты, приведенные в структурной таблице (см. табл. 52), а также представить их в виде таблицы истинности (табл. 53).

Таблица 53

| № | $x_1x_2x_3x_4\tau_1\tau_2\tau_3$ | $y_1y_2y_3y_4y_5y_6y_7y_8y_9y_{10}y_{11}y_{12}$ | $S_1R_1S_2R_2S_3R_3$ |
|---|---|---|---|
| 0 | 0 - - - 0 0 0 | 1 0 0 0 0 0 0 0 0 0 0 0 | 0 0 0 0 1 0 |
| 1 | 1 - - - 0 0 0 | 0 1 1 1 1 0 0 0 0 0 0 0 | 0 0 1 0 0 0 |
| 2 | - - - - 0 0 1 | 0 1 1 1 1 0 0 0 0 0 0 0 | 0 0 1 0 0 1 |
| 3 | - 1 - - 0 1 0 | 0 0 0 0 0 1 0 0 0 0 0 0 | 0 0 0 0 1 0 |
| 4 | - 0 - - 0 1 0 | 0 0 0 0 0 0 1 0 0 0 0 0 | 0 0 0 0 1 0 |
| 5 | - 0 1 - 0 1 1 | 0 0 0 0 0 0 0 0 0 0 0 1 | 0 0 0 1 0 1 |
| 6 | - 0 0 - 0 1 1 | 0 0 0 0 0 0 0 1 0 0 0 0 | 1 0 0 1 0 1 |
| 7 | - 1 - - 0 1 1 | 0 0 0 0 0 0 0 0 1 0 0 0 | 1 0 0 1 0 0 |
| 8 | - - - - 1 0 0 | 0 0 0 0 0 0 0 0 1 0 0 0 | 0 0 0 0 1 0 |
| 9 | - - - 1 1 0 1 | 0 0 0 0 0 0 0 0 0 0 0 0 | 0 1 0 0 0 1 |
| 10 | - - - 0 1 0 1 | 0 0 0 0 0 0 0 0 0 1 1 0 | 0 1 1 0 0 1 |

Для примера реализации логической схемы синтезируемого МПА Мили рассмотрим реализацию функций $y_1y_2$ и $S_2R_2$ (рис. 81).

$$y_1 = \bar{x}_1\bar{\tau}_1\bar{\tau}_2\bar{\tau}_3;$$

$$y_2 = x_1\bar{\tau}_1\bar{\tau}_2\bar{\tau}_3 \vee \bar{\tau}_1\bar{\tau}_2\tau_3;$$

$$S_2 = x_1\bar{\tau}_1\bar{\tau}_2\bar{\tau}_3 \vee \bar{\tau}_1\bar{\tau}_2\tau_3 \vee \bar{x}_4\tau_1\bar{\tau}_2\tau_3;$$

$$R_2 = \bar{x}_2x_3\bar{\tau}_1\tau_2\tau_3 \vee \bar{x}_2\bar{x}_3\bar{\tau}_1\tau_2\tau_3 \vee x_2\bar{\tau}_1\tau_2\tau_3 = \bar{\tau}_1\tau_2\tau_3.$$

