АРИФМЕТИЧЕСКИЕ И ЛОГИЧЕСКИЕ ОСНОВЫ

ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

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. Достаточно 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.

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

В заключение следует отметить, что в течение ряда лет литература, освещающая арифметику вычислительных машин, не выпускалась. В пособием сделана попытка устранить этот информационный пробел. Материал пособия базируется на работах [1-5].

# Арифметические основы вычислительной техники

## Системы счисления

В ЭВМ информация всегда 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 другие. Система счисления - совокупность 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 оперирования числами.

Различают два типа систем счисления - непозиционные и позиционные.

*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.

*Позиционной* системой счисления называется система записи любых по величине чисел ограниченным числом символов.

*Основание (базис)*  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 1 и бесконечности.

Любое число в системе счисления с основанием r может быть записано в общем виде:

A=an·rn+ an-1·rn-1+...+a1·r1+a0·r0+ a-1·r--1+...+a-rn-1·r-(rn-1)+a-rn·r-rn, (1)

или

,  (2)

где любая разрядная цифра ai∈{0,…,r-1}, a ri  - вес соответствующего разряда.

Запись числа в форме (1) назовем записью числа в развернутой форме. Свернутой формой записи чисел называется запись чисел в виде

A=a1a2 … ak.

Для любой системы счисления основание представляется как 1 (один) и 0 (ноль).

Например: 9 1 F 7

+1 +1 +1 +1

1010 102 1016 108

Вес разряда pi числа выражается соотношением:

pi = ri /r0 = ri ,

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 называются переносом. При вычитании передача 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 остановимся на них.

## Двоичная система счисления

Для записи числа в двоичной системе счисления используются две цифры: 0 и 1. Основание системы записывается как 10(2) (210=1·21+0·20). Используя данную систему любое число можно выразить последовательностью высоких и низких потенциалов или группой запоминающих 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 · 0=0 |  |
|  | 0+1= 1 |  | 1-0=1 |  | 0 · 1=0 |  |
|  | 1+0= 1 |  | 1-1=0 |  | 1 · 0=0 |  |
|  | 1+1=10 |  | 10-1=1 |  | 1 · 1=1 |  |

Рассмотрим несколько примеров демонстрирующих выполнение арифметических операций:



## Восьмеричная система счисления

В восьмеричной системе счисления используется восемь цифр: 0,1,2 … 7, а основание записывается как 10(8) (810=1·81+0·80). Рассмотрим примеры выполнения операций в восьмеричной системе счисления. При их выполнении используются правила представленные в таблицах сложения и умножения восьмеричных цифр.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Сложение | | | | | | | |  | Умножение | | | | | | | |  |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 10 |  | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |
|  | 2 | 3 | 4 | 5 | 6 | 7 | 10 | 11 |  | 2 | 2 | 4 | 6 | 10 | 12 | 14 | 16 |  |
|  | 3 | 4 | 5 | 6 | 7 | 10 | 11 | 12 |  | 3 | 3 | 6 | 11 | 14 | 17 | 22 | 25 |  |
|  | 4 | 5 | 6 | 7 | 10 | 11 | 12 | 13 |  | 4 | 4 | 10 | 14 | 20 | 24 | 30 | 34 |  |
|  | 5 | 6 | 7 | 10 | 11 | 12 | 13 | 14 |  | 5 | 5 | 12 | 17 | 24 | 31 | 36 | 43 |  |
|  | 6 | 7 | 10 | 11 | 12 | 13 | 14 | 15 |  | 6 | 6 | 14 | 22 | 30 | 36 | 44 | 52 |  |
|  | 7 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |  | 7 | 7 | 16 | 25 | 34 | 43 | 52 | 61 |  |



## Шестнадцатеричная система счисления

В шестнадцатеричной системе счисления используются шестнадцать символов: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Основание записывается как 10(16) (1610=1\*161+0\*160).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сложение | | | | | | | | | | | | | | | |  | Умножения | | | | | | | | | | | | | | | |
|  | 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 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | а | b | c | d | e | f | 10 |  | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | 10 | 11 |  | 2 | 2 | 4 | 6 | 8 | a | c | e | 10 | 12 | 14 | 16 | 18 | 1a | 1c | 1e |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f | 10 | 11 | 12 |  | 3 | 3 | 6 | 9 | c | f | 12 | 15 | 18 | 1b | 1e | 21 | 24 | 27 | 2a | 2d |
| 4 | 5 | 6 | 7 | 8 | 9 | A | b | c | d | e | f | 10 | 11 | 12 | 13 |  | 4 | 4 | 8 | c | 10 | 14 | 18 | 1c | 20 | 24 | 28 | 2c | 30 | 34 | 38 | 3c |
| 5 | 6 | 7 | 8 | 9 | a | B | c | d | e | f | 10 | 11 | 12 | 13 | 14 |  | 5 | 5 | a | f | 14 | 19 | 1e | 23 | 28 | 2d | 32 | 37 | 3c | 41 | 46 | 4b |
| 6 | 7 | 8 | 9 | a | b | C | d | e | f | 10 | 11 | 12 | 13 | 14 | 15 |  | 6 | 6 | c | 12 | 18 | 1e | 24 | 2a | 30 | 36 | 3c | 42 | 48 | 4e | 54 | 5a |
| 7 | 8 | 9 | a | b | c | D | e | f | 10 | 11 | 12 | 13 | 14 | 15 | 16 |  | 7 | 7 | e | 15 | 1c | 23 | 2a | 31 | 38 | 3f | 46 | 4d | 54 | 5b | 62 | 69 |
| 8 | 9 | a | b | c | d | E | f | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |  | 8 | 8 | 10 | 18 | 20 | 28 | 30 | 38 | 40 | 48 | 50 | 50 | 60 | 68 | 70 | 78 |
| 9 | a | b | c | d | e | F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |  | 9 | 9 | 12 | 1b | 24 | 2d | 36 | 3f | 48 | 51 | 5a | 63 | 6c | 75 | 7e | 87 |
| a | b | c | d | e | f | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |  | a | a | 14 | 1e | 28 | 32 | 3c | 46 | 50 | 5a | 64 | 6e | 78 | 82 | 8c | 96 |
| b | c | d | e | f | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |  | b | b | 16 | 21 | 2c | 37 | 42 | 4d | 58 | 63 | 6e | 79 | 84 | 8f | 9a | a5 |
| c | d | e | f | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |  | c | c | 18 | 24 | 30 | 3c | 48 | 54 | 60 | 6c | 78 | 84 | 90 | 9c | a8 | 84 |
| d | e | f | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |  | d | d | 1a | 27 | 34 | 41 | 4e | 5b | 68 | 75 | 82 | 8f | 9c | a9 | b6 | c3 |
| e | f | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |  | e | e | 1c | 2a | 38 | 46 | 54 | 62 | 70 | 7e | 8c | 9a | a8 | b6 | c4 | d2 |
| f | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |  | f | f | 1e | 2d | 3c | 4b | 5a | 69 | 78 | 87 | 96 | a5 | b4 | c3 | d2 | e1 |



## Критерии выбора системы счисления

Сформулируем требования, которым должна удовлетворять система счисления для ЭВМ.

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, то есть 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 другие.

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 наибольшего и наименьшего числа одинаков. Очевидно преимущество систем с меньшим основанием, так как представление соседних цифр в этих системах отличаются друг от друга больше, чем для систем с большим основанием.

Очевидно, что при наложении помехи на основной сигнал, соответствующий некоторой цифре, наиболее вероятна ошибка в устройствах, для которых используется система счисления с наибольшим основанием (см. рис.1). Следовательно, при увеличении основания помеха может привести к искажению числа.

3. ***Минимум оборудования***. Пусть b - количество цифр в числе, n - количество разрядов в каждом числе, тогда D = b∙n - количество цифроразрядов на одно число. Надо найти такую систему счисления, которая имеет минимальное





количество цифроразрядов при заданном количестве чисел N:

Рис. 1. Влияние помех на значение цифры

N= ,

n=,

D==.

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

N= ,

n=,

D==.

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(b)=.

Это позволит свести задачу нахождения D(b)min к исследованию функции на экстремум:

,

следовательно, rопт=e ≅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 ri уступает rопт. Для этого оценим каждое основание ri исходя из выражения:

D i (отн)=,

где Dmin= rопт log r оптN=e ∙ ln N. Следовательно получим

D(b)=.

Выполнив расчеты для некоторых оснований, получим следующие результаты:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ri | 2 | 3 | 4 | 5 | 6 | 7 | 8 | **. . .** |
|  | D i (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 доказательство 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 ее за базовую, вычислим соотношение времени, необходимого на сложение:

Tсл = n tпер = tпер logbN,

Для упрощения оценки перейдем от абсолютной оценки к относительной:

ő =,

Tсл max = Tсл (2) = tпер log2N,

ő=,

ri 2 3 4 5  **. . .**

ő 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 работы человека является десятичная система счисления. Но внутри 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***.

9.***Единственность представления числа***.

## Перевод чисел из одной системы счисления в другую

Так как числа, участвующие в операциях, могут быть представлены в различных позиционных системах счисления, то для выполнения действий над ними требуется привести их к одной системе счисления. Необходимо отметить, что целая и дробная части числа переводятся отдельно. Следовательно все методы перевода чисел можно подразделить на две группы: перевода целых и дробных чисел.

### Перевод целых чисел.

*Метод подбора степеней основания*. В соответствии с (2) целые числа в системах счисления с основаниями r1 и r2 могут быть представлены:

n k

A r1 =  ai r1i =  bj r2j = A r2 ,

i=0 j=0

В общем случае перевод числа из системы счисления с основанием r1 в систему счисления с основанием r2 можно представить как задачу определения коэффициентов bi нового ряда, изображающего число в системе счисления с основанием r2. Основная трудность в выборе максимальной степени основания r2, которая еще содержится в числе Ar1. Все действия должны выполняться по правилам r1-арифметики (то есть исходной системы счисления). После нахождения максимальной степени и соответствующего ей коэффициента необходимо найти коэффициенты для всех остальных (младших) степеней.

Пример: A10=37 , A2=?

37=1·25 + 0 ·24 + 0 ·23 + 1·22 + 0 ·21 + 1·20=100101

Нечетным двоичным числом 100101 является число, содержащее единицу в младшем разряде.

*Метод деления на основание системы счисления.* На основании (1) число Ar1 в системе счисления с основанием r2 запишется в виде:

Ar2 = an·r2n + an-1 ·r2n-1 + ... + a1·r21 + a0·r0.

Переписав это выражение по схеме Горнера, получим:

Ar2 = (...(an r2 + an-1) r2 + ... + a1) r2 + a0.

Разделив правую часть на r2, получим первый остаток a0 и целую часть

(...(an r2 + an-1) r2 + ... + a1). Разделив целую часть на r2, получим остаток a1 и новую целую часть. Выполнив деление n+1 раз, получим последнее целое частное an < r2, являющееся старшей цифрой числа.

Пример А10 = 37 ; A2 = ?; А5=?



### Перевод правильных дробей.

*Метод подбора величин, обратных степеням основания*.

A10 =0,716

А2 =0,1011...



Количество разрядов после запятой зависит от точности, с которой требуется представить число.

*Метод умножения на основание r2 новой системы счисления*. Из выражения (1) дробное число Ar1 в системе счисления с основанием r2 запишется в виде

Ar2 = a-1 r2-1 + ... + a-n r2-n.

Переписав это выражение по схеме Горнера, получим:

Ar2 = r2-1 (a-1 + r2-1 (a-2 + ... + a-n r2-1)...).

Умножив правую часть на r2, получим новую неправильную дробь, целая часть которой есть a-1 (старшая цифра числа Ar2). Продолжим процесс умножения дробной части на r2 n-1 раз, получим цифры a-2, a-3, . . . числа Ar2.

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

Пример A10=0,673 A2=? A16=?



Для перевода неправильных дробей отдельно выделяется целая и дробная части числа и, используя соответствующие методы, выполняется их перевод. Результаты записываются в виде новой неправильной дроби.

### Перевод чисел из системы счисления в систему счисления основания которых кратны степени 2

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

A8 = ,

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

+ младшая восьмеричная цифра с весом 80

++

+

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.

Пример: А8 = 45 А2= 0010 0101

100 101 = А2 2 5 = А16

## 

## 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 отрицательных 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.



Пример: A=+0,1101 A=+ 1101

[A]пр=0,1101 [A]пр=0.1101



Пример: A = - 0,1101 A = - 1101

[A]пр=1,1101 [A]пр=1.1101

Диапазон изменения машинных изображений для прямого кода лежит в пределах: -(1-2-n)[A]пр(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, то есть на этом сумматоре невозможно выполнение операции алгебраического сложения.

**Дополнительный код** числа. Число А′ называется *дополнением* к числу А, если выполняется соотношение: А + А′ = rn  для целых чисел, или А + А**'**=r0 для дробных чисел, где n- количество цифр в записи числа A.

Пример: A10 =378

n=3

A10′ =103 – А10=1000 - 378=622

378

621 - все разряды дополняются до младшей цифры системы счисления

1 - младший разряд дополняется до основания системы счисления

1000

n=4

А2 =1011,A2 ′=24 - А=10000 - 1011 = 0101 или А2′ = 0101

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

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

Для сложения чисел требуется сумматор и неважно, какие слагаемые подаются на его входы А или В. Пусть необходимо сложить

А = 487 А = 487

В = -348 В = 652

А-В = 139 А-В = 1 139

А + (103 – В) = А-В+103 (103 игнорируется).

А = 348 А = 348

В = -487 В = 513

А-В = -139 А-В = 861

Дополнительный код является математическим дополнением основания системы счисления.

 - для дробных чисел,  - для целых чисел

где  - абсолютное значение числа А, n – число цифр числа.

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





Рассмотрим несколько примеров сложения чисел в дополнительных кодах.

А= 0,1001 [A]доп = 0,1001 А= - 0,1001 [A]доп = 1,0111

В= - 0,0100 [B]доп = 1,1100 В= 0,0100 [B]доп = 0,0100

10,0101 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:

,

.

Выполнение арифметических операций над числами в обратном коде осуществляется на *сумматоре обратного кода.* Этот код имеет несущественный 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 дополнительном коде.

А= 0,1001 [A]обр = 0,1001 А= - 0,1001 [A]обр = 1,0110

В= - 0,0100 [B]обр = 1,1011 В= 0,0100 [B]обр = 0,0100

10,0100 1,1010

1

0,0101

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

Доказательство теоремы приведено в [1].

## Переполнение разрядной сетки

При выполнении некоторых арифметических операций может возникать явление переполнения разрядной сетки. Причиной переполнения может служить суммирование двух чисел с одинаковыми знаками (для чисел с разными знаками переполнение не возникает), которые в сумме дают величину, большую или равную 1 (при сложении правильных дробей) и величины rn (при сложении целых чисел).

Пример: A=+0,101 [A]доп = 0.101

B=+0,110 [B]доп = 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 или только из знакового разряда.

Функция переполнения имеет вид: f=П1П2 + П1П2 = П1 ⊕ П2

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 [A]доп = 1.011

B=- 0,011 [B]доп = 1.101

[A]доп+[B]доп = 1,000

## Модифицированные коды

Для обнаружения переполнения разрядной сетки можно использовать модифицированные коды. Модифицированные коды отличаются от обычных кодов тем, что знак числа кодируется двумя разрядами. 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).

А=+0,101 [A] моддоп = 00,101

B=+0,110 [B] моддоп = 00,110

[A]моддоп+[B] моддоп = 01,011

А=-0,101 [A] моддоп = 11,011

B=-0,110 [B] моддоп = 11,010

[A]моддоп+[B] моддоп = 10,101

Функция переполнения имеет вид: f=Зн1 Зн2 + Зн1 Зн2 = Зн1 ⊕ Зн2.

Логическая схема формирования единичного сигнала при возникновении переполнения имеет следующий вид



## Машинные формы представления чисел.

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

**Представление чисел в форме с фиксированной запятой**. Для сокращения длины разрядной сетки и упрощения обработки данных положение запятой может быть зафиксировано схемотехнически. При этом в слове данных сохраняются только две структурных компоненты: поле знака и поле цифр.

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

Определим диапазон представления чисел для этого формата.

Аmax=(2k-1)+(1-2-l).

В зависимости от размеров целой и дробной частей возможно следующее:

1) k=0, l=n Amax=1-2-n

± **,** 1 1  **. . .**  1 1

2) k=n, l=0 Amax=2n-1

± 1 1  **. . .**  1 1,

3) k=0, l=n Amin=2-n

± **,** 0 0  **. . .**  0 1

4) k=n, l=0 Amin=1

± 0 0  **. . .**  0 1,

Очевидно, что ограничение длины разрядной сетки приводит к ограничению диапазона хранимых чисел и потере точности их представления. Поэтому на практике широко используется и другая форма представления чисел.

**Представление чисел в форме с плавающей запятой.** В общем виде числа с плавающей запятой имеют следующий вид:

A=±mAr±pA,

где mA- мантисса, а рA- порядок числа А. Порядок (с учетом знака) показывает, на сколько разрядов и в какую сторону сдвинута запятая при замене формы записи числа с естественной на нормальную.

Например, А10 = 239,745 = 0,239745 103 = 239745 10-3.

Наиболее распространено и удобно для представления в ЭВМ ограничение вида r-1≤⏐mA⏐<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. Формат машинного изображения чисел с плавающей запятой должен включать знаковые поля (мантиссы и порядка), поле мантиссы и поле порядка числа и имеет следующий вид:

|  |  |  |  |
| --- | --- | --- | --- |
| ± | мантисса mA | ± | порядок p |

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



± **,** 1 1  **. . ,**  1 ± 1 **. . .** 1 1



± **,** 1 0  **. . ,**  0 ± 1 **. . .** 1 1

Для упрощения операций над порядками применяют представление чисел с плавающей запятой со смещенным порядком: p′=p+N, где N – целое положительное число (смещение), N= max(-p). Обычно N=2k, 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*. В зависимости от типа данных числа с плавающей запятой в памяти ЭВМ хранятся в одном из следующих трех форматов:







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 – исчезновение порядка (потеря порядка) получение очень малого числа близкого к нулю.

В стандарте IEEE крайние значения порядка (характеристики) зарезервированы и не используются для представления обычных чисел. Максимальное значение характеристики представленное всеми единицами при положительном знаке числа зарезервировано для представления значения (+ ∞) при нулевой мантиссе. При знаке минус число с максимальной характеристикой используется для представления (- ∞) и неопределенности. Значение с минимальной характеристикой равной нулю зарезервировано для представления денормализованных чисел (положительных и отрицательных), а также для представления нуля (представляется всеми нулями), причем различают +0 и –0.

## Погрешность выполнения арифметических операций

Выбор длины разрядной сетки и формы представления чисел тесно связаны с точностью получаемых при арифметических операциях результатов. При выполнении операций над числами с фиксированной запятой можно считать что результат точен (при условии отсутствии 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: А=[A]+∆A и B=[B]+∆B.

A+B=[A]+[B]+(∆A+∆B),

где абсолютная погрешность суммы ∆(A+B)=∆A+∆B.

A-B=[A]-[B]+(∆A-∆B),

где абсолютная погрешность разности ∆(A-B)=∆A-∆B.

