**За «спасибо» в tg @byaketava**

## 1. СИСТЕМЫ СЧИСЛЕНИЯ. КРИТЕРИИ ВЫБОРА СИСТЕМЫ СЧИСЛЕНИЯ.

**Система счисления** – это совокупность приёмов и правил записи чисел ограниченным количеством символов

**Требования к системе:**
− возможность представления любого числа в рассматриваемом диапазоне величин (в сс);
− единственность этого представления;
− простоту выполнения операций.

**3 категории:**

1) позиционные СС; (позиция имеет вес, значение цифры зависит от положения, 123/231)
Основание r СС – количество цифр, использующееся в данной СС
Базис СС – перечень степеней этого основания (0, 1, 2 … n)
Вес разряда Pi=ri/r0
Алфавит – совокупность всех цифр(символов) используемых для записи числа

2) непозиционные СС; (позиция не имеет значения (основание «1», римская СС))

3) смешанные СС. (число как линейная комбинация. (время, градусы и т.д.))
Записью числа в смешанной системе счисления называется перечисление его цифр в порядке уменьшения индекса

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

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

или

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

В технической реализации *длина числа* – длина разрядной сетки, а вообще – кол-во позиций в записи.

**<u>Критерии выбора системы счисления:</u>**

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

***2. Наибольшая помехоустойчивость кодирования цифр***.
При технической реализации любой СС диапазон изменения электрического значения наибольшей и наименьшей цифры одинаков. Очевидно преимущество систем с меньшим основанием, так как представление соседних цифр в этих системах отличается друг от друга больше, чем для систем с большим основанием. Так, при наложении помехи на основной сигнал, наиболее вероятна ошибка в устройствах, для которых используется система счисления с наибольшим основанием. Следовательно, при увеличении основания помеха может привести к искажению числа.

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

***4. Простота арифметических действий*.**
Чем меньше цифр в системе счисления, тем проще арифметические действия над ними.

***5. Наибольшее быстродействие.***
Время, затрачиваемое на формирование и выполнение переноса учитывается
Максимальное время сложения у двоичной системы т.к. в СС с меньшим основанием число записывается бОльшим количеством цифроразрядов.

***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 СС для выполнения операций человеком является десятичная. Но в ЭВМ для выполнения арифметических операций числа из десятичной требуется переводить во внутреннюю систему счисления. Для системы счисления с основанием, большим 10, появляются новые цифры. Таким образом, система счисления должна иметь минимальное число цифр, так как в этом случае можно пользоваться младшими цифрами десятичной системы счисления.

***8. Возможность представления любого числа в рассматриваемом диапазоне величин***.
Обеспечивает любая позиционная система счисления.

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

## 2. ПЕРЕВОД ЦЕЛЫХ ЧИСЕЛ ИЗ ОДНОЙ СИСТЕМЫ СЧИСЛЕНИЯ В ДРУГУЮ.

*- Метод подбора степеней основания.*
*- Метод деления на основание системы счисления.*
***Основная формула переписывается по схеме горнера, потом делится на основание новой СС, получается остаток и целое частное. Так делят до конца***

## 3. ПЕРЕВОД ДРОБНЫХ ЧИСЕЛ ИЗ ОДНОЙ СИСТЕМЫ СЧИСЛЕНИЯ В ДРУГУЮ.

*- Метод подбора величин, обратных степеням основания*
*- Метод умножения на основание $r_2$ новой системы счисления.*
***целая часть даёт цифру в число, а дробная дальше в умножение***

Существует перевод чисел из одной сс в другую если основание кратно степени 2
***Все действия в исходной выполняются***
Переводим дробную пока не получим нули или заданную точность

## 4. КОДИРОВАНИЕ ЧИСЕЛ (ПРЯМОЙ, ОБРАТНЫЙ, ДОПОЛНИТЕЛЬНЫЙ КОД).

Знаковый разряд, 1(r-1, где r – основание) – минус, 0 – плюс
1 байт – 8 бит
«код знака , код числа»

***ПРЯМОЙ КОД ЧИСЛА.***
При этом способе кодирования чисел кодируется знак, а значащая часть остается без изменения.

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

Недостаток – сложность выполнения сложения с разными знаками. Используется ***сумматор прямого кода***, в нём нет цепи поразрядного переноса между старшим значащим и знаковым разрядами, т.е. невозможно алгебраическое сложение.

***ДОПОЛНИТЕЛЬНЫЙ КОД ЧИСЛА.***

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

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

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

***Замена вычитания сложением в ЭВМ*:** большее по модулю число разместить на входах вычитающего устройства, итогу присвоить знак наибольшего по модулю числа.