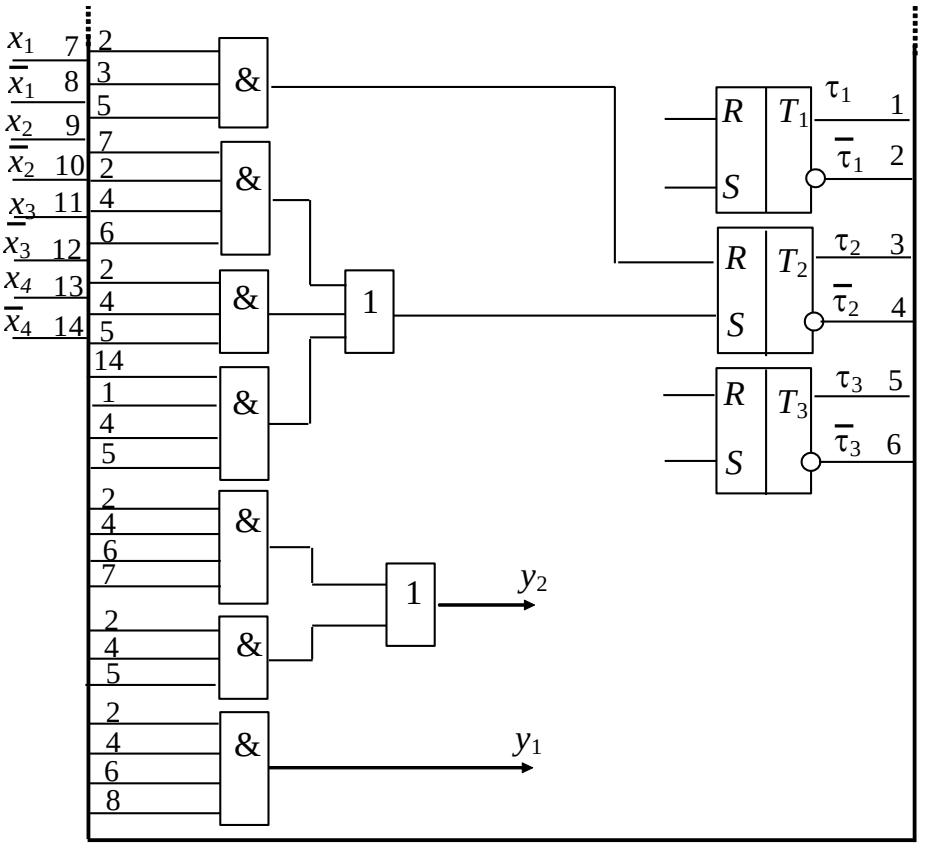



Рис. 81. Логическая схема автомата Мили

### 5.9.2. Синтез МПА Мура по ГСА

Для наглядного сравнения двух методов синтеза (автоматов Мили и Мура) синтез автомата Мура выполним для алгоритма, представленного на ГСА, изображенной на рис. 79. Как и ранее, для получения графа автомата исходная ГСА отмечается метками. Отметка ГСА для синтеза МПА Мура состоит в следующем:

- начальная и конечная вершины ГСА отмечаются меткой $a_1$;
- метками $a_2$, …, $a_m$ отмечаются все операторные вершины;
- одной меткой может быть отмечена только одна вершина ГСА.

На рис. 82 приведена отмеченная ГСА.