A⋅B=[A][B]+[A]∆B+[B]∆A+∆A∆B

произведением ∆A∆B можно пренебречь, следовательно,

A⋅B≈[A][B]+[A]∆B+[B]∆A,

то есть абсолютная погрешность произведения ∆(AB)≈[A]∆B+[B]∆A.

При выполнении операции деления



абсолютная погрешность частного ∆(A/B)=∆A/[B]-[A]∆B/([B])2.

## Округление

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

В общем виде число с плавающей запятой размещенное в разрядной сетке размерностью k имеет вид Ar=mark . Если для записи мантиссы используются только n разрядов, то число может быть представлено в виде двух частей: Ar=[ma]rn+[A0]rk-n, где [A0]rk-n=A0 – часть числа не вошедшая в разрядную сетку размерностью k.

В зависимости от того, как учитывается А0, при записи числа А в n-разрядную сетку, можно выделить несколько способов округления чисел.

1. *Отбрасывание А0*. При этом возникает относительная погрешность

õокр=|А0|rk-n/(|mA|rn), так как r -1≤ |mA| <1, 0≤ |A0| <1, то õокр=r--(n-1).

1. *Симметричное округление*. При этом производится анализ величины А0:



При условии |А0|≥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.

## Нормализация чисел

Число называется нормализованным, если его мантисса удовлетворяет условию 0 < |MA| ≤ r -n .

Нормализация – процесс, относящийся к числам, записанным в форме с плавающей запятой. 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 А необходимо сдвинуть влево на два разряда. При этом порядок необходимо уменьшить на два. Различают два вида сдвигов: простой и модифицированный.

Простой сдвиг – сдвиг, выполняемый по правилу:

|  |  |  |
| --- | --- | --- |
| Исходная  комбинация | Сдвиг влево | Сдвиг вправо |
| 0,a1a2….an | a1,a2….an0 | 0,0a1a2….an-1 |
| 1,a1a2….an | a1,a2….anα | 0,1a1a2….an-1 |

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

|  |  |  |
| --- | --- | --- |
| Исходная  комбинация | Сдвиг влево | Сдвиг вправо |
| 00,a1a2….an | 0a1,a2….an0 | 00,0a1a2….an-1 |
| 01,a1a2….an | 1a1,a2….an0 | 00,1a1a2….an-1 |
| 10,a1a2….an | 0a1,a2….anα | 1,1a1a2….an-1 |
| 11,a1a2….an | 1a1,a2….anα | 1,1a1a2….an-1 |

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

## Последовательное и параллельное сложение чисел

Параллельный способ передачи информации является более быстродействующим по сравнению с последовательным, но менее экономичным (требуется вместо одного проводника (и усилителя сигналов) n проводников).

В соответствии со способом приема/передачи информации устройства, обрабатывающие эту информацию, могут быть либо параллельного, либо последовательного действия.

***Последовательный сумматор.***

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

Tc(посл) ≅ n t, t-время задержки сигнала переноса на элементе задержки.



В этой схеме на входы x и y последовательно подаются попарно разряды слагаемых xi и yi. На выходе S формируются si разряды суммы. Триггер введен в схему для хранения значения переноса до следующего такта (следующей пары разрядов).

***Параллельный сумматор.***

старший разряд младший разряд

xn SM S x2  SM S x1 SMS

**. . .**

yn  y2 y1

P P P

Рис. 4. Схема параллельного суммирования.

Тс(пар) ≅ nτлэ ,

Очевидно, Tc(посл) ≤ Тс(пар), так как

t ≅ (3,6) kτлэ ,

где k - коэффициент запаса, обеспечивающий полное окончание всех переходных процессов в сумматоре последовательного действия k ∈ [1,2, 1,3].

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

## Сложение чисел с плавающей запятой

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

A=±mAr±pA,

B=±mBr±pB.

Алгоритм сложения чисел с произвольными знаками состоит в следующем.

1. Производится сравнение порядков pA и pB. Для этого из порядка числа A вычитается порядок числа B. Разность p=pA-pB указывает, на сколько разрядов требуется сдвинуть вправо мантиссу числа с меньшим порядком. Если p=pA-pB>0, то pA>pB и для выравнивания порядков необходимо сдвинуть вправо мантиссу MB. Если p=pA-pB<0, то pB>pA и для выравнивания порядков необходимо сдвинуть вправо мантиссу MA. Если p=pA-pB=0, то pA=pB и порядки слагаемых выравнивать не требуется.

2. Выполняется сдвиг соответствующей мантиссы до тех пор, пока p≠0.

3. Выполняется сложение мантисс MA и MB по правилу сложения правильных дробей.

4. Если при сложении мантисс произошло переполнение, то производится нормализация путем сдвига мантиссы суммы вместе со знаковым разрядом вправо на один разряд с увеличением порядка на единицу. Если же происходит денормализация, то выполняется сдвиг мантиссы результата на соответствующее количество разрядов в сторону, противоположную нарушению нормализации с соответствующим изменением порядка суммы.

Пример: МА=-0,10110 рА=+0111

МВ=-0,11011 рВ=+0101

[MA]доп=1,01010 p= [рА]доп+[-рВ]доп= 0.0111

[MB]доп=1,00101 1.1011

1 0.0010

так как [рА-рВ]доп>0, то сдвигу подвергается мантисса МВ.

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 [рА-рВ]доп<0, то необходимо либо прибавлять единицу до нуленого результата, либо измненить знак разности на противоположный и, как и выше, выполнять вычитание единицы.

[MB]доп=1,00101 0.0010

[-1]доп= 1,1111

[MB]доп=1,**1**0010 1 0.0001

[MB]доп=1,**11**001 01 [-1]доп= 1,1111

0.0000

[MB]доп=1,11001 01

[MA]доп=1,01010

11,00011 01 = [МА+В] рА+В=max(рА,pB)=pA=+0.0111

Полученный результат нормализован. После выполнения операции округления получим [МА+В]= 1,00011.

## Машинные методы умножения чисел в прямых кодах

Операция умножения состоит из ряда 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.

Введем некоторые обозначения, используемые ниже:  - частичное произведение,  - частичная сумма.

Ниже приводится схема четырех алгоритмов умножения .

**

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

Представим Мн= А = 0,а1а2…аn

Мт= B = 0,b1b2….bn = b12-1+b22-2+…+bn2-n+bn2-n.

Мн∙Мт = С=А∙В= 0,а1а2…аn ( b12-1+b22-2+…+bn2-n)=

=0+(b1∙0,а1а2…аn)2-1+…+(bn-1∙0,а1а2…аn)2-(n-1)+(bn∙0,а1а2…аn)2-n=

=0+b1∙A2-1+…+bn-1∙A2-(n-1)+…+bn∙A2-n=0+bn∙A2-n+bn-1∙A2-(n-1)+…+b1∙A2-1=

=(…((0+bn∙A)2-1+bn-1∙A)2-1+…+b1∙A)2-1

Ниже приведены (без вывода) остальные три реализации алгоритмов (Б, В и Г) умножения.

Мн∙Мт = С=А∙В = (0+bn∙A+bn-1∙A∙22+…+b1∙A∙2n-1)2-n (алгоритм Б)

Мн∙Мт = С=А∙В = (…(0+b1∙A) ∙21+b2∙A)∙21+…+bn∙A)∙2-n (алгоритм В)

Мн∙Мт = С=А∙В = 0+b1∙A∙2-1+b2∙A∙2-2+…+bn∙A∙2-n  (алгоритм Г)

Структурные схемы операционных устройств, выполняющих умножение по алгоритмам А,Б,В и Г приведены на рис 7.

Рассмотрим пример умножения чисел согласно алгоритму А.

Пример: МH = 0,1011

МT = 0,1101

b1 … b4

4

0,0000  начальное содержимое сумматора

0,1011 = Мн ∙ b4  первое частичное произведение

0,1011  первая частичная сумма

0,**0**101 1 ∙ 2-1 сдвиг первой частичной суммы

0,0000 = Мн ∙ b3 второе частичное произведение

0,0101 1  вторая частичная сумма

0,**0**010 11 ∙ 2-1 сдвиг второй частичной суммы

0,1011 = Мн ∙ b2 третье частичное произведение

0,1101 11  третья частичная сумма

0,**0**110 111 ∙ 2-1 сдвиг третьей частичной суммы

0,1011 = Мн ∙ b1 четвертое частичное произведение

1,0001 111  (возникло переполнение)

0,**1**000 1111 ∙ 2-1 сдвиг для получения верного результата Мн∙Мт

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.

Время умножения чисел по алгоритму А tумн = ( tсл + tсдв ) ⬝ n, где n - число разрядов Мт. Следовательно, сдвиг и сложение нельзя выполнять в одном автоматном такте. Это наглядно показано на рис 6.





## 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:

 (3)

где pi вероятность появления единицы в разрядах множителя, tсл – время формирования очередной частичной суммы, tсдв – время выполнения сдвига числа на один разряд. Анализируя выражение (3), можно предложить различные пути сокращения величины Тумн.: 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 встрече нулевого разряда в множителе не выполнять формирование очередного (нулевого) частичного произведения, не изменяющего содержимое сумматора. В зависимости от используемого алгоритма умножения выполняется сдвиг либо частичной суммы, либо частичного произведения без выполнения суммирования.

### Умножение с хранением переносов

Время, затрачиваемое на сложения двоичных чисел, состоит из времени необходимого для поразрядного сложе­ния и времени на формирование переноса tсл = t+ + tпер. Поразрядное сложение является элементарной опера­цией, и время на эту операцию может быть сокращено путем использования более бы­стродействующих элементов. В то же время, если исключить необходимость вы­полнения межразрядных переносов при сложении, то время умножения уменьшится на tпер. Переносы, формируемые при сложении, записываются в отдельный регистр. Содержимое этого регистра добавляется в сумматор вместе с очередым частичным произведением. При этом сложение может выполняться паралельно по всем разрядам. В заключение следует отметить, что этот метод используется с алгоритмом A.

Пример: МН = 0,1011

МТ= 0,1101

0,0000 

0,0000 регистр переносов

0,1011 = Мн ⬝ b4

0,1011 

0,0000 регистр переносов

0,**0**101 1 ⬝ 2-1

0.0000 = Мн ⬝ b3

0,0101 1 

0,0000 регистр переносов

0,**0**010 11 ⬝ 2-1

0,1011 = Мн ⬝ b2

0,1001 11 

0,0010 регистр переносов

0,**0**100 111 ⬝ 2-1

0,1011 = Мн ⬝ b1

0,1101 111 

0,0010 регистр переносов

0,**0**110 1111 ⬝ 2-1

0,1000 1111 

### Умножение на два разряда множителя одновременно.

Разбиение множителя на группы длиной k разрядов означает переход к новой системе счисления с основанием 2k. Если при этом удается сократить количество элементарных действий, выполняемых при умножении (сложение и сдвиги), то сокращается время умножения. Остановимся более подробно на примере умножения на два разряда одновременно (k=2).

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

В случае пары 00 необходимо выполнить только сдвиг частичной суммы на два разряда - ⬝ 2-2 .

Для пары 01 выполняется добавление множимого в сумматор с последующим сдвигом суммы на два разряда - ⬝ 2-2 ,

При наличии пары 10 возможны следующие варианты действий:

a) ⬝ 2-2 , то есть в этом случае происходят два сложения, что противоречит требованию;

б) ⬝ 2-2, в этом случае требуется дополнительный регистр для хранения удвоенного Мн;

в) ⬝ 2-2, что соответствует добавлению к частичной сумме сдвинутого на один разряд влево множимого;

г) ⬝ 2-1, то есть 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) ее можно представить в виде:

11 = (22 - 1)

МН ∙ 11= МН∙(22 - 1) = МН∙22- МН, то есть в текущем такте к частичной сумме добавляется множимое, взятое со знаком минус. Добавление МН∙22 реализуется путем увеличения на единицу следующей старшей пары разрядов.

В приведенной ниже таблице представлены правила преобразования множителя для системы (0,1,1).

Таблица 1.

|  |  |  |
| --- | --- | --- |
| Анализируемая пара разрядов Мт | перенос из предыдущей пары | Преобразованная пара . |
| 00 | 0 | 00 |
| 01 | 0 | 01 |
| 10 | 0 | 10 |
| 11 | 0 | 01 |
| 00 | 1 | 01 |
| 01 | 1 | 10 |
| 10 | 1 | 01 |
| 11 | 1 | 00 |

Пример: Мн = 0101

Мт = 11000111

Мтп = 0101001001

Умножение будем осуществлять согласно алгоритму А.

[- Мн]доп = 1.1011

2 Мн = 0.1010

0.0000 

1.1011  = −Mн

1.1011 

1.**11**10 11 ∙ 2-2

0.1010 = 2Mн

0.1000 11 

0.**00**10 0011 ∙ 2-2

0.**00**00 100011 ∙ 2-2  (∙ 2-4)

1.1011  =-Mн

1.1011 100011 

1.**11**10 11100011  ∙ 2-2

0.0101 = Mн

0.0011 11100011 

*Время умножения на два разряда одновременно.*

Появление любой из рассматриваемых пар множителей равновероятно. Следовательно, время умножения на два разряда множителя может быть выражено следующим соотношением:  = ( n/2 + 1) [0,75∙(tсл + tсдв) + (0,25∙tсдв], где n – количество разрядов множителя.

### 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. Может быть выведено общее правило сокращенного умножения:

1. Если цифра множителя bi-1 < r/2 , то + , где  = МH bi.

2. Если цифра множителя bi-1 ≥ r/2 , то + , где = [МH⬝( r - bi )]доп.

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

Пример: Мн = 011

Мт = С49

Мтп = 1457

Можно заранее заготовить кратные множители: МH ,2МH ,4МH поместив их в

дополнительные регистры.

Mн Это позволит сократить время,

необходимое для формирования

частичного произведения ,

2Mн равного от нуля до семи множи-

СМ

мых.

4Mн

Рис. 8. Схема формирования сомножителей при

умножении на четыре разряда множителя.

[+7Мн]доп = 0.10101 [+5Мн]доп= 0.01111 [+4Мн]доп= 0.01100

[-7Мн]доп = 1.01011 [-4Мн]доп= 1.00100

0.0000000 

1.1101011 = −7Mн

1.1101011 

1.**1111**110 1011 ∙ 2-4

0.0001111  = +5Mн

0.0001101 1011 

0.**0000**000 1101 1011  ∙ 2-4

1.1110100  =−4Mн

1.1110100 1101 1011 

1.**1111**111 0100 1101 1011 ∙ 2--4

0.0000011  = +Mн

0.0000010 0100 1101 1011  = Мн∙Мт

## 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 знакосочетания 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∙Мт]доп=2-Мн∙Мт. Для формирования произведения выполним умножение Мн на Мт’ (дополнение)

Мн∙Мт’= Мн ∙(1 - Мт)= Мн - Мн∙Мт.

Таким образом, погрешность Δ умножения равна разности [Мн∙Мт]доп- и Мт∙Мн’

Δ=2-Мн∙Мт- Мн - Мн∙Мт=2-Мн=[-Мн]доп=[ [Мн]доп]доп

и должна быть внесена в полученный результат в качестве поправки.

Пример: Мн = 0,1011 (алгоритм умножения А)

Мт = - 0,1101

Δ= [- МH]доп = 1.0101

[- МT]доп = 1.0011

0.0000 

0.1011 = Мн ⬝ b4

0.1011 

0.**0**101 1 ∙ 2-1

0.1011 = Мн ⬝ b3

1.0000 1  (произошло переполнение)

0**.1**000 01 ∙ 2-1 (коррекция)

0.**0**10 0001  ∙ 2-1 (∙ 2-1) ∙ 2-1

0.**0**010 0001  ∙ 2-1 ((∙ 2-1 )∙ 2-1)∙ 2-1

1.0101 Δ = [- МH]доп  (поправка)

1.0111 0001 =[MH ∙MT]доп

- 1000 1111 MH ∙MT

В этом примере, а также в некоторых примерах следующих ниже происходит переполнение разрядной сетки, в результате которого искажаются и знаковая и значащая часть числа. Для коррекции (восстановления верного значения) производится не модифицированный сдвиг. В результате этого знаковый разряд сдвигается в значащую часть, а знаковый разряд меняет свое значение на противоположное.

3) Mн<0,

Mт>0,

аналогично предыдущему случаю

[Мн∙Мт]доп=2-Мн∙Мт

Мн’∙Мт= (1 - Мн)∙Мт= Мт - Мн∙Мт

Таким образом, погрешность умножения равна разности [Мн Мт]доп  и Мт∙Мн’.

Δ=2-Мн∙Мт- Мт - Мн∙Мт=2-Мт=[-Мт]доп=[ [Мт]доп]доп

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

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

Мн∙Мт = А ∙ В = [ A ∙ b1∙ 2-1 ]доп + [ A ∙ b2∙ 2-2 ]доп+ ... + [A∙ bn∙ 2-n ]доп

На основании теоремы, что сумма дополнительных кодов есть дополнительный код, получаем [ A ∙ b1∙ 2-1 + A ∙ b2∙ 2-2 + ... + A∙ bn∙ 2-n ]доп = [Мн ∙Мт]доп.

В этом случае поправка вводится автоматически на каждом этапе умножения.

Пример: Mн = -0,1011

Mт = 0,1101

[Mн]доп= 1.0101

b1…b4

4

0.00000000 

1.**1**0101000  = [MH ∙b1 ∙ 2-1]доп

1.10101000 

1.**11**010100  = [MH ∙b2 ∙ 2-2]доп

1.01111100 

1.**1111**0101  = [MH ∙b4 ∙ 2-4]доп

1.01110001 [MH ∙MT]доп

- 10001111 MH ∙MT

4) MH<0,

MT<0,

Mн∙Mт= 2 - [Mн∙Mт]доп

[Mн]доп ∙ (1 - Mт) = [Mн]доп - [MH]доп ∙MT = [Mн]доп - [Mн∙Mт]доп

Δ=2 - [Mн∙Mт]gдоп - [Mн]доп + [Mн∙Mт]доп = 2 - [Mн]доп = [[Mн]доп]доп

Пример: Mн = - 0,1011 умножение будем выполнять

Mт = - 0,1101 по алгоритму умножения Г.

[Mн]доп = 1.0101

[Mт]доп = 1.0011

Δ =[-Mн]доп = 0.1011

0.00000000 

1.**111**01010 = [Mн∙b3 ∙ 2-3]доп

1.11101010 

1.**1111**0101 = [Mн ∙ b4 ∙ 2-4]доп

1.11011111 

0.**1011**0000 Δ (поправка)

0.10001111 [Mн∙Mт]доп

Далее коротко остановимся на **умножении целых чисел**. При представлении целых чисел в дополнительном коде знаковый разряд входит в число n разрядов. Следовательно, при умножении целых чисел (в отличие от дробных) в дополнительных кодах знаковый разряд участвует в умножении наряду со значащими. То есть умножение ведется на [Mт]доп , а не на Мт**’**.

1) MH > 0,

MT > 0.

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

2) МH>0,

MT<0,

[Мт]доп = 2n – Мт.