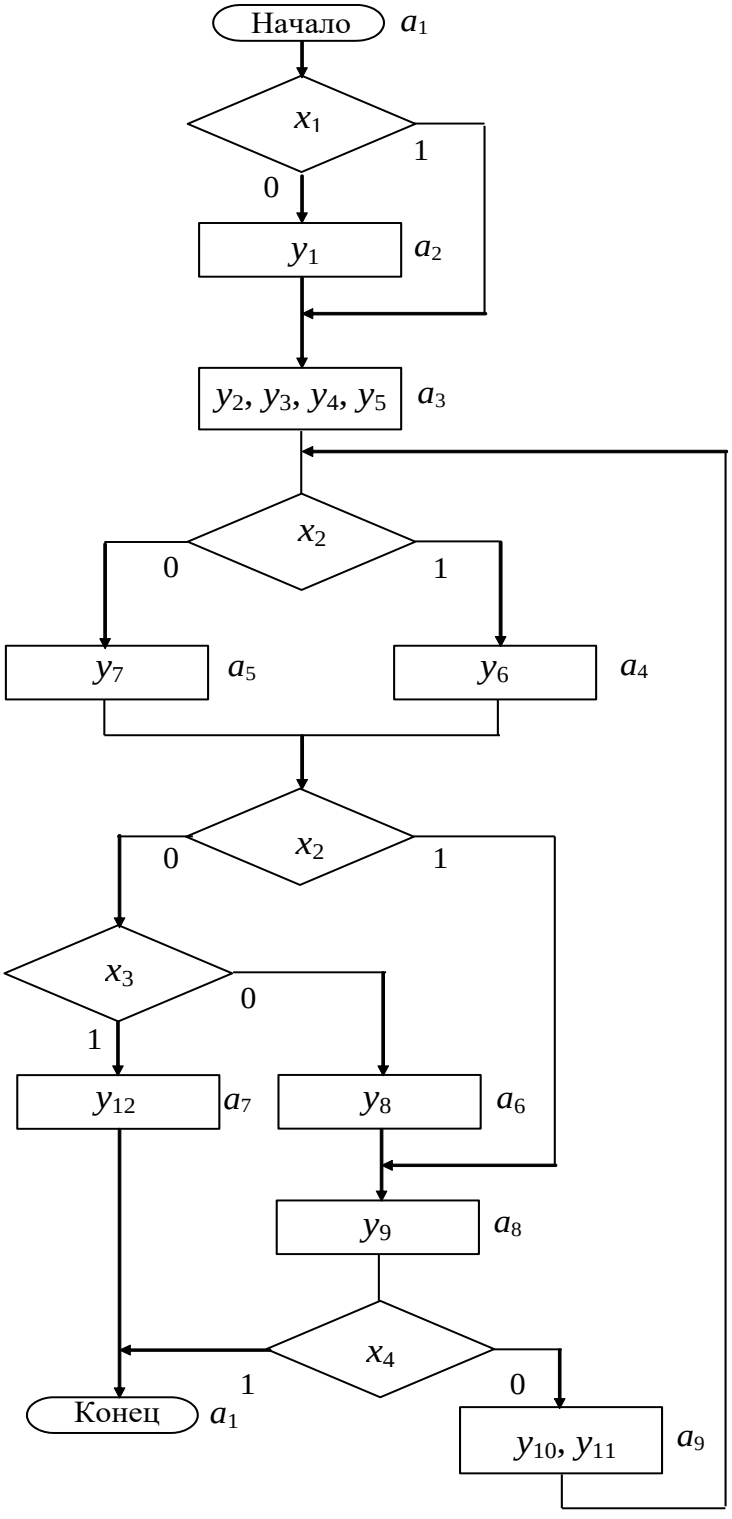



Рис. 82. Отмеченная ГСА при синтезе МПА Мура

Кодирование состояний автомата Мура выполним как и ранее в случае синтеза автомата Мили, поставив каждому состоянию в соответствие двоичный эквивалент номера состояния. Для нахождения всевозможных переходов автомата на отмеченной ГСА отыскиваются все пути вида:

$a_m x_i, ..., x_j a_s$ или $a_m X(a_m, a_s) a_s$.

Как и в предыдущем примере синтеза автомата Мили, при синтезе автомата Мура, для представления автомата, используем прямую структурную таблицу переходов (табл. 54).

Таблица 54

| Исходное состояние | Код исходного состояния | Состояние перехода | Код состояния перехода | Входной сигнал | Функции возбуждения |
|---|---|---|---|---|---|
| $a_m$, $Y(a_m)$ | $K(a_m)$ | $a_s$ | $K(a_s)$ | $X(a_m, a_s)$ | $F(a_m, a_s)$ |
| $a_1$, – | 0000 | $a_2$ | 0001 | $\overline{x_1}$ | $\overline{J_1}\,\overline{J_2}\,\overline{J_3}\,J_4$ |
| | | $a_3$ | 0010 | $x_1$ | $\overline{J_1}\,\overline{J_2}\,J_3\,\overline{J_4}$ |
| $a_2$, $y_1$ | 0001 | $a_3$ | 0010 | 1 | $\overline{J_1}\,\overline{J_2}\,J_3\,K_4$ |
| $a_3$, $y_2\,y_3\,y_4\,y_5$ | 0010 | $a_5$ | 0100 | $\overline{x_2}$ | $\overline{J_1}\,J_2\,K_3\,\overline{J_4}$ |
| | | $a_4$ | 0011 | $x_2$ | $\overline{J_1}\,\overline{J_2}\,\overline{K_3}\,J_4$ |
| $a_4$, $y_7$ | 0011 | $a_6$ | 0101 | $\overline{x_2}\overline{x_3}$ | $\overline{J_1}\,J_2\,K_3\,\overline{K_4}$ |
| | | $a_8$ | 0111 | $x_2$ | $\overline{J_1}\,J_2\,\overline{K_3}\,\overline{K_4}$ |
| | | $a_7$ | 0110 | $\overline{x_2}x_3$ | $\overline{J_1}\,J_2\,\overline{K_3}\,K_4$ |
| $a_5$, $y_6$ | 0100 | $a_6$ | 0101 | $\overline{x_2}\overline{x_3}$ | $\overline{J_1}\,\overline{K_2}\,\overline{J_3}\,J_4$ |
| | | $a_8$ | 0111 | $x_2$ | $\overline{J_1}\,\overline{K_2}\,J_3\,J_4$ |
| | | $a_7$ | 0110 | $\overline{x_2}x_3$ | $\overline{J_1}\,\overline{K_2}\,J_3\,\overline{J_4}$ |
| $a_6$, $y_8$ | 0101 | $a_8$ | 0111 | 1 | $\overline{J_1}\,\overline{K_2}\,J_3\,\overline{K_4}$ |
| $a_7$, $y_{12}$ | 0110 | $a_1$ | 0000 | 1 | $\overline{J_1}\;K_2\,K_3\,\overline{J_4}$ |
| $a_8$, $y_9$ | 0111 | $a_1$ | 0000 | $x_4$ | $\overline{J_1}\,K_2\,K_3\,K_4$ |
| | | $a_9$ | 1000 | $\overline{x_4}$ | $J_1\,K_2\,K_3\,K_4$ |
| $a_9$, $y_9\;y_{11}$ | 1000 | $a_5$ | 0100 | $\overline{x_2}$ | $K_1\,J_2\,\overline{J_3}\,\overline{J_4}$ |
| | | $a_4$ | 0011 | $x_2$ | $K_1\overline{J_2}\,J_3\,J_4$ |

Блок памяти МПА реализуем с использованием *JK*-триггеров. В структурной таблице автомата Мура совмещены столбцы $a_m$ и $Y(a_m)$. В столбце $F(a_m\ a_s)$ отмечены функции возбуждения, изменяющие содержимое соответствующего элемента памяти на некотором переходе.

Для удобства построения схемы, реализующей синтезируемый МПА Мура, результаты, приведенные в структурной таблице (см. табл. 54), как и ранее, представим в виде таблицы истинности (табл. 55).