Так как сомножители имеют разные знаки, то произведение Мн∙Мт<0, сле­довательно [Mн∙Мт]доп=22n - Mн∙Мт. Однако при умножении Мн∙[Мт]доп получается Mн∙(2n-Mт) =2n Mн - Mн∙Мт. Следовательно, погрешность в этом случае равна Δ=22n–Mн∙Мт–2n Mн+Mн∙Мт =22n–2n Mн =22n [–Мт]доп=22n[ [Мт]доп]доп.

Пример: Mн = +110

Mт = -101

[Mн]доп = 0.110

[Mт]доп = 1.011

= [- Mн]доп = 1.010

0.000 

0.110 = MH∙b4

0.110 

0.**0**11 0 ∙2-1

0.110  = MH∙b3

1.001 0  (возникло переполнение)

0.**1**00 10 ∙2-1 (коррекция)

0.**0**10 010 ∙2-1

0.110 = MH∙b1

1.000 010  (возникло переполнение)

0.**1**00 0010 ∙2-1 (коррекция)

1.010  (поправка)

1.110 0010 [MH MT]доп

- 001 1110 MH MT

3) МH<0

МT>0

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

a) с вводом поправки в получаемое произведение

[МH]доп = 2n - MH

Как и ранее, требуется получить [Мн∙Мт]доп= 22n - Мн∙Мт. Получаем

(2n - Мн) ∙ Мт= 2n ∙ Мт- Мн∙Мт.

 = 22n - Мн∙Мт - 2n ∙ Мт + Мн∙Мт = 2n(2n - Мт) = [Мт]доп ∙ 2n

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

MH∙MT = A∙B = [A ∙ b1 ∙ 2-1 ]доп + [A ∙ b2 ∙ 2 -2]доп+- ... + [A ∙ bn ∙ 2-n]доп=

=[A ∙ b1 ∙ 2-1 + A ∙ b2 ∙ 2 –2 +- ... + A ∙ bn ∙ 2-n]доп=[Мн∙Мт]доп.

Пример: Мн = -110 Мт = 101 [Мн]доп = 1.010

[Мт]доп = 0.101

b1 ... b4

0.0000000 

0.**0**000000  = [MH ∙ b1]доп ∙ 2-1

0.0000000 

1.**11**01000  = [MH ∙ b2]доп ∙ 2-2

1.1101000 

1.**1111**010  = [MH ∙ b4]доп ∙ 2-4

1.1100010 

4) MH < 0

MT < 0

При этом знакосочетании сомножителей в результате должно быть получено:

[Mн]доп = 2n – Mн,

[Mт]доп = 2n – Mт,

Mн ∙Mт = 22n - [Mн Mт]доп.

При умножении [Mн]доп∙[Mн]доп получается:

[Mн]доп∙[Mн]д =[Mн]доп ( 2n - Mт ) = 2n [Mн]доп - 

Пример: Мн = -110

Mт = -101

[Mн]доп = 1.010

[Mт]доп = 1.011

b1 ... b4

= [[Mн]доп]доп= 0.110

При умножении используем алгоритм Г.

0.0000000 

1.**1**010000  = [MH∙b1]доп∙2-1

1.1010000 

1.**111**0100 = [MH∙b3] доп∙2-3

1.1000100 

1.**1111**010  = [MH∙b4] доп∙2-4

1.0111110 

0.110  (поправка)

0.0011110 MH MT

## Умножение на 2 разряда Мт в дополнительных кодах.

|  |  |  |  |
| --- | --- | --- | --- |
| i | i+1 | i | i+1 |
| 00 | 11 | 00 | 10 |
| 01 | 01 | 01 | 10 |
| +4MH | -MH | +4MH | -2MH |

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.

Таблица 2.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| I ая пара | | i+1ая пара | | Преобраз. Пара | | Алгоритм | |
| х | y | z | − | x’ | y’ | A | Г |
| 0 | 0 | 0 |  | 0 | 0 | 1 | 6 |
| 0 | 0 | 1 |  | 0 | 1 | 2 | 7 |
| 0 | 1 | 0 |  | 0 | 1 | 2 | 7 |
| 0 | 1 | 1 |  | 1 | 0 | 3 | 8 |
| 1 | 0 | 0 |  | 1 | 0 | 4 | 9 |
| 1 | 0 | 1 |  | 0 | 1 | 5 | 10 |
| 1 | 1 | 0 |  | 0 | 1 | 5 | 10 |
| 1 | 1 | 1 |  | 0 | 0 | 1 | 6 |

В табл. 2 приведены преобразования i и i+1 пары МT .действия, осущестляемые при выполнении, например, алгоритмов А и Г.

(1)  (6) 

(2)  (7) 

(3)  (8) 

(4)  (9) 

(5)  (10) 

Если i+1 пара имеет старшую цифру 1, то умножение на эту пару будет соответствовать вычитанию одного или двух множителей.



Рис. 9. Логическая схема формирования сигнала сдвига частичной суммы.

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. Если на умножение приходит отрицательный множитель, то при преобразовании его старшей пары происходит формирование дополнительной пары:

Мт = -10110

[Мт]доп= 1.0 10 10

[Мт]допп= 01 01 01 10

При умножении на дополнительную пару (01) происходит вычитание одного множителя , а затем он должен быть добавлен в качестве поправки. Таким образом тратится два такта на выполнение взаимноисключающих операций.

Пример:

Мн = + 0101 умножение выполним согласно

MT = + 1100111 алгоритма Г.

[MH]доп = 0.0101

[MT]доп = 0.1 10 01 11

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

 = 10 10 10 01

[-2Mн]доп  = 1.0110 [2Mн]доп  = 0.1010

0.0000 00000000 

0.**00**10 10000000  = 2MH∙2-2

0.0010 10000000 

1.**1111** 01100000 = -2MH∙2-4

0.0001 11100000 

0.**0000 00**101000  = 2MH∙2-6

0.0010 00001000 

1.**1111 1111**1011  = -MH∙2-8

0.0010 00000011 = [Mн Mт]доп

Пример:

Mн= - 0101 умножение выполним согласно

Mт = -1100111 алгоритма Б.

[Mн]доп  = 1.1011

[Mт]доп = 1.0 01 10 01

 = 1.0 10 10 01

[2Mн]доп  = 1.0110 [-2Mн]доп  = 0.1010

0.000000 0000 

1.111111 1011 = MH

1.111111 1011 

0.000010 10**00** = -2MH ∙ 22

0.000010 0011 

1.110110 **0000** = 2MH ∙ 24

1.111000 0011 

0.1010**00 0000**  = -2MH ∙ 26

0.100000 0011 =MH MT

## Матричные методы умножения.

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

Пусть имеем сомножители: Мн = А = аn ... a2 a1

Мт = B = bn ... b2 b1

Рассмотрим схему традиционного (“школьного”) алгоритма умножения (Б).

А = аn ... a2 a1

\*  B = bn ... b2 b1

anb1 ... a3b, a2b1, a1b1

+anb2 **. . .** a2b2, a1b2

+ **. . .**

anbn ... a2bn, a1bn

C = C2n **. . . .** C2  C1

Рассмотренная схема умножения может быть представлена в виде матрицы.

Таблица 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | an | **. . .** | a2 | a1 |
| b1 | an b1 | **. . .** | a2 b1 | a1 b1 |
| b2 | an b2 | **. . .** | a2 b2 | a1 b2 |
| **.**  **.**  **.** |  | **. . .** |  |  |
| bn | an bn | **. . .** | a2 bn | a1 bn |

Каждый элемент ai bj ( i, j = 1, n) принимает значение 0 или 1. Произведение A∙B может быть получено, если суммировать элементы матрицы (по диагонали).



+

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 качестве их могут быть использованы одноразрядные двоичные сумматоры и полусумматоры.

На рис. 10 приведена структурная схема устройства умножения для реализации матричного алгоритма.

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 качеству оборудования становятся не столь строгими.

## Машинные методы деления

Деление - простое многократное вычитание вначале из делимого, затем из остатков. Введем обозначения, используемые ниже: Дм - делимое, Дт - делитель, Аi – очередной (i-ый) 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 цифра частного. Операция деления является операцией дающей не  всегда точный результат. Поэтому признаком окончания операции деления может быть достижение заданной точности (по числу сдвиговых сигналов). Если при выполнении деления получен нулевой i-ый остаток, то деление прекращается и в в оставшиеся разряды частного записываются нули. Первым шагом деления двух чисел машиной 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-ой цифры частного, число сдвигается вправо на один разряд с добавлением единицы к порядку, равному разности порядков делимого и делителя.

### Деление чисел в прямых кодах.

Алгоритм *деления с восстановлением остатка* состоит в следующем.

1. Выполняется пробное вычитание с формированием первого остатка A1=[Дм]доп+[-Дм]доп. Далее, если А1 < 0, то в первый разряд, расположенный слева от запятой заносится ноль (0, ), иначе единица (1, ) – переполнение и переход к пункту 5.

2. Если Аi < 0, то восстанавливаем предыдущий остаток Ai=Ai+[Дм]доп.

3. Формирование очередного остатка. Ai+1=Ai∙2+[-Дм]доп, то в очередной разряд частного справа от запятой записывается ноль (Чт(n)=0), иначе записывается единица (Чт(n)=1).

4. Если достигнута заданная точность частного или получен нулевой остаток Ai+1, то процесс деления окончен и переход к пункту 5, иначе переходим к пункту 2 алгоритма.

5. Окончание алгоритма.

Из рассмотренного алгоритма видно, что:

1) необходимо затрачивать время на восстановление остатка;

2) процесс деления не регулярный, в зависимости от делимого и делителя

частное будет содержать нулей больше или меньше, и чем больше нулей, тем больше требуется времени на восстановление остатков.



Как видно из примера, для получения остатка Аi+2 необходимо выполнить

Аi+2 = ( Ai+1 + ДT ) ∙ 21 - ДT = Ai+1 ∙ 21 + 2ДT - ДT = Ai+1 ∙ 21 + ДT.

Из этого следует, что восстанавливать остаток не обязательно. Достаточно сдвинуть полученный отрицательный 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 A1=[Дм]доп+[-Дм]доп. Далее, если А1 < 0, то в первый разряд, расположенный слева от запятой заносится ноль (0, ), иначе единица (1, ) – переполнение и переход к пункту 5.

2. Формирование очередного остатка. Если Аi < 0, то Ai+1=Ai∙2+[Дм]доп, иначе Ai+1=Ai∙2+[-Дм]доп.

3. Если Аi+1 < 0, то в очередной разряд частного справа от запятой записывается ноль (Чт(n)=0), иначе записывается единица (Чт(n)=1).

4. Если достигнута заданная точность частого или получен нулевой остаток Ai+1, то процесс деления окончен и переход к пункту 5, иначе переходим к пункту 2 алгоритма.

5. Окончание алгоритма.



### Деление чисел в дополнительных кодах.

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

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 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 A1=[Дм]доп+[Дм]доп, иначе A1=[Дм]доп+[-Дм]доп. Далее формируется первый разряд, расположенный слева от запятой - ноль (0, ) если знак А1 ≠ знаку Дт, иначе единица (1, ).

2. Формирование очередного остатка. Если знак Аi ≠ знаку Дт, то Ai+1=Ai∙2+[Дм]доп, иначе Ai+1=Ai∙2+[-Дм]доп.

3. Если знак Аi+1 ≠ знаку Дт, то в очередной разряд частного справа от запятой заносится ноль (Чт(n)=0), иначе единица (Чт(n)=1).

4. Если достигнута заданная точность частого или получен нулевой остаток Ai+1, то процесс деления окончен, иначе переходим к пункту 2 алгоритма.

Пример: Дм= - 0.1011 [ Дм ]доп = 1.0101

Дт = 0.1101 [ Дт ]доп = 0.1101 [-Дт ]доп = 1.0011

На деление Дм и Дт придут в дополнительном коде

****

## Методы ускорения деления.

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

При этом Дт выбирается (формируется) таким образом, чтобы после запятой шла единица, то есть он нормализован. Если очередной остаток получился настолько мал, что после “,” следует r+1 нулей, то в частное может быть записано r ”0” или ”1”, а остаток может быть сдвинут на r разрядов влево.

Итак, для осуществления отмеченных методов ускорения кроме УУ делением требуется логическая схема, осуществляющая 2 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…

## Двоично-десятичные коды

Пусть число A представлено в системе счисления с основанием r:



Цифры ai будем представлять двоичными разрядами d1,d2,…,dm. Каждому двоичному разряду припишем веса p1,p2,…,pm. Тогда каждый разряд ai числа A будет иметь вид , а все число

, (4)

где n и m определяют общее число двоичных разрядов.

Если каждый разряд числа имеет вес и при r≠2k не выполняется равенство pk=r ∙ pk-1, то системы принято называть *взвешенными*. Количество разрядов m должно удовлетворять выражению m ≥ log2r. Если десятичное число записано в виде (4), то будем говорить, что число представлено в двоично-десятичном коде. Наибольшее распространение из них получили коды, в которых десятичная цифра представляется двоичной тетрадой (BCD коды). Существует множество способов кодирования десятичных цифр. Существенным при этом является простота представления инверсных кодов и простота выделения (формирования) сигнала переноса из цифры в цифру.

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

* Четность, состоит в том, что четным десятичным цифрам соответствуют только четные двоичные коды, либо наоборот, что обеспечивает эффективность операций округления, умножения и деления чисел в BCD кодах.
* Дополнительность, заключается в том, что сумма двоичного кода и инверсного ему кода любой десятичной цифры должна быть равна 9. Это обеспечивает эффективность операции алгебраического сложения в BCD кодах.
* Упорядоченность, то есть большей десятичной цифре соответствует большая тетрада, и наоборот.
* Единственность представления десятичной цифры двоичной тетрадой.
* Взвешенность, то есть 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 

a = 1011  11, то а + а = 1111 = 15

В табл. 4 показаны различные способы кодирования десятичных цифр.

Таблица 4.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Дестичные  цифры | Эквивалент в коде | | | | | | |
| 8421 | 2421 | 7421 | 5211 | 8421+3 | 3а+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 |

### Суммирование чисел с одинаковыми знаками в коде 8421.

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 коде 8421 возможны три случая:

1) ( a + b ) ≤ 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 ≤ ( a + b ) ≤ 15. Если результат сложения двух чисел попадает в данный диапазон чисел, то возможны два случая результирующей тетрады.

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 логической схемой. Условием для формирования единицы переноса является возникновение запрещенной комбинации (наличие единицы в разрядах с весом 8 и 4 или 8 и 2). Однако тетраду надо освободить от десяти избыточных единиц. Это тоже делается принудительно добавлением 0110 (шестерки), что приводит к возникновению шестнадцатеричного переноса. Этот перенос игнорируется. Схема формирования принудительного переноса приведена на рис.11.

3) ( a + b ) ≥ 16. Здесь в процессе суммирования возникает шестнадцатеричный перенос, в результате которого тетраду покидают с десятком и те шесть единиц, которые принадлежат тетраде. Чтобы восстановить верное значение этой тетрады, необходимо к ней добавить 0110 (шесть).

перенос

1

x x x x

&

&

8 1000

9 1001

17 1 0001

0110 коррекция(+ 6) 0111

Рис. 11. Схема определения запрещенной комбинации

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

* если при потетрадном сложении перенос в соседнюю старшую тетраду не возникает, то результат суммирования не требует коррекции.
* 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***.

Пример: сложить числа А=169 и В=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



### Сложение чисел с разными знаками.

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 до 9. Следовательно, необходимо убрать разницу. Один из приемов формирования обратного BCD-кода состоит в добавлении во все тетрады отрицательного числа 0110, затем их инверсия.

При сложении чисел с разными знаками возможны следующие случаи.

1) a - b ≥ 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 пятнадцать единиц. Эти 15 единиц находятся и в сумме. Но благодаря шестнадцатеричному переносу из тетрады уходит 16 единиц ( 15+1 - которая и восстанавливается добавлением по цепи циклического переноса ).

1. 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. Но 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.

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, таким образом, это соответствует заему в нее 16 единиц (вместо необходимых 10). Следовательно, из нее необходимо удалить лишние шесть единиц, Для этого в тетраду добавляется 10 - дополнение шести до шестнадцати:

0. 0010 0000 1111

1010

0. 0010 0000 1001

+ 2 0 9

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

Таким образом, в тетраду производится заем, если результат:

* положительный и из тетрады нет переноса;
* отрицателен и из тетрады есть перенос.

### Двоично-десятичные коды с избытком 3

Иначе говоря, это коды чисел из системы ( 8421 + 3 ). В этом коде каждая десятичная цифра ai представляется в виде двоичного эквивалента суммы ai+3. В отличие от кода 8421 код 8421+3 – самодополняющийся, но не имеющий свойства взвешенности. Применяется наиболее часто в десятичной арифметике, так как при выполнении двоичного суммирования легко выделить десятичный перенос.

Возможны следующие два случая сложения чисел в коде 8421+3:

1. a + b ≤ 9 ; [ ( a + 3 ) + ( b + 3 ) ] ≤ 15.

И следовательно, в тетраде суммы будет лишних 6 единиц. Чтобы тетрада суммы осталась тоже с избытком 3, нужно вычесть 3.

1. a + b ≥ 10; [ ( a + 3 ) + ( b + 3 ) ] ≥ 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 коде 8421.

Пример: Выполнить сложение чисел 169 и 378 в коде 8421+3.

0.0100 1001 1100

A = 169 0.0110 1010 1011

B = 378 0.1011 01000111

A + B = 547 0011 0011 0011

0.1000 0111 1010

8 7 10

Пример : Выполнить вычитание из числа 378 числа 169 в коде 8421+3.

A = 378 0.0110 1010 1011

B = 169 1.1011 0110 0011

A - B = 209 1 0.0010 0000 1110

циклический перенос1

0.0010 0000 1111

+ 0011 -0011 -0011

0.0101 0011 1100

5 3 12

Пример: Выполнить вычитание из числа 169 числа 378 в коде 8421+3.

A = 169 0.0100 1001 1100

B = 378 1.1001 0101 0100

A - B = -209 1.1101 1111 0000

-0011 -0011 +1100

1.1010 1100 0011

- 0101 0011 1100

5 3 12

*Правило*: если из тетрады был перенос, надо добавить +0011, если переноса не было, – 0011 (добавить 1100), независимо от знака слагаемых и знака суммы.

### 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 коррекция результата не требуется.

Результат должен быть без избытка.

Пример :

A = 169 0.0111 1100 1111

B = 378 0.0011 0111 1000

A + B = 547 0.1001 0100 0111

0110 .

0.0101 0100 0111

Суммирование чисел с разными знаками производится в коде 8421, но в сущности тоже с избытком 6.

## Система счисления в остаточных классах (СОК)

Поиски новых путей повышения эффективности выполнения арифметических операций привели к заключению, что в рамках обычной позиционной системы счисления ускорения операций добиться сложно. Следует отметить, что позиционные системы счисления обладают существенным недостатком – наличием межразрядных связей, влияющих на способы реализации арифметических операций. В конечном итоге это приводит к усложнению аппаратуры и снижению быстродействия. Оказалось, что арифметика в которой межразрядные связи отсутствовали бы может быть построена на основе непозиционной системы счисления, в частности системы счисления в остаточных классах. В системе остаточных классов числа представляются остатками от деления на выбранную систему оснований, и все рациональные операции могут выполняться параллельно над цифрами каждого разряда в отдельности. В то же время системе остаточных классов присущи некоторые недостатки: ограниченность действий этой системы полем целых положительных чисел, трудность определения соотношений чисел по величине, определения выхода результата операции из диапазона и некоторые другие.