Таблица 55

| № | $x_1x_2x_3x_4\tau_1\tau_2\tau_3\tau_4$ | $y_1y_2y_3y_4y_5y_6y_7y_8y_9y_{10}y_{11}y_{12}$ | $J_1K_1J_2K_2J_3K_3J_4K_4$ |
|---|---|---|---|
| 0 | 0 - - - 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 | 0 * 0 * 0 * 1 * |
| 1 | 1 - - - 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 | 0 * 0 * 1 * 0 * |
| 2 | - - - - 0 0 0 1 | 1 0 0 0 0 0 0 0 0 0 0 0 | 0 * 0 * 1 * * 1 |
| 3 | - 0 - - 0 0 1 0 | 0 1 1 1 1 0 0 0 0 0 0 0 | 0 * 1 * * 1 0 * |
| 4 | - 1 - - 0 0 1 0 | 0 1 1 1 1 0 0 0 0 0 0 0 | 0 * 0 * * 0 1 * |
| 5 | - 0 0 - 0 0 1 1 | 0 0 0 0 0 1 0 0 0 0 0 0 | 0 * 1 * * 1 * 0 |
| 6 | - 1 - - 0 0 1 1 | 0 0 0 0 0 1 0 0 0 0 0 0 | 0 * 1 * * 0 * 0 |
| 7 | - 0 1 - 0 0 1 1 | 0 0 0 0 0 1 0 0 0 0 0 0 | 0 * 1 * * 0 * 1 |
| 8 | - 0 0 - 0 1 0 0 | 0 0 0 0 0 0 1 0 0 0 0 0 | 0 * * 0 0 * 1 * |
| 9 | - 1 - - 0 1 0 0 | 0 0 0 0 0 0 1 0 0 0 0 0 | 0 * * 0 1 * 1 * |
| 10 | - 0 1 - 0 1 0 0 | 0 0 0 0 0 0 1 0 0 0 0 0 | 0 * * 0 1 * 0 * |
| 11 | - - - - 0 1 0 1 | 0 0 0 0 0 0 0 1 0 0 0 0 | 0 * * 0 1 * * 0 |
| 12 | - - - - 0 1 1 0 | 0 0 0 0 0 0 0 0 0 0 0 1 | 0 * * 1 * 1 0 * |
| 13 | - - - 1 0 1 1 1 | 0 0 0 0 0 0 0 0 1 0 0 0 | 0 * * 1 * 1 * 1 |
| 14 | - - - 0 0 1 1 1 | 0 0 0 0 0 0 0 0 1 0 0 0 | 1 * * 1 * 1 * 1 |
| 15 | - 0 - - 1 0 0 0 | 0 0 0 0 0 0 0 0 0 0 1 1 0 | * 1 1 * 0 * 0 * |
| 16 | - 1 - - 1 0 0 0 | 0 0 0 0 0 0 0 0 0 0 1 1 0 | * 1 0 * 1 * 1 * |

Для примера реализации логической схемы синтезируемого МПА Мура рассмотрим реализацию функций $y_1y_2$ и $J_2K_2$ (рис. 83).

$$y_1 = \bar{\tau}_1\bar{\tau}_2\bar{\tau}_3\tau_4;$$

$$y_2 = \bar{x}_2\bar{\tau}_1\bar{\tau}_2\tau_3\bar{\tau}_4 \vee x_2\bar{\tau}_1\bar{\tau}_2\tau_3\bar{\tau}_4 = \bar{\tau}_1\bar{\tau}_2\tau_3\bar{\tau}_4;$$

$$J_2 = \bar{x}_2\bar{\tau}_1\bar{\tau}_2\tau_3\bar{\tau}_4 \vee x_2\bar{x}_3\bar{\tau}_1\bar{\tau}_2\tau_3\tau_4 \vee x_2\bar{\tau}_1\bar{\tau}_2\tau_3\tau_4 \vee x_2\bar{x}_3\bar{\tau}_1\bar{\tau}_2\tau_3\tau_4 \vee x_2\tau_1\bar{\tau}_2\bar{\tau}_3\bar{\tau}_4 =$$

$$= \bar{x}_2\bar{\tau}_1\bar{\tau}_2\tau_3\bar{\tau}_4 \vee \bar{\tau}_1\bar{\tau}_2\tau_3\tau_4 \vee x_2\tau_1\bar{\tau}_2\bar{\tau}_3\bar{\tau}_4;$$

(использование неопределенных наборов 13 и 14 строк таблицы истинности позволяет выполнить дальнейшее упрощение функции $J_2$)

$$J_2 = \bar{x}_2\bar{\tau}_1\bar{\tau}_2\tau_3\bar{\tau}_4 \vee \bar{\tau}_1\tau_3\tau_4 \vee x_2\tau_1\bar{\tau}_2\bar{\tau}_3\bar{\tau}_4;$$

$$K_2 = \bar{\tau}_1\tau_2\tau_3\bar{\tau}_4 \vee x_4\bar{\tau}_1\tau_2\tau_3\tau_4 \vee x_4\bar{\tau}_1\tau_2\tau_3\tau_4 = \bar{\tau}_1\tau_2\tau_3\bar{\tau}_4 \vee \bar{\tau}_1\tau_2\tau_3\tau_4 = \bar{\tau}_1\tau_2\tau_3;$$

(использование неопределенных наборов 3, 4, 5, 6 и 7 строк таблицы истинности позволяет выполнить дальнейшее упрощение функции $K_2$)

$$K_2 = \bar{\tau}_1\tau_3.$$

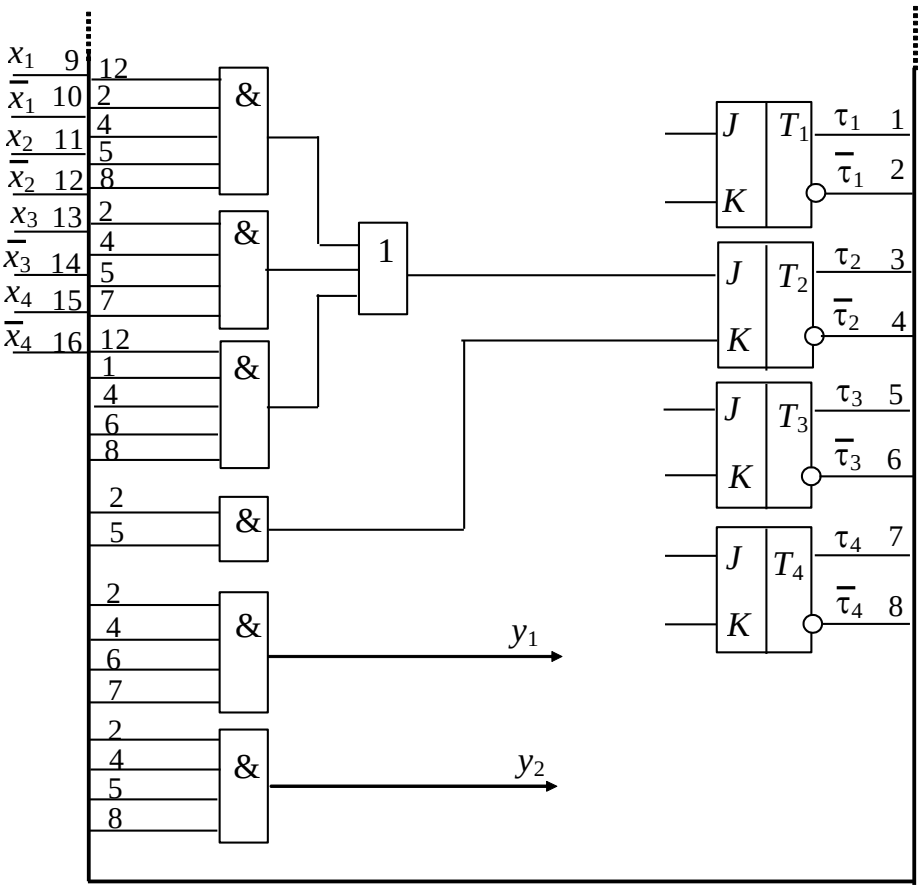



Рис. 83. Логическая схема автомата Мура

В заключение отметим, что, как видно из примеров, синтезированный МПА Мили работает не медленнее чем МПА Мура для одного и того же алгоритма. Для рассмотренных примеров МПА Мили его блок памяти оказался так же более экономичным (на один триггер меньше).

### 5.10. Синхронизация автоматов

Переход автомата из одного состояния в другое осуществляется за счет изменения состояний элементов памяти. На этом этапе может возникать нарушение функционирования автомата, вызванное явлениями, получившими название *гонки* и *риск сбоя*.

*Гонки* возникают из-за неодновременного срабатывания элементов памяти автомата, при переходе из одного состояния в другое, вследствие разброса во времени переключения триггеров, а также различия по времени поступления сигналов на их входы. Например, пусть под действием некоторого входного сигнала $X(a_m, a_s)$ с кодом 00 автомат должен перейти из состояния $a_m$ с кодом 101 в состояние $a_s$ с кодом 110 (рис. 84). Если второй триггер изменит свое значение − переключится из 0 в 1 ранее, чем третий переключится из 1 в 0, то автомат перейдет в промежуточное состояние 111. Иначе, если третий триггер сработает ранее второго, то автомат перейдет в промежуточное состояние 100.

Таким образом, если на некотором переходе в автомате одновременно изменяют свое состояние несколько элементов памяти, то между ними возника-

ет «состязание». Если из промежуточного состояния автомат в конечном счете переходит в требуемое состояние $a_s$, то «состязания» называются некритическими, если в ложное, например 011, − то критическими или гонками.