***Определение***. Если задан ряд положительных целых чисел p1, p2, ..., pn, называемых в дальнейшем основаниями системы, то под системой счисления в остаточных классах будем понимать такую систему, в которой целое положительное число представляется в виде набора остатков ( вычетов ) по выбранным основаниям

N = (α1, α2, ..., αn),

причем образование цифр αi осуществляется следующим образом:

αi = N - , для i = 1, 2, ..., n, (5)

то есть цифра i-го разряда αi числа N есть наименьший положительный остаток от деления N на pi.

Таким образом, в СОK в отличие от обобщенной позиционной системы счисления образование цифры каждого разряда производится независимо друг от друга. Очевидно, что αi < pi.

В теории чисел доказано, что если числа pi взаимно простые между собой, то описанное цифрами α1,α2, ..., αn представление числа N -- единственно.

Диапазон представимых чисел в СОK равен:

 = p1, p2, ..., pn.

Рассмотрим правила выполнения операций сложения и умножения в СОK в случае, если числа и результат находятся в диапазоне [O, ].

Пусть операнды A и B представлены соответственно остатками αi и βi по основаниям pi, i=1,2, ... ,n. Результаты операций A+B и A∙B представлены остатками γi и δi по тем же основаниям pi, то есть

A=(α1 α2 ... ,αn),

B=(β1, β2, ... ,βn),

A+B=(γ1, γ2, ... ,γn),

A∙B=(δ1, δ2, ... ,δn),

при этом A < B < A+B < A∙B <

γi ≡ αi + βi (mod pi),

то есть γi сравнимо с αi + βi по модулю pi;

δi ≡ αi βi (mod pi).

При этом:

γi ≡ αi + βi - ,

δi ≡ αi βi - .

Учитывая (5), отсюда следует:

γi = A+B - . (6)

Из представления A и B следует:

А = ki pi + αi,

B = li pi + βi,

где ki и li - целые неотрицательные числа.

Тогда A + B = (ki + li) pi + αi + βi ,

=ki + li +,

A + B = (ki + li)pi + .

Подставляя полученные выражения в (6), получим:

γi ≡ αi + βi - . (7)

В случае умножения:

δi ≡ αi βi - .

Аналогично сложению (A + B) получим:

A∙B = ki li pi2 + (αi li +βi ki)pi + αi βi,

= ki li pi + αi li +βi ki + .

Таким образом:

δi ≡ αi βi - . (8)

Пример: Сложить числа А = 23 и В = 48 в СОК.

Пусть основаниями системы являются p1 = 3, p2 = 5, p3 = 7.

По выбранным основаниям числа А и В в СОК примут вид:

А = 23 = (2, 3, 2),

B = 48 = (0, 3, 6),

получим А + В = (2, 1, 1).

Легко проверить, что 23 + 48 = 71,

а число 71 = (2, 1, 1) в СОК.

Пример. Умножить число А=14 на В=8 в СОК

А=14=(2, 4, 0)

B=8=(2, 3, 1)

в соответствии с (4) получим:

γ1 = 4 -  = 4 - 3 = 1,

γ2 = 12 -  = 12 - 2∙5 = 2,

γ3 = 1 -  = 1 - 0.7 = 1,

таким образом А∙В = 122 = (1, 2, 1).

Охарактеризуем в общих чертах достоинства и недостатки введенной СОK.

***Достоинства****:*

* -независимость образования разрядов числа, в силу чего каждый разряд несет информацию обо всем исходном числе, а не о промежуточном числе, получившемся в результате формирования младших разрядов (позиционная система счисления). Отсюда вытекает независимость разрядов числа друг от друга и возможность их параллельной обработки, что в свою очередь в дальнейшем позволяет вводить принципиально новые методы арифметического контроля. При введении дополнительного контрольного основания остаток, взятый по этому основанию, при его избыточности позволяет контролировать и исправлять ошибки в цифрах по рабочим основаниям;
* малоразрядность остатков, представляющих число. Ввиду малого количества кодовых комбинаций открывается возможность построения табличной (не вычислительной) арифметики.

*Недостатки:*

* невозможность визуального сопоставления чисел, так как внешняя запись числа не дает представления о его величине;
* отсутствие простых признаков выхода результата операций за пределы [0,ρ];
* ограниченность действий системы сферой целых положительных чисел;
* получение во всех случаях точного результата исключает возможность приближенных вычислений, округлений.

### Представление отрицательных чисел в СОК

Рассмотрим правила выполнения операций вычитания в системе остаточных классов для чисел А и В, удовлетворяющих условию А-В ⊆[0,];

А = (α1, α2, ... ,αn),

В = (β1, β2, ... ,βn),

A-B = (γ1, γ2, ... ,γn),

при этом А<ρ, B< ρ, 0<=А-В< ρ.

Как и ранее, получаем для А-В

γi = αi - βi - ,

γi ≡ αi - βi (mod pi).

Операция вычитания при положительном результате выполняется вычитанием соответствующих цифр разрядов, в результате приводится наименьший положительный остаток.

При отрицательной разности цифр берется ее дополнение к основанию. При этом знак результата в результате никак не отражен.

Возникает необходимость ввести специальным образом знак в представление числа и определить правила выполнения операций, обеспечивающих получения не только величин, но и знака результата.

Рассмотрим варианты введения отрицательных чисел.

Пусть p1, p2, ... pn - основания системы счисления.

 = p1, p2, ... ,pn  - диапазон представимых чисел.

Пусть p2 =2. Обозначим через В СОK Р=(1, 0, 0, ... , 0).

Будем оперировать числами, лежащими в диапазоне 0≤ |N| <P.

Примем в качестве нуля число Р и представим положительные числа N= |N| как N' = P + |N|, отрицательные числа N = - |N| как N' =P - |N|.

Таким образом, N' =P+N - искусственная форма (общая форма представления + и - чисел).

Следовательно, мы всегда будем иметь дело с положительными числами, так как |N|<P.

Но числа в интервале [0,P) в искусственной форме будут отображать отрицательные, а в интервале [P,) - положительные числа.

Операцию сложения и вычитания можно выполнять следующим образом:

N′1 = P + N1,

N′2 = P + N2,

N′1 + N′2 = P + N1 + P + N2 = 2P + (N1 + N2).

Для суммы N1 + N2  искусственная форма:

(N1 + N2)′ = P + (N1 + N2),

(N′1 + N′2) = N′1 + N′2 - P

или так как P = (1, 0, 0, ..., 0), то (N1 + N2) ′ =N′1 + N′2 + P. (9)

Пример. Пусть p1 = 2, p2 = 3, p3 = 5, p4 = 7.

P = 3∙5∙7 = 105

N1 = 17, N2 = 41

N′1 = (1, 0, 0, 0) + (1, 2, 2, 3) = (0, 2, 2, 3),

N′2 = (1, 0, 0, 0) + (1, 2, 1, 6) = (0, 2, 1, 6),

согласно (9) (N1 + N2) ′ = (0, 2, 2, 3) + (0, 2, 1, 6) + (1, 0, 0, 0) = (1, 1, 3, 2) - искусственная форма N1 + N2, что можно проверить, перейдя и десятичной системе счисления.

(17 + 41)′ = (58)′ = 105 + 58 = (1, 0, 0, 0) + (0, 1, 3, 2) = (1, 1, 3, 2)

Пример. N1 = 17, N2 = 41.

N′1 = (0, 2, 2, 3),

N′2 = (1, 0, 0, 0) - (1, 2, 1, 6) = (0, 1, 5, 1),

(N1 + N2)′ = (0, 2, 2, 3) + (0, 1, 5, 1) + (1, 0, 0, 0) = (1, 0, 1, 4),

(17 - 41)′ = (-24)′ = 105 - 24 = (1, 0, 0, 0) - (0, 0, 4, 3) = (1, 0, 1, 4).

Переход из положительного числа в отрицательное и обратно, то есть образование его дополнительного кода производится вычитанием данного числа из числа (1, p1, p2, ... ,pn).

+41 = (1, 2, 1, 6)

- 41 = (1, 3, 5, 7) - (1, 2, 1, 6) = (0, 1, 4, 1) = 64.

Следует отметить, что если вычитаемое уже было представлено в искусственной форме, то для получения дополнительного кода надо его вычитать из (2,p1,p2, ..., pn).

## Контроль работы цифрового автомата

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.

Третий вид ошибок является объектом для системы контроля. 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 заключаться в выполнении ”двойного счета” и последующего сравнения результатов, в проверке соотношений вычисляемых функций (sin2x+cos2x=1), в контроле результата при изменении шага вычисления и так далее. Однако эти методы направлены на выявление факта появления ошибки в вычислении, но не определяют место, где она произошла.

### 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 ω=ρk(1-ρ)n-k.

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

.

Тогда полная вероятность искажения информации

.

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 суммой по модулю 2.

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

В теории кодирования [5] показано, что систематический код обладает способностью обнаружить ошибки только тогда, когда минимальное кодовое расстояние для него больше или равно 2t:

dmin≥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 dmin≥ 2t+1.

### Обнаружение и исправление одиночных ошибок путем

### использования дополнительных разрядов

Рассмотрим возможность использования дополнительных (контрольных) разрядов для обнаружения и исправления ошибок. Эта возможность заключается в том, что к n информационным разрядам добавляется один контрольный разряд. В него записывается 0 или 1 таким образом, чтобы для каждого из передаваемых чисел сумма разрядов по модулю 2 была бы равна 0 (кодирование по методу четности) или 1 (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 так далее) числа разрядов одновременно. Пример реализации метода контроля по методу четности-нечетности приведен ниже в табл. 5.

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

Таблица 5.

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

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | a1 | a2 | a3 | A4 | a5 | k1 |  |
|  | a6 | a7 | a8 | a9 | a10 | k2 |  |
|  | a11 | a12 | a13 | a14 | a15 | k3 |  |
|  | a16 | a17 | a18 | a19 | a20 | k4 |  |
|  | a21 | a22 | a23 | a24 | a25 | k5 |  |
|  | k6 | k7 | k8 | k9 | k10 |  |  |

Если ошибка произошла в разряде as (единица изменилась на ноль или наоборот), то при проверке на четность (нечетность) сумма по i-й строке и j-му столбцу (на пересечении которых находится элемент as) 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 as на противоположное можно исправить возникшую ошибку.

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

### Коды Хемминга

Американский ученый Р. Хемминг предложил способ кодирования информации, позволяющий не только обнаруживать, но и исправлять ошибки при передаче одиночного слова любой разрядности. 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 таким образом, 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 2k≥(m+n+1).

2. (m+k) – разрядные позиции нумеруются от единицы до (m+k), начиная от младшей значащей. Контрольные разряды k обознаются P0, P1, P2, …,Pk-1 и помещаются в разряды, имеющие номера 1,2,4,8, …,2k-1 (m+k) – разрядного числа. Остальные m разрядов могут быть размещены в любом порядке между контрольными разрядами.

3. Контрольные разряды P0, P1, P2, …,Pk-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 | **…** |  |
|  | **…** |  |  | **.** | **.** | **.** |  |  |  |  |  |  |  |  |  |  |

P0 выбрано с таким расчетом, чтобы в позициях 1, 3, 5, 7, 9, 11 … число единиц каждого слова было четным, P1 – выбрано для того, чтобы выполнялось условие четности в разрядах 2, 3, 6, 7, 10, 11, 14, 15 …, аналогично P2 контролирует позиции 4, 5, 6, 7,12,13,14,15,20… и P3 для разрядов 8, 9,10,11,12,13,14,15,24,25…

На основании рассмотренных правил в таблице показаны семиразрядные коды. Контрольные разряды обозначены P0, P1 и P2 и помещены в позициях 1, 2 и 4.

Таблица 6.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Разряды | | | | | | |  |
|  | Число | 7 | 6 | 5 | 4 | 3 | 2 | 1 |  |
|  |  | A | B | C | P2 | D | P1 | P0 |  |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |
|  | 1 | 0  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 |  |

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. 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 и которую необходимо исправить.

Пусть, например, передается число шесть 0110011, а принимается в виде 0110111, то есть произошла ошибка в третьем разряде. Выполняя контроль на четность с помощью разрядов P0, P1 и P2, находим:

*Контрольное число*

P0  (1, 3, 5, 7) = (1, 1, 1, 0) нечетность 1

P1  (2, 3, 6, 7) = (1, 1, 1, 0) нечетность 1

P2  (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 Pi выбран так, чтобы проверять четность той совокупности разрядных позиций, чьи контрольные числа содержат единицу в позиции 2i.

# Логические основы вычислительной техники

## Двоичные переменные и булевы функции

Для формального описания устройств вычислительной техники при их анализе и синтезе используется аппарат *алгебры логики*. Алгебру логики называют также булевой алгеброй. Основными понятиями алгебры логики являются двоичные переменные и переключательные (булевы) функции.

*Двоичные переменные* могут принимать только два значения 0 (ложь) и 1 (истина), и обозначаются символами x1, x2, … , xn. Двоичные (логические, булевы) переменные являются аргументами булевых (переключательных) функций.

Функция f, зависящая от n переменных x1, x2, ...., xn, называется *булевой*, или переключательной, если функция f и любой из ее аргументов xi, (i = 1...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* аргументов можно рассматривать как *n*-местную алгебраическую операцию на множестве *B*. При этом алгебра *<B;Ω*>, где *Ω* – 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. 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).

x1 x2 ... xn-1 xn 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 0 f(1,1,...,0,0)

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 аргументов могут принимать значения 0 и 1 и обозначаются f(x1,x2, … ,xn). Булевы функции могут служить аргументами более сложных логических функций.

## Способы задания булевых функций

Для задания произвольной булевой функции широко используются **табличный (матричный**) и **аналитический** способы. При табличном способе булева функция f (х1, ...,хn) задается таблицей истинности (табл. 7), в левой части которой представлены все возможные двоичные наборы длины n, а в правой указывается значения функции на этих наборах.

Таблица 7.

|  |  |  |
| --- | --- | --- |
| №  набора | х1х2х3 | f |
| 0 1 2 3 4 5 6 7 | 000 001 010 011 100 101 110 111 | 0 1 0 0 1 1 0 1 |

Под двоичным набором понимается совокупность значений аргументов х1,х2, ...,xn булевой функции f. Двоичный набор имеет длину n, если он представлен n цифрами из множества {0,1}. В табл. 7 представлены все двоичные наборы длины 3. Иногда двоичные наборы из таблицы истинности булевой функции удобно представлять их номерами. Запишем аргументы х1,х2, ...,xn 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. Булевы функции, зависящие от большого числа переменных, задавать таблицей истинности неудобно в силу ее громоздкости. Например, таблица истинности булевой функции 8 переменных будет содержать 28 = 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: х1, х2, ..., хj-1 и хj, хj+1, ..., хn. Переменными x1, x2, ..., xn отмечают строки таблицы истинности, задавая в каждой строке значение соответствующего двоичного набора длины j-1. Переменными xj, xj+i, ..., xn отмечают ее столбцы, задавая в каждом столбце значения соответствующего двоичного набора длины n-j+1. Значение функции записывается в клетке на пересечении соответствующей строки и столбца (табл. 8).

Таблица 8 .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x1,x2,...xj-1 | xj, xj+1, ...,xn | | | |
| 00...0 | 0...1 | ... | 11...1 |
| 00...0 |  |  |  |  |
| 00...1 |  |  |  |  |
| ... |  |  |  |  |
| 11...1 |  |  |  |  |

При аналитическом способе булева функция задается формулой, то есть аналитическим выражением, построенным из операций булевой алгебры. Аналитический способ задания булевых функций занимает особое место в проектировании цифровых автоматов. Фактически все преобразования над булевыми функциями, необходимые для построения цифровых автоматов, ведутся на аналитическом уровне.

## Основные понятия алгебры логики



Существует не более чем различных булевых функций n перемен-ных. К этому выводу легко прийти, пользуясь простыми комбинаторными рассуждениями, и вспомнив, что на каждом из 2n наборов функции могут принимать два значения.

Функций от одной переменной четыре: это *константа* 0, *константа* 1, *тождественная функция*, то есть функция, значение которой совпадает с аргументом и функция *отрицания* значение которой противоположно значению аргумента. Отрицание будем обозначать x.

*x* 0 *x x* 1

0 0 0 1 1

1 0 1 0 1

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

Переменная *xi* называется *фиктивной* (несущественной) переменной функции *f*(*x*1*,···,xn*), если

*f*(*x*1*,···,xi-*1,0*,xi+*1*,···,xn*) *= f*(*x*1*,···,xi-*1,1*,xi+*1*,···,xn*)

для любых значений *x*1*,···,xi-*1*,xi+*1*,···,xn*. Иначе переменная *xi* называется *существенной*.

Функции двух переменных представлены в табл. 9 .

Таблица 9.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| х1х2 | f0 | f1 | f2 | f3 | f4 | f5 | f6 | f7 | f8 | f9 | f10 | f11 | f12 | f13 | f14 | f15 |
| 00 01 10 11 | 0 0 0 0 | 0 0 0 1 | 0 0 1 0 | 0 0 1 1 | 0 1 0 0 | 0 1 0 1 | 0 1 1 0 | 0 1 1 1 | 1 0 0 0 | 1 0 0 1 | 1 0 1 0 | 1 0 1 1 | 1 1 0 0 | 1 1 0 1 | 1 1 1 0 | 1 1 1 1 |

Отметим наиболее часто используемые функции из числа приведенных в таблице:

f0 (x1, x2) = 0 - тождественный ноль (константа 0);

f1 (x1, x2) = x1 ∙ x2 – конъюнкция (логическое произведение, И). Иногда употребляется знак & или /\:

f3 (x1, х2) = x1 - повторениеx1;

f5 (x1, x2) = x2- повторение x2;

f6 (x1, x2) = x1 x2 - сложение по модулю 2 или сумма mod 2;

f7 (х1, х2) = x1 + x2 - дизъюнкция (логическое сложение, ИЛИ) или знак V;

f8 (x1, x2) = x1 *↓* x2 - функция Вебба (стрелка Пирса, ИЛИ-НЕ);

f9 (х1, х2) = x1 ~ x2 - эквивалентность;

f13(x1, x2) = x1 → x2 - импликация;

f14(x1, x2) = x1 \ x2 - штрих Шеффера (И-НЕ);

f15(x1, x2) = 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 при возведении в степень и извлечении корня, остаются неизменными, если принять справедливость равенств 1·1= 1 и 0·0= 0. Операции вычитания и деления не рассматриваются и не допускаются.

**Логическое отрицание** (**функция НЕ**). Логическим отрицанием высказывания x называется такое сложное высказывание f1(x), которое истинно, когда x ложно, и наоборот. Функция НЕ записывается следующим образом f1=x. Реализующий функцию НЕ элемент приведен на рис. 13а.

**Логическое умножение (конъюнкция)**. Конъюнкция (функция И) двух переменных x1 и x2 это сложное высказывание, которое истинно только тогда, когда истинны x1 и x2, и ложно для всех остальных наборов переменных. Логическая функция конъюнкции имеет вид f=x1·x2. Для обозначения операции конъюнкции используются также символы & и Λ. Функция логического умножения (И) от n переменных имеет вид f2=(x1, x2, …, xn)= x1·x2· … ·xn = Λ xi. Элемент, реализующий операцию логического умножения, изображен на рис. 13б.

**Логическое сложение (дизъюнкция)**. Дизъюнкция (функция ИЛИ) двух переменных x1 и x2 – это сложное высказывание, которое истинно тогда, когда истинна хотя бы одна из переменных x1 и x2, и ложно, когда они обе ложны. Логическая функция дизъюнкции имеет вид f=x1+x2. Для обозначения операции дизъюнкции используется также символ V. Функция логического сложения (ИЛИ) от n переменных имеет вид f2=(x1, x2, …, xn)= x1+x2+ … +xn = V xi. Элемент, реализующий операцию логического сложения, изображен на рис. 13в.

**Отрицание конъюнкции (операция Шеффера)**. Отрицание конъюнкции (функция И-НЕ) двух переменных x1 и x2 – сложное высказывание, ложное только при истинности обоих аргументов x1 и x2. Логическая функция И-НЕ имеет вид f=x1·x2. Элемент, реализующий указанную операцию, изображен на рис. 13г и называется элементом Шеффера или элементом И-НЕ.

**Отрицание дизъюнкции (операция Пирса (Вебба))**. Отрицание дизъюнкции (функция ИЛИ-НЕ) двух переменных x1 и x2 – сложное высказывание, истинное только тогда, когда оба аргумента принимают ложное значение. Логическая функция ИЛИ-НЕ имеет вид f=x1+x2. Элемент, реализующий указанную операцию, изображен на рис. 13д и называется элементом Пирса или элементом ИЛИ-НЕ.

**Сложение по модулю 2 (исключающее ИЛИ)**. Сложение по модулю два это сложное высказывание, которое истинно только тогда, когда истинна только одна из переменных x1 и x2. Логическая функция ”сумма по модулю два” имеет вид f=x1⊕x2. Если число переменных n>2, то функция истинна на тех наборах, в которых число единиц нечетно. Элемент, реализующий операцию сумма по модулю два, изображен на рис. 13ж.

1

⊕

&

1

1

&

a) б) в) г) д) ж)

рис. 13. Схемы логических элементов.

**Импликация**. Это высказывание, принимающее ложное значение только в случае если x1 истинно и x2 ложно.

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

*Суперпозицией булевых функций f*0 и *f*1*,...,fn* называется функция *f*(*x*1*,...,xm*) *= f*0(*g*1(*x*1*,...,xm*)*,...,gk*(*x*1*,...,xm*)), где каждая из функций *gi*(*x*1, *...,xm*) либо совпадает с одной из переменных (тождественная функция), либо – с одной из функций *f*1*,...,fn*.

Функция *f*(*x,y*) *= ¬*(*x & y*) является суперпозицией функций *¬* и &. Функция *g*(*x,y*) *= x*  (*x*  *y*) является суперпозицией функций  и . Функция *h*(*x,y,z*) = (*x & y*)  *z* является суперпозицией функций  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 следует отметить:

* одна и та же функция может быть представлена разными формулами;
* каждой формуле соответствует своя суперпозиция и, следовательно, своя схема соединений элементов;
* между формулами представления булевых функций и схемами, их реализующими, существует взаимно однозначное соответствие.

Очевидно, среди схем, реализующих данную функцию, есть наиболее простая. Поиск логической формулы, соответствующей этой схеме, представляет большой практический интерес. Преобразование формул булевых функций основано на использовании соотношений булевой алгебры.

## Основные законы алгебры логики

Основные законы алгебры логики позволяют проводить эквивалентные преобразования логических 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 + 1=1; x + x=1;
* всегда ложны высказывания: x ∙ 0=0; x ∙ x=0;
* правило двойного отрицания х=х.

*Переместительный* закон:

* для дизъюнкции x1+x2 = x2+x1;
* для конъюнкции x1∙x2 = x2∙x1;
* для суммы по модулю два x1x2 = x2x1;

*Сочетательный* закон

* для дизъюнкции x1+(x2+x3)=(x1+x2)+x3;
* для конъюнкции x1∙(x2∙x3)= (x1∙x2)∙x3;
* для суммы по модулю два x1(x2x3) = (x1x2)x3;

то есть группирование переменных внутри дизъюнкции (конъюнкции) не изменяет значений функции.

*Распределительный* закон:

* для дизъюнкции x1+x2∙∙x3=(x1+x2)(x1+x3);

то есть дизъюнкция переменной и конъюнкции эквивалентна конъюнкции дизъюнкций этой переменной с сомножителями;

* для конъюнкции x1∙(x2+x3)= x1∙x2+x1∙x3;

то есть конъюнкция переменной и дизъюнкции равносильна дизъюнкции конъюнкций этой переменной со слагаемыми.

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

* для дизъюнкции x1+x2=x2 ∙ x1;
* для конъюнкции x1∙x2=x2+x1;

то есть отрицание дизъюнкции (конъюнкции) переменных равно конъюнкции (дизъюнкции) отрицаний этих переменных.

Правило де Моргана справедливо для любого числа переменных

x1+x2+…+xn= x1 ∙ x2 ∙ … ∙ xn,

x1∙x2∙…∙xn= x1 + x2 + … ∙ xn.

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* x1∙x2+x1∙x2=x1 ;

Следующие соотношения могут быть выведены из рассмотренных выше:

x1+x1∙x2 = x1 ;

x1+x1∙x2 = x1∙1 +x1∙x2 = x1 ∙(1 + x2) = x1∙1 = x1;

x1 ∙(x1x2) = x1;

## Формы представления функций алгебры логики

Основными понятиями, лежащим 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 x1x2x3 , x1x4 , x1x2x4 являются элементарными конъюнкциями, а выражения вида x1x2x3 , x1x4 , x1x2x4 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 x1+x2+x3 , x1+x4 , x1+x2+x4 , а выражения вида x1+x2+x3, x1+x4 , x1+x2+x4 не являются элементарными дизъюнкциями.

*Дизъюнктивной нормальной формой* (ДНФ) булевой функции называется дизъюнкция конечного числа элементарных конъюнкций.



Число переменных, входящих в элементарную конъюнкцию, определяет *ранг* этой конъюнкции.

*Совершенной ДНФ* (СДНФ) логической функции от n аргументов 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 дизъюнктивно; переменная имеет знак инверсии, если на соответствующем наборе имеет нулевое значение.*



В общем виде это можно записать следующим образом:



Элементарные конъюнкции, образующие СДНФ, называют также *конституентами* (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.



*Совершенной КНФ* (СКНФ) логической функции от n аргументов 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 конъюнктивно; переменная имеет знак инверсии, если на соответствующем наборе имеет единичное значение.*



Элементарные дизъюнкции, образующие СКНФ, называют *конституентами* (составляющими) *нуля* (макстерм), так как они соответствуют наборам, при которых функция принимает нулевое значение. Построение СКНФ по таблице истинности называют составлением булевой функции по условиям ложности.

Введём обозначение:



**Теорема (Разложение в дизъюнкцию).** Любую функцию f(x1,...,xm) для любого n (1  n  m) можно представить в виде

f(x1,...,xm) = bigvsx11 & ... & xnn & f(1,...,n,xn+1,...,xm)

Доказательство. Покажем, что для любого набора значений переменных (x1,...,xn,xn+1,...,xm) значения левой и правой частей совпадают. Возьмём фиксированный набор (x1,...,xn,xn+1,...,xm). Рассмотрим выражение x11 & ... & xnn. Если одно из значений xii равно 0, то и всё выражение равно 0. Тогда и выражение x11 & ... & xnn & f(1,...,n,xn+1,...,xm) равно 0. Единице же выражение x11 & ... & xnn равно только в том случае, если 1 = x1, ..., n = xn. При этом f(1,...,n,xn+1,...,xm) = f(x1,...,xn,xn+1,...,xm) Таким образом, значение правой части всегда равно f(x1,...,xm), то есть значению левой части.

**Теорема (Разложение в конъюнкцию).** Любую функцию f(x1,...,xm) для любого n (1  n  m) можно представить в виде

Разложения по всем переменным дают суперпозицию конъюнкции, дизъюнкции и отрицания.

**Следствие 1 (Совершенная дизъюнктивная нормальная форма).**

Любая функция f может быть представлена в следующей форме:

f(x1,...,xm) = bigvsx11 & ... & xmm & f(1,...,m) =

= bigvs1 x11 & ... & xmm

**Следствие 2 (Совершенная конъюнктивная нормальная форма).**

Любая функция f может быть представлена в следующей форме:



Таким образом, любая булева функция может быть представлена суперпозицией конъюнкции, дизъюнкции и отрицания. Разложение по всем переменным в дизъюнкцию называется совершенной дизъюнктивной нормальной формой функции, а в конъюнкцию – совершенной конъюнктивной нормальной формой. Совершенная дизъюнктивная и конъюнктивная нормальная формы дают способ представления булевой функции через суперпозицию конъюнкции, дизъюнкции и отрицания.

Чтобы получить совершенную дизъюнктивную нормальную форму, надо взять все наборы, на которых значение функции равно 1 и записать для каждого из них конъюнкцию переменных и их отрицаний. Если в наборе значение переменной равно 0 – то переменную надо взять с отрицанием, если 1 – без отрицания. Из получившихся конъюнкций надо построить дизъюнкцию.

Чтобы получить совершенную конъюнктивную нормальную форму, надо взять все наборы, на которых значение функции равно 0 и записать для каждого из них дизъюнкцию переменных и их отрицаний. Если в наборе значение переменной равно 0 – то переменную надо взять без отрицания, если 1 – с отрицанием. Из получившихся дизъюнкций надо построить конъюнкцию.

## Системы функций алгебры логики

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 сколь угодно сложную булеву функцию. Проектирование цифровых автоматов основано на знании таких систем булевых функций. Последнее особенно важно для определения набора элементарных логических схем, из которых можно построить произвольный цифровой автомат. Проблема функциональной полноты является центральной проблемой функциональных построений в алгебре логики.

*Функционально полной системой* булевых функций (ФПСБФ) называется совокупность таких булевых функций (f1, f2, ... fk), 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 Р 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 Р. 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 функция алгебры логики может быть представлена многочленом вида:



где кi – коэффициент, принимающие значения 0 или 1.

Теорема позволяет представить любую ФАЛ в виде полиномов различной степени.

Задача построения полинома Жегалкина сводится к нахождению коэффициентов ki, Для любых конституент единицы k1 и k2 имеет место следующее соотношение k1 V k2=k1 ⊕ k2. Оно позволяет выполнить переход от СДНФ к полиному Жегалкина. Для этого достаточно заменить в СДНФ символ ∨ (дизъюнкции) на символ ⊕, выполнить подстановку вида х=х ⊕ 1 с последующими преобразованиями в алгебре Жегалкина.

Перечислим предполные классы булевых функций:

*Класс линейных функций*. Функция алгебры логики называется линейной, если ее можно представить полиномом первой степени:



Примем без доказательства утверждение, что при суперпозиции линейных функций получаем линейную функцию.

Существует 8 линейных функций от 2 переменных.

Таблица 10.

|  |  |  |  |
| --- | --- | --- | --- |
|  | k2 k1 k0 | f(x1x2…xn)=k0⊕k1x1⊕k2x2 |  |
|  | 0 0 0 | 0 |  |
|  | 0 0 1 | 1 |  |
|  | 0 1 0 | x1 |  |
|  | 0 1 1 | x1 ⊕ 1 |  |
|  | 1 0 0 | x2 |  |
|  | 1 0 1 | x1 ⊕ 1 |  |
|  | 1 1 0 | x1 ⊕ x2 |  |
|  | 1 1 1 | 1 ⊕ x1 ⊕ x2 |  |

Следовательно, в функционально полном базисе должна содержаться хотя бы одна нелинейная логическая функция.

*Класс функций, сохраняющих ноль*. К булевым функциям сохраняющим константу 0, относят такие булевы функции f(x1,...,xn), для которых справедливо соотношение f(0,...,0)=0.

*Класс функций, сохраняющих единицу* К булевым функциям сохраняющим константу 1, относят такие булевы функции f(x1,...,xn), для которых справедливо соотношение f(1,...,1)=1.

*Класс монотонных функций*. Функция алгебры логики называется монотонной если при любом возрастании набора аргументов значения этой функции не убывают.Двоичный набор A=<a1,a2,...,an> не меньше двоичного набора B=<b1,b2,...,bn> , если для каждой пары (ai,bi) i = 1...n справедливо соотношение ai >= bi.

Если у двух наборов есть и большие и меньшие аргументы f(0,1) и f(1,0), то наборы считаются несравнимыми. Таким образом f(0,0)⇒f(0,1) ⇒f(1,1) или f(0,0)⇒f(1,0) ⇒f(1,1).

*Класс самодвойственных функций*. Булевы функции f1(x1,...,xn) и f2(x1,...,xn) называются двойственными друг другу, если выполняется соотношение



К самодвойственным булевым функциям относят такие булеы функции, которые двойственны по отношению к самим себе, то есть булева функция называется самодвойственной, если на любых двух противоположных наборах х1,х2,…,хn и  она принимает противоположные значения .

Любая ФАЛ, полученная с помощью операции суперпозиции и подстановки из функций одного класса, принадлежит этому же классу.

*Базисом* называется полная система ФАЛ, с помощью которой любая ФАЛ может быть представлена суперпозиций исходных функций.

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

не являющуюся линейной,

не сохраняющую ноль,

не сохраняющую единицу,

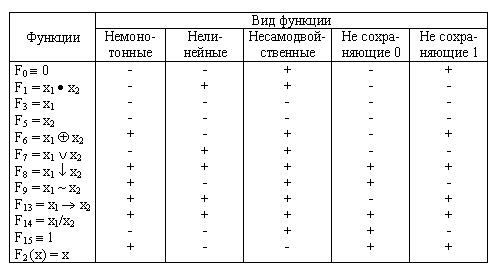
не являющуюся монотонной,

не являющуюся самодвойственной.

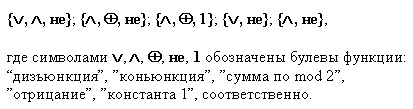
Иначе говоря, система булевых функций является функционально полной тогда и только тогда, когда она целиком не содержится ни в одном из предполных классов.

Рассмотрим примеры ФПСБФ. Для удобства изложения материала сведем элементарные булевы функции двух переменных и некоторые функции одной переменной в таблицу, классифицируя каждую из них по признакам принадлежности к предполным классам.

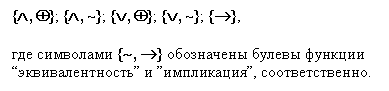
Таблица 11.



Из таблицы видно, что каждая из функций F8 и F14 являются ФПСБФ. Иными словами, используя, например, только булеву функцию F14 - "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:



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 хотя бы одну немонотонную булеву функцию. Примерами ослабленных ФПСБФ могут служить следующие системы:



Логические элементы, реализующие логические 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, что удаление одной (любой) функции превращает систему ФАЛ в неполную. Иначе говоря, функционально полный базис – это набор операций алгебры логики (и соответствующих им элементов) позволяющих построить любую функцию алгебры логики.

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.





## Минимизация ФАЛ

При проектировании цифровых автоматов широко используются методы минимизации булевых функций, позволяющие получать экономичные схемы. Общая задача минимизации булевых функций может быть сформулирована следующим образом: найти аналитическое выражение заданной булевой функции в форме, содержащей минимально возможное число букв. Следует отметить, что в общей постановке данная задача пока не решена, однако достаточно 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 булеву функцию).

Булева функция g(x1,...,xn) называется *импликантой* булевой функции f(x1,...,xn), если для любого набора переменных, на котором g=1, справедливо f=1.

Таблица 12.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x3x2x1 | f | g1 | g2 | g3 | g4 | g5 | g6 | g7 |
| 000 001 010 011 100 101 110 111 | 0 0 0 1 0 0 1 1 | 0 0 0 0 0 0 0 1 | 0 0 0 0 0 0 1 0 | 0 0 0 0 0 0 1 1 | 0 0 0 1 0 0 0 0 | 0 0 0 1 0 0 0 1 | 0 0 0 1 0 0 1 0 | 0 0 0 1 0 0 1 1 |



Импликанта g булевой функции f, являющаяся элементарной конъюнкцией, называется *простой*, если никакая часть импликанты g не является импликантой функции f.

Из примера видно, что импликанты g3 = x1x2 и g5 = x2x3 являются простыми импликантами функции f. Импликанты g1, g2, g4, g6 не являются простыми, так как их части являются импликантами функции f, например g1 является частью g3. 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: g3 и g5. Следовательно, сокращенная ДНФ функции f имеет вид

f = g3 + g5 = x1x2 + x2x3.

Как видно из табл. 12, импликанты g3, g5 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 убрать одну или несколько простых импликант, удаление которых не приводит к изменению значений функции на всевозможных значениях ее переменных. Такие простые импликанты назовем *лишними*.

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, то есть булева функция может иметь несколько тупиковых ДНФ.

Тупиковые ДНФ булевой функции f, содержащие минимальное число букв, являются *минимальными*. МДНФ тоже может быть несколько.

Минимизировать функции, то есть 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.

### Метод Квайна

**Теорема Квайна**. Для получения минимальной формы логической функции необходимо в СДНФ произвести все возможные склеивания и поглощения так, что в результате будет получена сокращенная ДНФ. Сокращенная ДНФ в общем случае может содержать лишние простые импликанты, которые необходимо выявить 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. *соотношение* *неполного склеивания*:

, где  и  - две конъюнкции, а *F* - любое элементарное произведение;

2. *соотношение* *поглощения*

.

Справедливость обоих соотношений 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 р = xi1xi2 ... xin может быть получена. В самом деле, применяя к р операцию развертывания (обратную операции склеивания):



по всем недостающим переменным xik, ..., xim исходной функции f, получаем совокупность S конституент единицы. При склеивании всех конституент из S получим импликанту р. Последнее очевидно, поскольку операция склеивания обратна операции развертывания. Множество S конституент обязательно присутствует в совершенной ДНФ функции f поскольку р - ее импликанта.

В результате выполнения склеивания получается конъюнкция n-1 ранга, а конъюнкции  и  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 конститутиентами единицы исходной СДНФ. В строке напротив каждой простой импликанты ставится метка под теми наборами (конститутиентами единицы), на которых она принимает значение 1. Соответствующие конститутиенты 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, то есть таких, которые (и только они) покрывают некоторый исходный набор. Рассмотрим на примере минимизации логической функции:

fСДНФ= V (1,2,5,6,7)=x1x2x3+ x1x2x3+ x1x2x3+ x1x2x3+ x1x2x3

1 2 3 4 5

Выполним операцию склеивания:

1 – 3 (x1) x2x3 1

2 – 4 (x1) x2x3 2

3 – 5 (x2) x1x3 3

4 – 5 (x3) x1x2 4

В результате выполнения первого шага склеивания получаем четыре новые конъюнкции, простых импликант не выявлено. Полученные конъюнкции более не склеиваются и образуют сокращенную ДНФ.

fсокр СДНФ=x2x3+ x2x3+ x1x3+ x1x2

Для выявления обязательных простых импликант и фрормирования на их основе минимального покрытия строится импликантная таблица (табл. 13). В строках импликантгой таблицы записываются простые импликанты, а в столбцах конституэнты единицы. Звездочка ставится если простая импликанта покрывает контитуэнту.