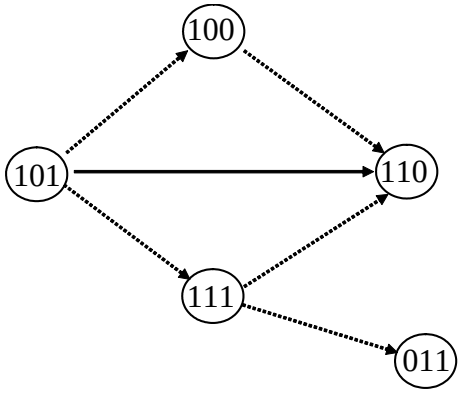



Рис. 84. Демонстрация явления состязаний

Существует два основных подхода к устранению гонок: программный (используя специальные методы кодирования) либо аппаратный. Программный (алгоритмический) подход основан на соседнем кодировании состояний. При соседнем кодировании состояния автомата из множества $A=\{a_1, \ldots, a_m\}$ кодируются таким образом, что на любом переходе из $a_m$ в $a_s$ изменяет свое состояние не более чем один элемент памяти. Однако соседнее кодирование возможно выполнить не для всех автоматов.

Аппаратный подход основан на использовании двух ступеней памяти (рис. 85). Первая ступень памяти построена на триггерах $T_1',\ldots,T_R'$, вторая − на триггерах $T_1'',\ldots,T_R''$ Информация в триггеры первого уровня $T_1',\ldots,T_R'$ записывается по тактовому сигналу Ти, а в триггеры второго уровня $T_1'',\ldots,T_R''$ – по сигналу $\overline{\text{Ти}}$, следующему непосредственно за Ти.

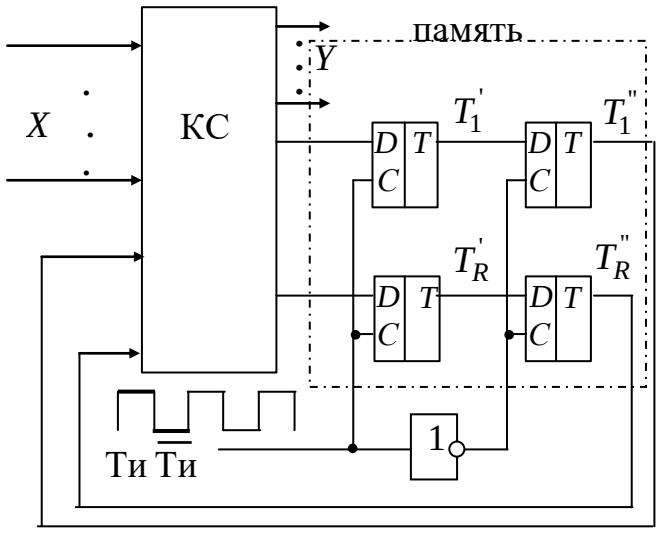



Рис. 85. Автомат с двухуровневой памятью

Если в течение первого полупериода (высокий уровень Ти) между триггерами первой ступени и возникают «состязания», то они все равно не изменяют

состояния триггеров второй ступени, поскольку отсутствует синхросигнал $\overline{\text{Ти}}$ (низкий уровень Ти). Затем, с приходом синхроимпульса $\overline{\text{Ти}}$, изменяют свое состояние триггеры второй ступени. Промежуточные коды, формируемые на их выходах, приводят к изменению (и, возможно, искажению) $\tau_1, \ldots, \tau_r$, а следовательно, и $D_1, \ldots, D_r$. Однако триггеры первой ступени не изменят своего состояния, поскольку отсутствует сигнал Ти. Таким образом, верный код состояния $a_s$ с выходов памяти первой ступени переписывается в триггеры второго уровня, что соответствует переходу автомата в состояние $a_s$.

Если комбинационная схема автомата построена из синхронизируемых элементов, то гонки также устраняются путем разделения синхронизации памяти и комбинационной схемы (рис. 86). В этом случае двойная память не требуется.