Таблица 13.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | x1x2x3 | X1x2x3 | x1x2x3 | x1x2x3 | x1x2x3 |  |
|  | x2x3 | \* |  | \* |  |  |  |
|  | x2x3 |  | \* |  | \* |  |  |
|  | x1x3 |  |  | \* |  | \* |  |
|  | x1x2 |  |  |  | \* | \* |  |

Простые импликанты являются обязательными так как только они покрывают конституэнтыи включаются в минимальное покрытие. Остается одна непокрытая конституэнта x1x2x3 которая может быть покрыта одной из двух оставшихся простых импликант. Это приводит к получению двух тупиковых форм.





### Метод Блейка - Порецкого

Метод позволяет получать сокращенную ДНФ булевой функции f из ее произвольной ДНФ. Базируется на применении формулы обобщенного склеивания:

,

справедливость которой легко доказать. Действительно,

Следовательно,



В основу метода положено следующее утверждение: если в произвольной ДНФ булевой функции f произвести все возможные oбобщенные склеивания, а затем выполнить все поглощения, то в результате получится сокращенная ДНФ функции f.

Рассмотрим пример. Пусть булева функция f задана произвольной ДНФ.



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 х1. В результате склеивания получим:



Первый и третий элемент исходной ДНФ допускают обобщенное склеивание как по переменной х1, так и по х2. После склеивания по x1 имеем:



После склеивания по x2 имеем:



Второй и третий элемент ДНФ допускают обобщенное склеивание по переменной х2. После склеивания получаем:



Выполнив последнее обобщенное склеивание, приходим к ДНФ:

После выполнения поглощений получаем:



Попытки дальнейшего применения операции обобщенного склеивания и поглощения не дают результата. Следовательно, получена сокращенная ДНФ функции f. Далее задача поиска минимальной ДНФ решается с помощью импликантной матрицы точно так же, как в методе Квайна.

### Метод минимизирующих карт Карно (Вейча)

При минимизации логической функции от небольшого числа переменных удобным является графический метод представления функции с помощью диаграмм (карт) Вейча и их разновидности - Карно. Карта Вейча представляет собой развертку n-мерного куба на плоскости. При этом вершины куба представляются клетками карты, каждой из которых поставлена в соответствие конститутиента единицы или нуля. Переменные, обозначающие клетки диаграммы, расставляются таким образом, чтобы наборы, записанные в двух смежных клетках, имели кодовое расстояние, равное единице. Поскольку такие наборы располагаются в смежных клетках, они получили название *соседних наборов*. В клетку карты, соответствующую конститутиенте 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:

* количество клеток карты в одном контуре должно быть равно 2n;
* для контура, содержащего 2n, клеток должно быть **n** осей симметрии;
* количество контуров должно быть минимально;
* число единиц в контуре должно быть максимально;
* контуры могут пересекаться, то есть некоторая клетка может входить в несколько контуров.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | х2 | | | |
| x1 | 1 | 1 | 1 |  |
| 1 |  | 1 |  |
|  | х3 | | | |

Рис. 15. Карта Вейча для fСДНФ

На рис. 15 показана заполненная карта Вейча, соответствующая функции fСДНФ. 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 несколько тупиковых форм ФАЛ. Таким образом, по данной карте может быть получена одна из тупиковых форм:





Если функция задана в форме ДНФ, то не обязательно 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | x2 | | | |  |
| x1 | 1 |  | 1 |  | x4 |
| 1 |  |  |  |
| 1 |  | 1 | 1 |
|  |  | 1 | 1 |
|  | x3 | | | |  |

Рассмотрим сказанное на примере функции

fДНФ=x1x2+ x1x2x3+ x1x2x3x4+ x1x2x3x4

Первому члену ДНФ поставлены в соответствие четыре клетки карты, второму – две клетки, третьему и четвертому по одной клетке соответственно.

Далее объединение единиц в контуры и выбор их минимального числа осуществляется рассмотренным выше методом.

Рис. 16. Карта Вейча

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | x2 | | | |  |
| x1 | 1 | 0 | 1 | 1 | x4 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 |
|  | x3 | | | |  |

Рис. 17. Карта для инверсии функции

Если выбраны самые большие контуры и использовано по возможности, меньшее их число, то будет получена самая простая дизъюнктивная нормальная форма. 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, не содержащие единиц, и поступая с ними так же, как мы поступали с клетками, содержащими единицы, можно получать конъюнктивные нормальные формы.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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. 18. Структура карты Карно

Если логическая функция задана таблицей истинности, то более удобной для графического представления функции является карта Карно. В отличие от карты Вейча в карте Карно строки и столбцы закодированы r-разрядным кодом Грея. Код Грея – двоичный код, в котором рядом стоящие коды – соседние (их кодовое расстояние равно единице). В карте Карно каждой клетке соответствует код состоящий из кода строки и кода столбца (рис. 18).

На рис. 19 показано соответствие клеток карты Карно и строк таблицы истинности. При этом в карте рис. 19 б показаны координаты единичных и нулевых значений функции, а на карте рис. 19 в показано соответствие строк таблицы истинности и ячеек карты.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | x1 | x2 | x3 | 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  а) | 1 | 1 | 1 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 0 | 0 | 1 | 3 | 2 |
| 1 | 4 | 5 | 7 | 6 |

б) в)

Рис. 19 Таблица истинности и карта Карно

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 0 | 000  1 | 001  1 | 011  0 | 010  0 |
| 1 | 100  0 | 101  1 | 111  0 | 110  1 |

## 

### Минимизация коньюнктивных нормальных форм.

Минимизация КНФ производится аналогично рассмотренным методам минимизации ДНФ булевых функций, поэтому остановимся лишь на основных положениях.

Напомним, что конституентой нуля называется функция, принимающая значение 0 на одном наборе. Она выражается дизъюнкцией всех переменных функций. Например, набору 0110 соответствует конституента нуля x1+x2+x3+x4.

*Имцлицентой* g булевой функции f называется функция, принимающая значение 0 на подмножестве нулевых наборов функции f.

*Простой имплицентой* функции f называется элементарная дизъюнкция, являющаяся имплицентой функции f, причем никакая ее собственная часть 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 при поиске минимальной ДНФ, так как возможны только два варианта: либо данная простая имплицента поглощает данную конституенту нуля, либо нет в соответствии с соотношением поглощения:

(A v x)A = A

Что касается первого этапа - поиска всех простых имплицент, то практически все методы минимизации ДНФ имеют свои аналоги для КНФ. Расссмотрим это подробнее.

Соотношение склеивания по Квайну:



Соотношение склеивания по Блейку:



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 и использование второго дистрибутивного закона. Например, функция, заданная минимальной ДНФ: *x1/x2 v /x1x2* дает возможность определить ее сокращенную КНФ:



По диаграмме Вейча поиск минимальной КНФ осуществляется так же просто, как в случае ДНФ. Отличие состоит лишь в том, что анализируются нулевые наборы и переменные выписываются с инверсиями. Например, для функции, заданной диаграммой (рис. 20).

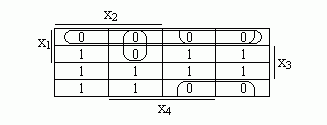


Рис. 20. Карта Вейча для поиска f min КНФ

минимальной КНФ, является



Для сравнения найдем минимальную ДНФ:



В данном случае ДНФ оказалась проще, В общем случае о сравнительной сложности минимальных ДНФ и КНФ нельзя говорить заранее, но можно отметить следующее: количество букв минимальной ДНФ произвольной функции f и минимальной КНФ функции /f одинаково.

### Минимизация не полностью определенных ФАЛ

Если при синтезе логической схемы, реализующей некоторую ФАЛ n переменных, окажется, что некоторые наборы из общего числа 2n никогда не смогут появиться на входах схемы, то данная логическая функция не определена на этих наборах. Тогда 2n наборов переменных можно подразделить на три группы: наборы, на которых функция принимает единичное значение L, нулевое значение D и группа наборов, на которых функция не определена N (неопределенные наборы). ФАЛ, содержащая неопределенные наборы, называется неполностью 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 более простой минимизированной логической функции.



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | x2 | | | |  | |  |
| x1 | 1 |  | 1 | \* | |
| 1 | \* |  | \* | |
|  | x 3 | | | |  | |  |

Звездочками на карте (рис. 21) отмечены наборы на которых функция f не определена. Если не учитывать неопределенные наборы, то записать минимальная форма будет иметь вид: . В случае если неопределенные наборы участвуют в образовании контуров, а следовательно, и fМДНФ, то функция примет следующий вид: . Таким образом, схемная реализация полученной fМДНФ будет дешевле.

Рис.21 Карта Вейча

Приведем несколько примеров минимизации частичных булевых функций:



### Кубическое задание функций алгебры логики.

Как следует из рассмотренного выше, функция алгебры логики (булева функция) может быть задана:

* аналитически (системой булевых функций);
* словесным описанием;
* таблицей истинности;
* картами (диаграммами) Венна, Вейча, Карно;
* логической схемой;

Более компактной формой записи функций алгебры логики является форма, когда вместо полного перечисления всех конъюнкций (дизъюнкций) используют номера наборов, на которых функция принимает единичное значение. Так, например, форма записи f(x1x2x3)=V F(0,2,3) означает, что функция от трех переменных принимает единичное значение на 0, 2 и 3 наборах из таблицы истинности. Такая форма записи называется *числовой*.

Некоторые методы минимизации ориентируются на задание булевой функции в виде кубического покрытия. При этом логическая функция удобно интерпретируется с использованием ее геометрического представления. Так, функцию двух переменных можно интерпретировать как плоскость, заданную в системе координат х1х2. Получится квадрат, вершины которого соответствуют комбинациям переменных. Для геометрической интерпретации функции трех переменных используется куб, заданный в системе координат х1х2х3 .

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 определим координатную систему с координатами (e1,......,en), ei=0,1.Установим соответствие между членом СДНФ x1e1 x2e2... xnen и некоторой вершиной e1 ,e2 , ...., en куба. При этом xiei = xi если ei=1, и xiei = xi если ei=0.



Рис.23. Геометрическое представление функции двух и трех переменных

Для отображения булевой функции n переменных на n – кубе устанавливается соответствие между членами СДНФ и вершинами n - куба. На n- кубе определим координатную систему с координатами (e1,......,en), ei=0,1.Установим соответствие между членом СДНФ x1 x2 ... xn и некоторой вершиной e1 ,e2 , ...., en куба.

Каждый набор при кубическом задании ФАЛ называется кубом.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 14. | | | | |
|  | х1 | х2 | х3 | f |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | - |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 |

Как следует из таблицы истинности, функция f образована на трех группах наборов переменных: L={3,4,5,6,7}, D={0,2} и N={1}.

Конъюнкции максимального ранга (конститутиенты единицы) принято называть 0-кубами. Множество 0- кубов образуют кубический комплекс

011

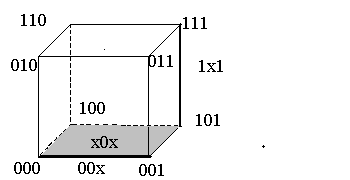
100

К0 = 101

110

111

Над 0-кубами, кодовое расстояние которых равно 1, выполняется операция склеивания, в результате которой образуются новые кубы, содержащие свободные координаты.*Свободная (независимая)* координата может принимать как нулевое, так и единичное значение, остальные компоненты называются *связанными*.. Куб, содержащий свободные координаты, покрывает кубы, на которых он образован. Куб с одной независимой координатой х называется *кубом первой размерности* и в геометрическом представлении это ребро, покрывающее обе вершины. Кубы, образующиеся в результате последовательного выполнения операции склеивания, назовем r-кубами, где r – размерность полученного куба.



Геометрическая интерпретация сказанного показана на рис. 24. В результате склеивания кубов 101 и 111 (0-кубы, вершины) образован 1-куб 1x1 (ребро), а 1-кубов 00x и 10х - 2-куб х0х (грань).

Рис. 24. Образование новых кубов.

Кубическое представление ФАЛ позволяет обойтись тремя переменными 0,1,х вместо х1, х2,...,хn .

Количество свободных координат в кубе определяет его размерность r, чем больше r тем больше свободных координат и тем меньше входов будет иметь схема.

Цена схемы оценивается количеством входов: ,

где k-количество полученных кубов, n-ri - количество единичных и нулевых значений i-го куба.

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.

Ниже приведено изображение куба, соответствующего булевой функции от четырёх переменных (гиперкуб).

Как следует из рисунка, геометрическое представление 4-куба уже довольно сложное. Поэтому для функций, зависящих более чем от четырёх переменных, предпочтительным является аналитическое представление булевых функций.

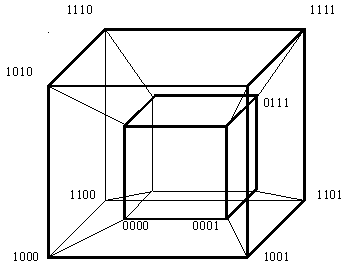


Рис. 25. Геометрическое представление функции четырех переменных

### Метод Квайна-Мак Класки

Это метод, ориентированный на числовое задание ФАЛ 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 первичных импликант. В случае большого их количества сложность этого сравнения существенно возрастает. В рассматриваемом методе все исходные n-кубы разбиваются на непересекающиеся подгруппы по количеству единиц в кубе. Пусть, например, задана функция:

f СДНФ(x1x2x3x4) = V (2, 3, 4, 6, 9, 10, 11, 12)

Сформируем кубический комплекс K, состоящий из 0-кубов.

K=(0010, 0011, 0100, 0110, 1001, 1010, 1011, 1100)

Выполнив разбиение комплекса K на группы, получим:

Попарное сравнение можно проводить только между соседними по номеру группами кубов, так как кубы, порождающие новые кубы, должны иметь кодовое расстояние, равное 1. В результате сравнения кубов получим:

После выполнения первого шага алгоритма простых импликант не выявлено. Полученные 1-кубы разобьем на n групп кубов в зависимости от местоположения свободной координаты в кубе.

Далее выполняется сравнение кубов внутри каждой из групп. В результате чего могут быть получены 2-кубы, которые, в свою очередь, аналогично 1-кубам будут объединены в группы по совпадению свободных координат и вновь выполнено сравнение. На каждом шаге сравнения выявляются кубы, не участвовавшие в образовании новых кубов, и исключаются из дальнейшего упрощения. Для рассматриваемого примера сравнение в группах  …  привело к образованию двух новых кубов х01х и х01х и кубов, не образовавших новых {х100, 0х10, 10х1, 01х0}.

Дальнейшее сравнение не приводит к формированию новых кубов . Таким образом, получено множество простых импликант:

fсокр.ДНФ={х100, 0х10, 10х1, 01х0, х01х}

Далее аналогично методу Квайна строится импликантная таблица (табл.15). Формирование минимального покрытия сводится к выявлению обязательных простых импликант и на их основе построению тупиковых форм.

Таблица 15.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Простые импликанты | Конститутиенты единицы | | | | | | | |  |
|  | 0010 | 0100 | 0011 | 1100 | 0110 | 1001 | 1010 | 1011 |  |
|  | х100 |  | \* |  | \* |  |  |  |  |  |
|  | 0х10 | \* |  |  |  | \* |  |  |  |  |
|  | 10х1 |  |  |  |  |  | \* |  | \* |  |
|  | 01х0 |  | \* |  |  | \* |  |  |  |  |
|  | х01х | \* |  | \* |  |  |  | \* | \* |  |

Из таблицы следует, что простые импликанты x100, 10x1, x01x являются обязательными. Оставшиеся две простые импликанты не являются обязательными и образуют следующие две тупиковые формы.

f МДНФ = {х100, 10х1, х01х, 0х10} – I тупиковая форма.

f МДНФ = {х100, 10х1, х01х, 01x0} – II тупиковая форма.

### Алгоритм извлечения (Рота)

Метод Рота ориентируется на задание логической функции 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 извлечения: \*,∩ ,#.***

Реализация алгоритма извлечения осуществляется на основе специальных логических операций, которые позволяют полностью формализовать процесс получения минимальной формы.

*Операция умножения кубов (\*).*

Операцияпроизведения кубов а=а1а2...аn и b=b1b2...bn обозначается как с=а\*b и служит для образования r-куба, противоположные (r-1) – грани которого содержатся в кубах а и b. Предварительные координаты куба c определяются в соответствии с таблицей.

|  |  |  |  |
| --- | --- | --- | --- |
| \* | 0 | 1 | х |
| 0 | 0 | y | 0 |
| 1  ai | у | 1 | 1 |
| x | 0 | 1 | х |

⎩ \_\_\_\_\_\_\_\_\_\_\_\_\_\_∨\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_⎭

bi

Окончательно координаты куба формируются:

m(a1\*b1)m(a2\*b2) ... m(an\*bn,) если ai\*bi=y только для одного i

a\*b=

∅, если ai\*bi=y, для i ≥ 2

где m(ai\*bi) – окончательная i-я координата куба с, m(0)=0, m(1)=1, m(x)=x.

Эта операция соответствует операции склеивания: образуется новый r-куб, если кодовое расстояние двух исходных кубов равно 1. Рассмотрим некоторые примеры.



1) 011 2) 11х

\*001 \*01х

0y1 ⇒ 0x1– ребро покрывает две вершины у1х ⇒ х1х – грань

3) 0х1 4) х1х

\*1х0 \*011

уху ⇒ ∅ – две координаты имеют значение у. 011

*Операция пересечения кубов* (**∩**)*.*

Операция пересечения кубов а=а1а2...аn и b=b1b2...bn обозначается как с=а**∩**b и служит для выделения куба с=с1с2...сn  , являющегося общей частью кубов а и b. Координаты с1с2...сn определяются согласно таблице:

|  |  |  |  |
| --- | --- | --- | --- |
| ∩ | 0 | 1 | x |
| 0 | 0 | ∅ | 0 |
| 1  ai | ∅ | 1 | 1 |
| x | 0 | 1 | x |

⎩\_\_\_\_\_\_\_\_\_\_\_\_\_\_∨\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_⎭

bi

m(a1∩b1)m(a2∩b2) ... m(an∩bn,)

a∩b=

∅, если существует такое i, для которого ai ∩ bi =∅

где m(ai\*bi) – окончательная i-я координата куба с, m(0)=0, m(1)=1, m(x)=x.

Рассмотрим примеры.

1) ∩100 2) ∩1х0

001 10х

10∅ ⇒ ∅ общей части у 100 ⇒ 100 общая часть

кубов нет; (вершина) кубов (рёбер);

3) ∩х1х 4)  ∩0хх

0хх 0х0

01х ⇒ 01х общая часть – ребро; 0х0 ( совпадает с

операцией \*).



*Операция вычитания кубов (#).*

Операция вычитания из куба а=а1а2...аn куба b=b1b2...bn обозначается как с=а#b и служит для удаления из куба а общей части кубов а и b.

Координаты куба с формируются согласно таблице:

|  |  |  |  |
| --- | --- | --- | --- |
| # | 0 | 1 | x |
| 0 | z | y | z |
| 1  ai | y | z | z |
| x | 1 | 0 | z |

⎩ \_\_\_\_\_\_\_\_\_\_\_\_\_\_∨\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_⎭

bi

∅, если ∀ i, аi#bi=z; z-означает, что координаты совпадают;

c=а#b= a, если ∃ i, аi#bi=y, у-означает, что координаты противоположны;

U а1а2...аi-1 αi аi+1 ... аn, если αi равно 0 или 1 для одного или

i=1 нескольких i.

По этим αi-координатам производится объединение (U) кубов, получаемых в результате замены в кубе а символа х на соответствующее значение (0,1) координаты αi. Рассмотрим примеры выполнения операции #.

1) #х1х 2) #х1x

x11 110

zz0 ⇒ x10 0z1 ⇒ 01x

3) #хx1 4) #х11 x11

x10 xx1

z0y ⇒ xx1 zzz ⇒ ∅



5) # 0ххx

хх01

zz10 ⇒ 0x1x

0xx0

Далее рассмотрим алгоритм извлечения (Рота) на примере минимизации булевой функции, заданной покрытием С0.



Рис. 29. Геометрическое задание исходного покрытия.

Исходное покрытие С0 задано объединением множеств кубов L и N. Кубы комплекса N- это наборы на которых функция не определена и включаются в покрытие ради возможного дополнительного упрощения комплекса L в процессе минимизации. Минимальное покрытие комплексов L и N, С min – называется К- покрытием L.

Общий алгоритм построения минимального К-покрытия называется алгоритмом извлечения и состоит в следующем:

▪ нахождении множества Z простых импликант комплекса К;

▪ 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 С0 комплекса К в множество простых импликант Z осуществляется с помощью операции умножения кубов.В результате первого шага (С0\*С0) предусматривается выявление как новых кубов Сy (первой и более высокой размерности), так и кубов, которые не образуют новых кубов (включаются в множество Z0). Из полученных новых кубов образуется множество А1. В1=С0-Z0. Для следующего шага формирования множества Z формируется множество С1=А1U В1. Для уменьшения мощности множества кубов С1 выполним операцию поглощения (удаления) кубов образующих множество С1 кубами из множества А1 (А1⊆С1). Для рассматриваемого примера получим:

Таблица 16.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | С0\*С0 | х010 | 0х10 | 0000 | 0х01 | 1110 | 1x10 |
|  | х010 | - |  |  |  |  |  |
|  | 0х10 | 0010 | - |  |  |  |  |
|  | 0000 | 00у0 | 00у0 | - |  |  |  |
|  | 0х01 | ø | ø | 000у | - |  |  |
|  | 1110 | 1у10 | у110 | ø | ø | - |  |
|  | 1х01 | ø | ø | ø | ух01 | ø | - |
|  | А1 | 00х0 | х110 | 000х | хх01 |  |  |
|  | 1х10 |  |  |  |  |  |

00х0

1х10

00х0 000х А1 00х0

1х10 х110 1х10

А1=х110 хх01 после выполнения 000х

000х С1= х010 ⇒ операции **⇒** С1= х110

хх01 0х10 поглошения хх01

0000 В1 х010

Z0=Ø 0х01 0х10

1110

1х01

Среди кубов С0 возможно находятся такие кубы, которые с кубами множества А1 могут дать новые кубы, или оказаться простыми импликантами, после второго шага (С1\*С1). При формировании таблицы для выполнения операции С1\*С1 следует учесть, что В1\*В1 уже выполнялось на шаге С0\*С0. Следовательно:

С1\*С1=(А1UВ1)\*(А1UВ1)=(А1\*А1)U(А1\*В1)U(В1\*А1)U(В1\*В1)=(А1\*А1)U(А1\*В1)

Таблица 17.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | С1\*С1 | 00х0 | 1х10 | 000х | х110 | хх01 |
|  | 00х0 | - |  |  |  |  |
|  | 1х10 | у010 | - |  |  |  |
|  | 000х | 0000 | ø | - |  |  |
|  | х110 | 0у10 | 1110 | ø | - |  |
|  | хх01 | 000у | ø | 0001 | ø | - |
|  | х010 | 0010 | 1010 | 00у0 | ху10 | Ø |
|  | 0х10 | 0010 | ух10 | 00у0 | 0110 | Ø |
|  | А2 | ø | хх10 | ø | хх10 | Ø |

В результате выполнения умножения С1\*С1 получено:

А2={хх1х}

00х0

Z1=

000х

Необходимо отметить, что куб хх01 не дал нового куба. Но это куб второй размерности и новые кубы может дать на третьем шаге (С3\*С3). Поэтому его не следует включать в число кубов, образующих множество Z1.

1х10 хх10

х110 1х10

В2=хх01 х110 хх10

=

C2=А2UB2=

х010 х010 хх01

0х10 0х10

хх01

Таблица 18.

|  |  |  |
| --- | --- | --- |
|  | С2\*С2 | хх10 |
|  | хх10 | - |
|  | хх01 | ø |
|  | А3 | ø |

Таким образом, получено А3= Ø, следовательно, новых кубов нет.

хх10

Z2=

хх01

В3=С2-Z2= Ø; C3=A3UB3= Ø.

На этом процесс выявления простых импликант окончен.



00х0

000х

Z=Z0UZ1UZ2=

хх01

хх10

Необходимо выяснить, не содержатся ли в этом множестве “лишние” импликанты.

1. *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 (#) и пересечения (∩) кубов.

Таблица 19.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | z#(Z-z) | 00x0 | 000x | xx01 | xx10 |
|  | 00х0 | - | zzz1  0001 | 11zy  xx01 | 11zz  1x10  x110 |
|  | 000х | zz1z  0010 | - | 11zz  1x01  x101 | y1yz  1x10  1yyz  x110 |
|  | xх01 | zzyy  0010 | zzzz  ø | - |  |
|  | xх10 | zzzz  ø | ø | zzyy  1x01  zzyy  x101 | - |
|  | остаток | Ø | Ø | 1x01  x101 | 1x10  x110 |

Таким образом из таблицы получено множество L-экстремалей.



1. Если результат вычисления будет Ø хотя бы в одном, любом случае, то это значит, что среди простых импликант есть такие кубы, которые покрывают *уменьшаемый*, а следовательно этот уменьшаемый не может быть L-экстремалью.

2. Если же полученный результат не Ø, то в противоположность предыдущему утверждению *уменьшаемый* куб оказался кубом большей размерности по отношению к другим простым импликантам.

1. 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 20.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | z#(Z-z)∩L | 1x01 | x101 | 1x10 | x110 |
|  | x010 | ø | ø | 1010 | ø |
|  | 0x10 | ø | ø | 1010 | ø |
|  | 0000 | ø | ø | ø | 0110 |
|  | 0x01 | ø | 1101 | ø | ø |

Из таблицы видно, что куб 1x01 не пересекается с кубами комплекса L. Однако куб x101 имеет с кубом 0x01 (из комплекса L) общую вершину 0101. Оба куба (1x01, x101) входят в куб более высокой размерности xx01 (L-экстремаль). Таким образом, куб 1x01, 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 Е. В результате вычитания получим L1=L#Е.

Таблица 21.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | L#Е | x010 | 0x10 | 0000 | 0x01 |
|  | xx01 | zzyy  x010 | zzyy  0x10 | zzzy  0000 | zzzz  ø |
|  | xx10 | zzzz  ø | zzzz  ø | zzyz  0000 | ø |
|  |  | ø | ø | 0000 | ø |

Из таблицы видно, что L1={0000}. Однако непокрытые L-экстремалями кубы должны быть покрыты другими импликантами из множества.

Z=Z-E= 

Теперь из полученного множества Z надо выбрать минимальное число кубов, с минимальной ценой (максимальной размерностью), чтобы покрыть непокрытые L-экстремалями элементы комплекса L. Выбор так называемого немаксимального куба осуществляется с помощью операции частичного упорядочивания кубов (<).

Куб a будет не максимален по отношению к кубу b, если выполняются одновременно два условия:

1) Сa ≥ Cb, где Са – цена куба а:;

2) a ∩ L1 ⊆ b ∩ L1, куб b покрывает не меньше кубов чем куб а.

Z

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 22. | | | |  |
|  |  | ∩ | 0000 |  |
|  | а | 00х0 | 0000 |  |
|  | b | 000х | 0000 |  |

Сa = Cb

Следовательно, кубы а и b равноценны и для покрытия вершины 0000 можно выбрать любой из них в качестве экстремали второго порядка

Е′2={000x} или E′′2={00x0}.

Следовательно, могут быть получены две тупиковые формы.

 - первая тупиковая форма



- вторая тупиковая форма

### Минимизация ФАЛ методом преобразования логических выражений

Рассмотрим подход к упрощению ФАЛ заключающийся в применении к ней скобочных преобразований. Пусть имеется функция

f=x1x3x4x6+x2x3x4x6 + x5x6 + x7

Применим к ней скобочные преобразования, в результате чего получим функцию f=((x1+x2)x3x4 + x5)x6 + x7.

Из выражений видно, что цена схемы до минимизации была равна 14, после стала равна 11. Таким образом общая стоимость схемы сократилась , однако функции до преобразования соответствовала схема имеющая два уровня элементов, а после 5 уровней. Таким образом, полученная схема будет работать примерно в 2,5 раза медленнее исходной (до преобразования).

## Применение правил и законов алгебры логики к синтезу некоторых цифровых устройств

### Синтез одноразрядного полного комбинационного сумматора

Пусть имеется два числа

A=a1a2 . . . a i-1a ia i+1 . . . an

B=b1b2 . . . b i-1bib i+1 . . . bn

В зависимости от значений аргументов ai, bi, zi формируется значение булевых функций Ci, и Пi. Введем следующие обозначения.

ai  ⇒ x Ci ⇒ С

bi  ⇒ y Пi ⇒ П

zi ⇒ z

Таблица истинности, отражающая алгоритм работы сумматора, имеет следующий вид.

Таблица 23.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | x | y | z | С | 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 | ⇐ Логические нули |
|  | 1 | 0 | 1 | 0 | 1 |  |
|  | 1 | 1 | 0 | 0 | 1 |  |
|  | 1 | 1 | 1 | 1 | 1 |  |

Запись одной функции с участием другой носит название совместной минимизации. С учетом этого функция ***C*** будет иметь вид



.

Таким образом, логическая схема синтезированного одноразрядного полного комбинационного сумматора имеет вид (рис. 30):

1

z

x

y

Рис. 30. Логическая схема полного сумматора.

С

П

&

1

&

1

&

1

&

&

### Синтез одноразрядного комбинационного полусумматора

Одноразрядный полусумматор это устройство для сложения разрядов двух чисел без учета переноса из предыдущего разряда и имееющее два входа (два суммируемые разряды) и два выхода (суммы и переноса).

Таблица истинности, отражающая алгоритм работы полусумматора, имеет следующий вид.

Таблица 24.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | x | y | C | П |  |
|  | 0 | 0 | 0 | 0 |  |
|  | 0 | 1 | 1 | 0 |  |

x y

&

&

1

&

П

C а)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 0 | 1 | 0 | Логическая схема, соответствующая записанной |
|  | 1 | 1 | 0 | 1 | системе булевых функций, имеет вид (рис.31а): |

Данная схема может быть упрощена, если функция C будет записана на нулевых наборах и использована совместная минимизация.







## 

## 

Рис. 31. Логическая схема полусумматора до и после упрощения

### 

### Синтез одноразрядного полного комбинационного сумматора на двух полусумматорах

Согласно рассмотренному выше материалу функция суммирования для полного комбинационного сумматора имеет вид

,

введем обозначение ,

.

Аналогично можно выполнить преобразование функции переноса П.



Таким образом, схема полного сумматора на двух полусумматорах будет иметь следующий вид (рис. 32).



### Синтез одноразрядного комбинационного вычитателя

Пусть имеется два числа

A=a1a2 . . . a i-1a ia i+1 . . . an

B=b1b2 . . . b i-1bib i+1 . . . bn

В зависимости от значений аргументов ai, bi, zi формируется значение булевых функций Ci, и Пi. Введем следующие обозначения.

ai  ⇒ x Рi ⇒ Р - функция разности

bi  ⇒ y Зi ⇒ З - функция заема

zi ⇒ z - заем из i-го разряда

Таблица истинности, соответствующая устройству, выполняющему операцию вычитания, имеет следующий вид.

Таблица 25.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | x | y | z | Р | З |  |
|  | 0 | 0 | 0 | 0 | 0 | Выполним склеивания 1 и 3, 2 и 3, 3 и 4 наборов |
|  | 0 | 0 | 1 | 1 | 1 |  |
|  | 0 | 1 | 0 | 1 | 1 |  |
|  | 0 | 1 | 1 | 0 | 1 | С |
|  | 1 | 0 | 0 | 1 | 0 | Как видно функция разности P и суммы С сов- |
|  | 1 | 0 | 1 | 0 | 0 | падают (функция С получена ранее). |
|  | 1 | 1 | 0 | 0 | 0 |  |
|  | 1 | 1 | 1 | 1 | 1 |  |

### Объединенная схема одноразрядного комбинационного сумматора-вычитателя

Для системы булевых функций C, P, П и З, полученных выше, может быть построена следующая логическая схема (рис. 33) на элементах И, ИЛИ и НЕ.



### Триггер со счетным входом как полный одноразрядный сумматор

Триггером называется устройство, имеющее два устойчивых состояния и способное под действием входного сигнала скачком переходить из одного устойчивого состояния в другое. Триггер – это простейший цифровой автомат с памятью и способностью хранить 1 бит информации.

Триггер со счетным входом (Т-триггер) может быть рассмотрен как полный сумматор, работающий в три такта.

В основе работы этого устройства лежит операция ”сумма по модулю 2”.

**1-й такт**. 1) τ(t-1)=0 – триггер находится в исходном состоянии.

2) Т=x первое слагаемое подается на вход триггера

,

следовательно, после первого такта содержимое триггера будет соответствовать его входному сигналу.

**2-й такт**. Во втором такте на вход триггера подается второе слагаемое y.

,

на выходе триггера формируется сумма по модулю 2 слагаемых x и y.

**3-й такт**. На третьем такте на вход триггера поступает значение, соответствующее третьему слагаемому – z.



.

Из полученной функции видно, что на выходе T-триггера получена полная сумма трех слагаемых и, следовательно, триггер со счетным входом является полным сумматором, работающим в три такта.

# Введение в теорию конечных автоматов

## Основные понятия теории автоматов

Все рассмотренные выше устройства относятся к классу комбинационных схем, то есть 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 как 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* теорию автоматов. 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 прежде всего 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: структурная и абстрактная. 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,δ,λ,a1),

где А={aa,…,am} – множество внутренних состояний абстрактного автомата, Z=[z1,…,zk} и W={w1,…,wl} – соответственно множества входных и выходных абстрактных слов, δ - функция переходов, λ - функция выходов, a1 – начальное состояние автомата. Абстрактный автомат может быть представлен как устройство с одним входом и одним выходом (рис. 34) на которые подаются абстрактные входные слова и формируются абстрактные выходные слова:



Понятие *состояния автомата* используется для описания систем выходы которых зависят не только от входных сигналов, но и от предыстории, то есть 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 Мили* или автоматами первого типа называют автоматы, для которых выходной символ w(t) не завит явно от входного символа z(t), а определяется только состоянием автомата в момент времени t. Закон функционирования автомата Мура может быть описана системой уравнений.



К автоматам второго типа или *автоматам Мили*, относятся автоматы поведение которых может быть описано системой уравнений.



Следовательно, в отличие от автомата Мура для автомата Мили выходной символ w(t) зависит не только от текущего состояния, но и от входного символа.

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,δ,λ1,λ2,a1), где А, Z, δ и а1 – определены выше, а W={w1,…,wl} и U={u1,…,ul} – выходной абстрактный алфавит автомата Мили и Мура соответственно, λ1 и λ2 - функции выходов. Абстрактный С-автомат может быть представлен как устройство с одним входом на который поступают слова из входного алфавита X и двумя выходами (рис. 35), на которых формируются абстрактные входные слова из выходных алфавитов W и U.



Отличие С-автомата от моделей автоматов Мили и Мура заключается в том, что он одновременно реализует две функции выходов λ1 и λ2  каждая из которых характерна для одной из двух моделей.



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 одного и того же входного символа. Состояние аs называется *устойчивым,* если для любого zk∈Z такого, что as=δ(am,zk) as=δ(as,zk). Автомат S является *асинхронным,* если каждое его состояние устойчиво, иначе *синхронным*.

Автомат называется *полностью определенным*, если область определения функции δ совпадает с множеством пар (am,zk), f функции λ для автомата Мили с множеством пар (am,zk), Мура с am. У *частичного* автомата функции δ и λ определены не для всех пар. Автомат является *инициальным* если в нем выделено начальное состояние а1.

## Способы задания автоматов

Закон функционирования автоматов может быть задан в виде систем уравнений, таблиц 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). С помощью таблиц 26 и 27 задан закон функционирования абстрактного автомата Мили для которого

A={a1,a2,a3,a4}, Z={z1,z2,z3}, W={w1,w2,w3,w4,w5}

 Строки таблиц отмечены входными символами (элементы множества Z), а столбцы состояниями (элементы множества А). Входные символы и состояния которыми помечены строки и столбцы относятся к моменту времени t. В таблице 26 (переходов) на пересечении сроки zi(t) и столбца am(t) ставится состояние as(t+1)=δ(am(t),zi(t)). В таблице 27 (выходов) на пересечении сроки zi(t) и столбца am(t) ставится выходной символ w(t)=λ(am(t),zi(t)), соответствующий переходу из состояния аm в состояние as. Таким образом, по таблицам переходов и выходов можно проследить последовательность работы автомата. Так, например, в начальный момент времени t=0 автомат, находясь в состоянии a1 (первый столбец) под действием входного символа z1 может перейти в состояние a2 при этом выходной символ не формируется, символа z2 в состояние a4 с формированием выходного символа w2, z3 в состояние a3 с формированием выходного символа w3. Далее если на вход автомата, установленного в исходное состояние аm ⊆A в моменты времени t=1,2,…,n подается некоторая последовательность букв входного алфавита (входных символов) zi⊆Z, то на выходе автомата будут последовательно формироваться буквы выходного алфавита (выходные символы) wj⊆W, при этом автомат будет переключаться в состояния as⊆A. Следовательно, с помощью таблиц переходов и выходов можно получить выходную реакцию автомата на любое входное слово.

Как отмечалось выше, для автомата Мура выходной символ не зависит от 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 переходов и выходов каждый столбец отмечается состоянием am ⊆А и выходным символом w(t)=λ(a(t)), соответствующим этому состоянию.

Другим более наглядным способом описания закона функционирования автомата является представление его в виде графа. Граф автомата – ориентированного граф, вершины которого соответствуют состояниям, а дуги переходам между ними. Дуга, направленная из вершины am в вершину as соответствует переходу из состояния am в as. Вначале дуги записывается входной символ zi влияющий на переход as=δ(am,zi), а символ wj записывается в конце дуги (автомат Мили) или вначале (автомат Мура). На рис. 36 приведен граф автомата Мили соответствующий закону функционирования, описанному выше (таблицы 26 и 27).