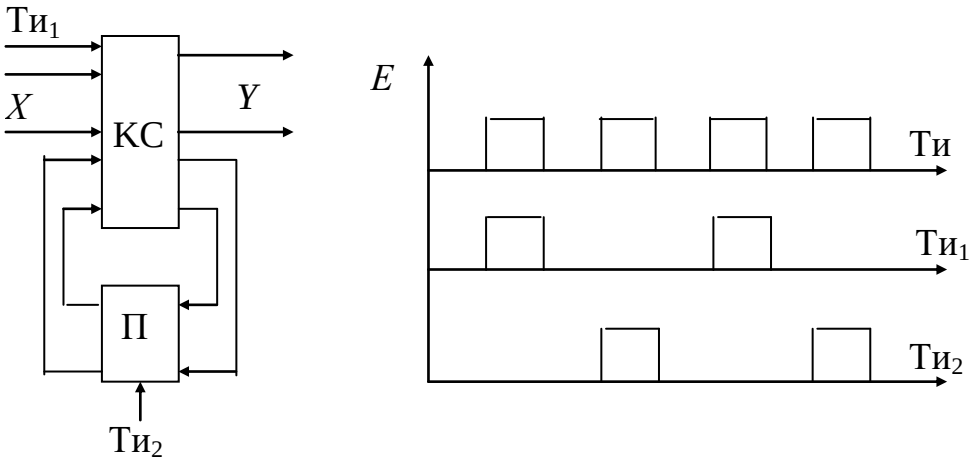



Рис. 86. Раздельная синхронизация

*Риск сбоя*. Наличие некритических «состязаний» не нарушает правил перехода в автомате, но создает возможность возникновения риска сбоя. Риск сбоя заключается в том, что при переходе в некоторое промежуточное состояние (реально существующее в алгоритме) может быть выработан кратковременный ложный выходной сигнал. Например, в автомате Мура при переходе из состояния 101 в состояние 110 появляется кратковременный сигнал $y_k$ в промежуточном состоянии 100 (рис. 87).

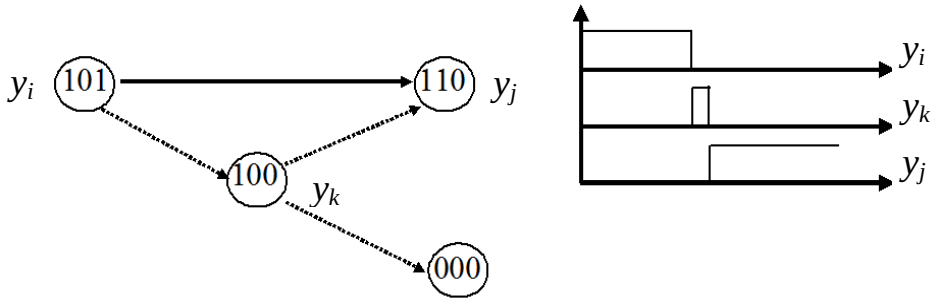



Рис. 87. Демонстрация явления риска сбоя

Хотя длительность ложного сигнала $y_k$ достаточно мала, его возникновение может привести к непредсказуемым последствиям в работе устройства, которым управляет автомат. Для устранения риска сбоя можно воспользоваться следующими методами:

− выходы автомата, на которых может возникнуть риск сбоя, соединить через конденсатор небольшой емкости с нулевым выходом источника питания (рис. 88);

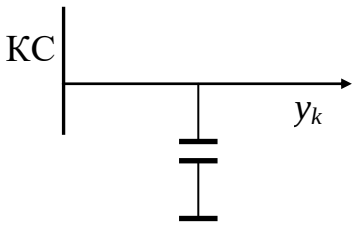



Рис. 88. Устранение риска сбоя с использованием емкости

− произвести буферизацию выходных сигналов;

− выполнить синхронизацию выходных сигналов автомата. При этом комбинационная схема имеет дополнительный вход синхронизации $C_s$, поступающий с задержкой относительно основного синхросигнала. Сигналы на выходах автомата появляются только при наличии этого сигнала. Сигнал можно сформировать, как показано на рис. 89.

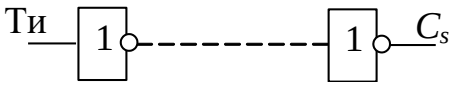



Рис. 89. Устранение риска сбоя с использованием линии задержки Ти

Схема может быть построена, например, из инверторов (их должно быть четное число). Данный подход позволяет устранить риск сбоя только для автомата Мура, так как синхронизация позволяет формировать выходные сигналы после окончания переходных процессов.

**Контрольные вопросы и задания**

1. Дайте определение абстрактного автомата.
2. Приведите способы задания абстрактного автомата.
3. В чем состоит отличие класса автоматов Мили и Мура.
4. Дайте определение структурного автомата.
5. Дайте понятие триггера.
6. Приведите таблицу работы и логическую формулу *D*-триггера.
7. Приведите таблицу работы и логическую формулу *T*-триггера.
8. Приведите таблицу работы и логическую формулу *RS*-триггера.
9. Приведите таблицу работы и логическую формулу *JK*-триггера.
10. Выполнить канонический метод синтеза структурного автомата Мили. Использовать *D*-триггер (*T*-, *RS*-, *JK*-). Закон функционирования автомата задан таблицами:

| δ | $a_1$ | $a_2$ | $a_3$ |
|---|---|---|---|
| $z_1$ | - | $a_3$ | $a_2$ |
| $z_2$ | $a_3$ | $a_2$ | $a_1$ |
| $z_3$ | $a_2$ | - | - |

| λ | $a_1$ | $a_2$ | $a_3$ |
|---|---|---|---|
| $z_1$ | - | $w_3$ | $w_1$ |
| $z_2$ | $w_4$ | $w_3$ | $w_3$ |
| $z_3$ | $w_3$ | - | - |

11. Выполнить канонический метод синтеза структурного автомата Мура. Использовать *D*-триггер (*T*-, *RS*-, *JK*-). Закон функционирования автомата задан совмещенной таблицей:

| λ | $w_1$ | - | $w_2$ |
|---|---|---|---|
| δ | $a_1$ | $a_2$ | $a_3$ |
| $z_1$ | - | $a_2$ | $a_2$ |
| $z_2$ | $a_1$ | $a_3$ | $a_3$ |
| $z_3$ | $a_3$ | - | - |

12. Что представляет собой явление «состязания» элементов памяти автомата и как оно связано с гонками?

13. Что представляет собой явление «состязания» элементов памяти автомата и как оно связано с риском сбоя?

**Литература**

1. Савельев, А. Я. Прикладная теория цифровых автоматов: учебник для вузов по специальности ЭВМ / А. Я. Савельев. – М. : Высшая школа, 1987. – 272 с.

2. Савельев, А. Я. Основы информатики: учебник для вузов/ А. Я. Савельев. –М. : Изд-во МГТУ им. Н. Э. Баумана, 2001. – 328 с.

3. Майоров, С. А., Новиков, Г. И. Проектирование цифровых вычислительных машин / С. А. Майоров. –М. : Высшая школа, 1972. – 343 с.

4. Поснов, Н.Н. Арифметика вычислительных машин в упражнениях и задачах: системы счисления, коды. / Н. Н. Поснов. – Минск : Университетское, 1984 . – 223 с.

5. МикроЭВМ, микропроцессоры и основы программирования / А.Н. Морозевич [и др.] ; под ред. А. Н. Морозевича. – Минск : Выш. шк, 1990 – 352 с.

6. Гашков, С. Б. Системы счисления и их применение. (серия «Библиотека ”Математическое просвещение”») / С. Б. Гашков. – М. : МЦНМО, 2004. – 52 с..

7. Питерсон, У., Уэлдон, Э. Коды, исправляющие ошибки / пер. с англ. –М. : Мир, 1976. – 593 с.

8. Лысиков, Б. Г. Цифровая вычислительная техника: учеб. /Б. Г. Лысиков. – Минск : УП Экоперспектива, 2002. – 264 с

9. Лысиков, Б. Г. Арифметические и логические основы цифровых автоматов: учеб. для вузов по специальности «Электрон. вычисл. машины» / Б.Г. Лысиков. – 2-е изд., перераб. и доп. – Минск : Выш.школа, 1980. – 336 с.

10. Угрюмов, Е. П. Цифровая схемотехника: учеб. пособие / Е. П. Угрюмов. – 2-е изд., перераб. – СПб. : БХВ, 2004. – 528 с.

11. Андреева, Е. Н., Фалина, И. Н. Системы счисления и компьютерная арифметика: серия «Информатика» / Е. Н. Андреева – 2-е изд. – М. : Лаборатория Базовых Знаний, 2000. – 248 с.

12. Глушков, В.М. Синтез цифровых автоматов / В. М. Глушков. – М. : Физматгиз, 1962 – 476 с.

13. Захаров, Н. Г., Рогов, В. Н. Синтез цифровых автоматов: учеб. пособие / Н. Г. Захаров, В. Н. Рогов. – Ульяновск : УлГТУ, 2003. – 135 с.

14. Миллер, Р. Теория переключательных схем. В 2 т. Т1. / Р. Миллер. – М. : Наука, 1970. – 416 с.

15. Карпов, Ю. Г. Теория автоматов / Ю. Г. Карпов. – СПб. : Питер, 2003. – 208 с.

16. Яблонский, С. В. Введение в дискретную математику: учеб. пособие / С. В. Яблонский. – 2-е изд. – М. : Наука, 1986. – 384 с.

17. Баранов, С. И. Синтез микропрограммных автоматов / С. И. Баранов. – Л. : Энергия, 1979. – 232 с.

18. Шоломов, Л. А. Основы теории дискретных логических и вычислительных устройств / Л. А. Шоломов. – М. : Наука, 1980. – 400 с.

19. Скляров, В. А. Синтез автоматов на матричных БИС / В. А. Скляров – Минск : Наука и техника. 1984. – 286 с.

Содержание

Св. план 2013, поз. 34

*Учебное издание*

**Луцик** Юрий Александрович
**Лукьянова** Ирина Викторовна

# АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОСНОВЫ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

УЧЕБНОЕ ПОСОБИЕ

Редактор *Е.С. Чайковская*
Корректоры
Компьютерная правка, оригинал - макет
Подписано в печать . .2013. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Times». Отпечатано на ризографе. Усл. печ. л.
Уч.-изд. л. . Тираж 250 экз. Заказ .

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П.Бровки, 6.