## Структурный автомат

В отличие от абстрактного автомата структурный автомат имеет L входов и N выходов. На входы структурного автомата поступают наборы входных двоичных переменных из множества X={x1,x2,…,xL}, а на выходах формируются выходные двоичные сигналы из множества Y={y1,y2,…,yN}. Структурная модель автомата представляет собой две взаимосвязанные части: комбинационную и память. Комбинационная часть автомата кроме сигналов из множества Y так же формирует двоичные сигналы, подаваемые на входы элементов памяти D={d1,d2,…,dr}. Эти сигналы называются *функциями возбуждения элементов памяти* и представляют собой код состояния перехода. Сигналы, формируемые на выходах элементов памяти T={τ1,τ2,…,τr} подаются на входы комбинационной схемы наряду с входными переменными и называются *переменными обратной связи*. Переменные обратной связи являются кодом текущего состояния автомата. Структурная схема автомата изображена на рис. 37.



## Память автомата

Под элементом памяти (триггером) подразумеваются простейшие схемы, которые предназначены для приема, хранения и передачи одного бита информации. 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) иесинхронизируемые (синхронные) триггеры. У асинхронных триггеров запись информации происходит под действием информационных сигналов, у синхронных под действием разрешающих сигналов;

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 (Т-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 так далее), 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;

Назначение входов J и K аналогичное входам R и S.

D- информационный вход. Используется для приема информации записываемой в триггер;

T- счетный вход;

С- вход синхронизации.

***D-триггер***. Принцип работы синхронного D-триггера основан на том, что сигнал на выходе после переключения, равен сигналу на входе D до переключения. Основное назначение D-триггера – задержка сигнала, поданного на вход D. D-триггеры могут быть построены по различным схемам. На рис. 38 приведена схема одноступенчатого D-триггера на элементах И-НЕ и его условное изображение.



В таблице 28 приведена информация работе D-триггера. Переключение состояний выполняется по формуле τ(t+1)= τ(t)С V DC.



***T-триггер***. Принцип работы синхронного D-триггера основан на том, что сигнал на выходе после переключения, равен сигналу на входе D до переключения. Основное назначение D-триггера – задержка сигнала, поданного на вход D. D-триггеры могут быть построены по различным схемам. На рис. 39 приведена схема одноступенчатого D-триггера на элементах И-НЕ и его условное изображение.



В таблице 29 приведена информация работе Т-триггера. Переключение состояний выполняется по формуле τ(t+1)= τ(t) ⊕ T.



***RS-триггеры***. Асинхронные RS-триггеры являются наиболее простыми триггерами. Триггеры такого типа построены на двух логических элементах :2 ИЛИ-НЕ – триггер с прямыми входами (рис. 40 ) и 2 И-НЕ – триггер с инверсными входами. Выход каждого из логических элементов подключен к одному из входов другого элемента, что обеспечивает триггеру два устойчивых состояния.



Таблица 30 определяет переходы RS-триггера по формуле

τ(t+1)= τ(t)R V S.

Таблица 30. Состояния RS-триггера.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | R S    S | | | |
| 00 | 01 | 10 | 11 |
| 0 | 0 | 1 | 0 | x |
| 1 | 1 | 1 | 0 | x |

Возможны следующие режимы работы RS-триггера:

S=0, R=0 – режим хранения информации (значение триггера не изменяется);

S=0, R=1 – режим сброса (триггер всегда устанавливается в 0);

S=1, R=0 – режим записи логической единицы (триггер устанавливается в 1);

S=1, R=1 – запрещенная комбинация (значение триггера не неопределенное).

***JK-триггеры.*** Асинхронный двухступенчатый JK-триггер строится на базе RS-триггера. JK-триггер имеет два информационных входа. 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 условное обозначение JK-триггера приведены на рис. 41.



Таблица 31 определяет переходы JK-триггера согласно логической формулы τ(t+1)= τ(t)J V τ(t)K .

Таблица 31. Состояния JK-триггера.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | J K    S | | | |
| 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 всегда устанавливается в 0);

J=1, K=0 – режим записи логической единицы (триггер устанавливается в 1);

J=1, K=1 – режим инверсии содержимого триггера.

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.

## Канонический метод синтеза

Синтез цифровых устройств выполняется в два этапа:

* этап абстрактного синтеза;
* этап структурного синтеза.

Для перехода от абстрактного автомата к его структурной схеме требуется:

1) поставить каждой букве входного алфавита Z=[z1,…,zk} совокупность двоичных сигналов из множества X={x1,x2,…,xL}, то *есть закодировать входные символы* абстрактного автомата. Значение L вычисляется следующим образом L=intlog2|X| = [log2|X|], где |X| - мощность множества Х (число различных элементов множества Х);

2) поставить в соответствие каждому выходному символу из W={w1,…,wl} совокупность двоичных выходных сигналов из множества Y={y1,y2,…,yN}, то *есть закодировать выходные символы* абстрактного автомата. Значение N вычисляется следующим образом N=intlog2|Y| = [log2|Y|];

3) поставить в соответствие каждому состоянию абстрактного автомата А={aa,…,am} совокупность состояний элементов памяти T={τ1,τ2,…,τr}, то есть *закодировать состояния* абстрактного автомата. Количество элементов памяти выбирается из условия r=intlog2|А| = [log2|А|];

4) составить систему уравнений для функций y1,y2,…,yN,d1,d2,…,dr предназначенной для построения логической схемы комбинационной части структурной схемы.

Полученная таким образом система логических функций называется *канонической*.

Рассмотрим пример структурного синтеза автомата Мили блок памяти которого построен на Т-триггерах. Исходные данные для выполнения синтеза структурной схемы заданы таблично (таблицы 32 и 33). Таблица истинности работы Т-триггера приведена в таблице 29.

Определяем вначале общее количество входов, выходов и элементов памяти автомата.



L=intlog2|X| = [log2|X|]= [log24]=2;

N=intlog2|Y| = [log2|Y|]= [log24]=2;

r=intlog2|А| = [log2|А|]= [log26]=3;

Структурная схема автомата изображена на рисунке (см. рис.42).

 На основании полученных значений построим таблицы и выполним кодирование входных, выходных символов и состояний исходного автомата (таблица 34).



На основании кодирования строим таблицы переходов и выходов структурного автомата (таблицы 35 и 36). Они получаются путем занесения соответствующих значений из таблицы 34 в исходные таблицы 32 и 33.





Используя таблицу переходов Т-триггера (табл. 29) построим таблицу 37 - функций возбуждения элементов памяти по таблице 35.

Если r-й триггер на некотором переходе переключается из состояния 0 в состояние 1, или наоборот, то ui=1, в противном случае (то есть если r-й триггер не переключается) ui=0. Например, рассмотрим переход из состояния 10 в состояние 11 (3-й столбец 3-я строка). Первый триггер (установленный в 1) не меняет своего значения, поэтому функция возбуждения элемента памяти для него u1=0. Второй триггер изменяет свое значение с 0 на 1, следовательно u2=1.Остальные клетки таблицы 37 заполняются аналогично. На основании таблиц 36 и 37 запишем систему логических функций для построения комбинационной схемы автомата.

Для упрощения комбинационной схемы выполним минимизацию каждой из логических функций. Для минимизации функций используем метод минимизирующих карт Карно. На рисунке 43 изображены 5 карт Карно для минимизации каждой из пяти логических функций



 По результатам минимизации запишем систему минимальных функций.

 На рис. 36. изображена логическая схема, построенная для полученной системы булевых функций. При построении схемы использованы элементы ”И” и ”ИЛИ”.



## Пример синтеза МПА Мили по ГСА

Как отмечалось выше известны два типа автоматов:Мили и Мура. В качестве примера рассмотрим синтез микропрограммного автомата управляющего операционным автоматом для выполнения деления чисел в дополнительных кодах. ГСА соответствующая алгоритму деления изображена на рис. 45.



После пробного вычитания Зн См может быть равен 0, это означает что Дм больше Дт (переполнение). В этот момент СТ равен 0, деление прекращается (переход в конец по стрелке 2). В последних тактах Зн См может быть равен 0. Это означает, что Аi > Дт, но СТ уже не в нуле и алгоритм выполняется по стрелке 4. Если Зн См равен 1, то остаток отрицательный и деление пойдет по стрелке 3.

Алгоритм синтеза МПА Мили по ГСА состоит в:

* разметке ГСА метками Мили;
* кодировании внутренних состояний;
* построении таблицы переходов по отмеченной ГСА:
* построении таблиц истинности или системы булевых функций:
* построении логической схемы автомата.

Для получения графа автомата Мили исходная ГСА отмечается метками Мили. Каждой метке на ГСА ставится во взаимно однозначное соответствие состояние автомата. Алгоритм отметки ГСА метками Мили состоит в следующем:

* Выход начальной и вход конечной вершин отмечаются меткой а1;
* Входы всех вершин, следующих за операторными отмечаются метками а2,…,аm.;
* Одной меткой может быть отмечен только один вход.

На рис. 46 приведена отмеченная метками Мили ГСА.

Кодирования состояний автомата может быть выполнено как и ранее если каждому состоянию поставить в соответствие двоичный эквивалент номера состояния. Для нахождения всевозможных переходов автомата на отмеченной ГСА отыскиваются все пути вида



При достаточно большом числе состояний и переходов на ГСА наглядность задания автомата графом теряется. В этом случае более удобным является задание автомата структурной таблицей, содержащей всю необходимую для синтеза информацию. Структурная таблица может быть прямой или обратной. В *прямой* таблице (табл. 38) вначале записываются все переходы из состояния а1, затем из состояния а2 и так далее. В *обратной* таблице сначала все переходы в состояние а1, затем в а2 и так далее.



В последнем столбце отмечены те функции возбуждения, которые являются обязательными. Обязательными функциями переключения памяти на каждом переходе является множество сигналов приводящих к изменению содержимого каждого из элементов памяти на соответствующем переходе. В таблице 39 в столбце F(amas) приведены функции переключения элементов памяти для случая реализации ее на RS-триггере.



## Синхронизация автоматов

Нарушением функционирования автомата может быть вызвано явлениями, получившими название гонки и риск сбоя.

*Гонки* возникают из-за неодновременного срабатывания элементов памяти автомата. Например, пусть под действием некоторого входного сигнала X(amas) с кодом 00 автомат должен перейти из состояния am с кодом 101 состояние as с кодом 110 (см. рис. 47). Если второй триггер изменит свое значение (переключится) с 0 в 1 ранее чем третий переключится с 1 в 0, то автомат перейдет в промежуточное состояние 111. Иначе, если третий триггер сработает ранее второго, то в промежуточное состояние 100.

Таким образом, если на некотором переходе в автомате одновременно изменяют свое состояние несколько элементов памяти, то между ними возникает ”состязание”. Если из промежуточного состояния автомат, в конечном счете, переходит в требуемое состояние as, то называются ”состязание” некритическими, если в ложное, например 011, то критическими или гонками.



Существует два основных подхода к устранению гонок: программный и аппаратный. Программный (алгоритмический) подход основан на соседнем кодировании состояний. При соседнем кодировании состояний автомата из множества A={a1,…,am} кодируются таким образом, что на любом переходе изменяют свое состояние не более чем один элемент памяти. Однако соседнее кодирование воз

можно выполнить не для всех автоматов.

Аппаратный подход основан на использовании двух ступеней памяти рис. 48. Первая ступень памяти построена на триггерах T1,…,Tr, вторая на триггерах T1,…,Tr. Информация в триггеры первого уровня T1,…,Tr записывается по тактовому сигналу Ти, а в триггеры второго уровня T1,…,Tr по сигналу Ти следующему непосредственно за Ти.



Если в течении первого полупериода (Ти) между триггерами первой ступени и возникают ”состязания”, то они все равно не изменяют состояния триггеров второй ступени, поскольку отсутствует синхросигнал Ти. Затем, с приходом синхроимпульса Ти изменяют свое состояние триггеры второй ступени. Промежуточные коды формируемые на их выходах приводят к изменению (и искажению) τ1,…,τr, а следовательно и D1,…,Dr. Однако триггеры первой ступени не изменят своего состояния, поскольку отсутствует сигнал Ти. Таким образом, в итоге верный код состояния as с выходов памяти первой ступени переписывается в триггера второго уровня, что соответствует переходу автомата в состояние as.

Если комбинационная схема автомата построена из синхронизируемых элементов, то гонки так же устраняются путем разделения синхронизации памяти и комбинационной схемы (рис. 49). В этом случае двойная память не требуется.

*Риск сбоя*. Наличие некритических ”состязаний”, не нарушает правил перехода в автомате, но создает возможность возникновения раска 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 при переходе из состояния 101 в состояние 110 появляется кратковременный сигнал yk в промежуточном состоянии 00 (см. рис. 50).



Хотя длительность ложного сигнала yk достаточно мала, однако его возникновение может привести к непредсказуемым последствиям в устройстве, которым управляет автомат. Некоторые методы устранения риска сбоя состоят в следующем.

- Выходы автомата, на которых может возникнуть риск сбоя? соединяются через конденсатор небольшой емкости с нулевым выходом источника питания.



- Буферизация выходных сигналов.

# Литература

1. Савельев А.Я. Прикладная теория цифровых автоматов. Учеб. Для вузов по спец ЭВМ. –М.: Высш. шк., 1987.

2. Поснов Н.Н. Арифметика вычислительных машин в упражнениях и задачах: системы счисления, коды. –Мн.: Изд-во ”Университетское”, 1984.

3. Морозевич А.Н. Дмитриев А.Н. и др. Микро-ЭВМ микропроцессоры и основы программирования. –Мн.: Выш. шк., 1990

4. Акушинский И.Я., Юдицкий Д.И. Машинная арифметика в остаточных классах –М.:”Советское радио”, 1968.

5. Питерсон У., Уэлдон Э. Коды исправляющие ошибки./Пер. с англ. . –М.: Мир, 1976.

[Введение 3](#_Toc57576233)

[Арифметические основы вычислительной техники 4](#_Toc57576234)

[Системы счисления 4](#_Toc57576235)

[Двоичная система счисления 5](#_Toc57576236)

[Восьмеричная система счисления 5](#_Toc57576237)

[Шестнадцатеричная система счисления 6](#_Toc57576238)

[Критерии выбора системы счисления 7](#_Toc57576239)

[Перевод чисел из одной системы счисления в другую 10](#_Toc57576240)

[Перевод целых чисел. 10](#_Toc57576241)

[Перевод правильных дробей. 11](#_Toc57576242)

[Перевод чисел из системы счисления в систему счисления основания которых кратны степени 2 12](#_Toc57576243)

[Кодирование чисел 12](#_Toc57576244)

[Переполнение разрядной сетки 15](#_Toc57576245)

[Модифицированные коды 16](#_Toc57576246)

[Машинные формы представления чисел. 16](#_Toc57576247)

[Погрешность выполнения арифметических операций 18](#_Toc57576248)

[Округление 19](#_Toc57576249)

[Нормализация чисел 19](#_Toc57576250)

[Последовательное и параллельное сложение чисел 20](#_Toc57576251)

[Сложение чисел с плавающей запятой 21](#_Toc57576252)

[Машинные методы умножения чисел в прямых кодах 22](#_Toc57576253)

[Ускорение операции умножения 26](#_Toc57576254)

[Умножение с хранением переносов 26](#_Toc57576255)

[Умножение на два разряда множителя одновременно. 27](#_Toc57576256)

[Умножение на четыре разряда одновременно. 29](#_Toc57576257)

[Умножение в дополнительных кодах. 30](#_Toc57576258)

[Умножение на 2 разряда Мт в дополнительных кодах. 34](#_Toc57576259)

[Матричные методы умножения. 36](#_Toc57576260)

[Машинные методы деления 38](#_Toc57576261)

[Деление чисел в прямых кодах. 38](#_Toc57576262)

[Деление чисел в дополнительных кодах. 40](#_Toc57576263)

[Методы ускорения деления. 41](#_Toc57576264)

[Двоично-десятичные коды 41](#_Toc57576265)

[Суммирование чисел с одинаковыми знаками в коде 8421. 43](#_Toc57576266)

[Сложение чисел с разными знаками. 45](#_Toc57576267)

[Двоично-десятичные коды с избытком 3 46](#_Toc57576268)

[Код с избытком 6 для одного из слагаемых 47](#_Toc57576269)

[Система счисления в остаточных классах (СОК) 48](#_Toc57576270)

[Представление отрицательных чисел в СОК 51](#_Toc57576271)

[Контроль работы цифрового автомата 52](#_Toc57576272)

[Некоторые понятия теории кодирования 53](#_Toc57576273)

[Обнаружение и исправление одиночных ошибок путем 54](#_Toc57576274)

[использования дополнительных разрядов 54](#_Toc57576275)

[Коды Хемминга 55](#_Toc57576276)

[Логические основы вычислительной техники 57](#_Toc57576277)

[Двоичные переменные и булевы функции 57](#_Toc57576278)

[Способы задания булевых функций 58](#_Toc57576279)

[Основные понятия алгебры логики 59](#_Toc57576280)

[Основные законы алгебры логики 62](#_Toc57576281)

[Формы представления функций алгебры логики 63](#_Toc57576282)

[Системы функций алгебры логики 65](#_Toc57576283)

[Минимизация ФАЛ 70](#_Toc57576284)

[Метод Квайна 71](#_Toc57576285)

[Метод Блейка - Порецкого 73](#_Toc57576286)

[Метод минимизирующих карт Карно (Вейча) 74](#_Toc57576287)

[Минимизация коньюнктивных нормальных форм. 77](#_Toc57576288)

[Минимизация не полностью определенных ФАЛ 78](#_Toc57576289)

[Кубическое задание функций алгебры логики. 79](#_Toc57576290)

[Метод Квайна-Мак Класки 82](#_Toc57576291)

[Алгоритм извлечения (Рота) 84](#_Toc57576292)

[Минимизация ФАЛ методом преобразования логических выражений 93](#_Toc57576293)

[Применение правил и законов алгебры логики к синтезу некоторых цифровых устройств 93](#_Toc57576294)

[Синтез одноразрядного полного комбинационного сумматора 93](#_Toc57576295)

[Синтез одноразрядного комбинационного полусумматора 94](#_Toc57576296)

[Синтез одноразрядного полного комбинационного сумматора на двух полусумматорах 95](#_Toc57576298)

[Синтез одноразрядного комбинационного вычитателя 95](#_Toc57576299)

[Объединенная схема одноразрядного комбинационного сумматора-вычитателя 96](#_Toc57576300)

[Триггер со счетным входом как полный одноразрядный сумматор 97](#_Toc57576301)

[Введение в теорию конечных автоматов 97](#_Toc57576302)

[Основные понятия теории автоматов 97](#_Toc57576303)

[Способы задания автоматов 99](#_Toc57576304)

[Структурный автомат 101](#_Toc57576305)

[Память автомата 102](#_Toc57576306)

[Канонический метод синтеза 105](#_Toc57576307)

[Пример синтеза МПА Мили по ГСА 110](#_Toc57576308)

[Синхронизация автоматов 114](#_Toc57576309)

[Литература 117](#_Toc57576310)