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

***ОБРАТНЫЙ КОД ЧИСЛА.***

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

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

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

**Теорема**. Сумма обр.кодов есть обр.код суммы.

## 5. ПЕРЕПОЛНЕНИЕ РАЗРЯДНОЙ СЕТКИ. ПРИЧИНЫ И ПРИЗНАКИ ПЕРЕПОЛНЕНИЯ. МОДИФИЦИРОВАННЫЕ КОДЫ

Явление переполнения возникает (при операциях сложения/умножения) в случае если разрядной сетки недостаточно (т.е. значащей части не хватает для хранения полученной информации)

***Для обнаружения переполнения*** можно использовать следующие признаки:
− знаки слагаемых не совпадают со знаком суммы;
− есть перенос только в знаковый (П1) или только из знакового (П2) разряда.

```
if (П1 && ! П2)||(! 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 <u>не</u> фиксируется***. Это происходит тогда, когда сумма модулей двух отрицательных чисел равна *удвоенному весу* единицы старшего разряда числа.

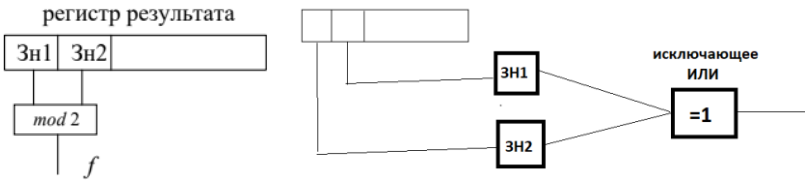
*Пример.* $A = -0{,}101$ $\quad [A]_{доп} = 1{,}011$

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

$[A+B]_{доп} = 1{,}000$

**Модифицированные коды** – знак числа кодируется не одним, а двумя разрядами (Зн1 и Зн2).

f = Зн1 (+) Зн2.




| | | |
|---|---|---|
| =1 | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

крайний левый знак верный всегда

## 6. ФОРМЫ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ В ЭВМ (ЧИСЛА С ФИКСИРОВАННОЙ ЗАПЯТОЙ (ТОЧКОЙ))

Представление чисел в форме с фиксированной запятой. Для сокращения длины разрядной сетки и упрощения обработки данных положение запятой может быть зафиксировано.

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

| | | |
|---|---|---|
| 1) $k=0,\ l=n\quad A_{max}=1-2^{-n}$ | ± | 1 1 . . . 1 1 |
| 2) $k=n,\ l=0\quad A_{max}=2^{n}-1$ | ± | 1 1 . . . 1 1 |
| 3) $k=0,\ l=n\quad A_{min}=2^{-n}$ | ± | 0 0 . . . 0 1 |
| 4) $k=n,\ l=0\quad A_{min}=1$ | ± | 0 0 . . . 0 1 |

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

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

***Целые*** числа в ЭВМ представляются в формате ***с фиксированной запятой***.
Возможны четыре формата представления целых чисел:
− целое число; (2/4 байта, отриц в доп коде)
− короткое целое число; (аналогично, но 4 байта)
− длинное целое число; (аналогично, но 8 байт)
− упакованное десятичное число.(10 байт, 18 цифр, 2 в каждом байте, в левом бите левого байта знак)

К ***достоинствам*** использования чисел с фиксированной запятой относится ***простота выполнения арифметических операций.***
К ***недостаткам*** – ограничение длины разрядной сетки приводит к ***ограничению диапазона*** хранимых чисел и ***потере точности*** их представления в случае дробных чисел.

## 7. ФОРМЫ ПРЕДСТАВЛЕНИЯ ЧИСЕЛ В ЭВМ (ЧИСЛА С ПЛАВАЮЩЕЙ ЗАПЯТОЙ (ТОЧКОЙ))

Общий вид чисел с плавающей запятой можно записать следующим образом:

$A = +\text{-}m_A\, r^{+\text{-}p}$

где r – основание СС, в которой записано число;
$m_A$ – мантисса числа
p − порядок числа.

$r^{-1} \le |m_A| < 1$

**Классическая сетка:**


В языках высокого уровня используется следующее представление чисел с плавающей запятой:
(знак)(мантисса)E(знак)(порядок)
Например, −5.35E−2 означает число $-5.35 \cdot 10^{-2}$.

В зависимости от типа данных числа с плавающей запятой в памяти ЭВМ хранятся в одном из следующих трех форматов:
− одинарной точности; 4 байта, смещение 127
− двойной точности; 8 байт, смещение 1023
− расширенной точности. 10 байт, смещение 16383

**ЭВМ сетка:**
**(*мантисса приводится к «1,…» и хранится без единицы, мантисса в прямом коде* )**
Для упрощения операций над порядками применяют представление чисел с плавающей запятой со смещенным порядком:
p' = p + N, где N – целое положительное число (**смещение**). p+127=p’
Обычно $N = 2^k - 1$, где k− количество цифр в записи порядка.
Поле знака порядка избыточно, так как p' всегда положительно.
Такие смещенные порядки называют ***характеристиками***.
Поле характеристики – это степень числа 2, на которую умножается мантисса, плюс смещение


Имеется два нуля – положительный и отрицательный.

***Наименьшее положительное число*** – *нулевой знаковый бит*, p=1, и значение *мантиссы*, равное *нулю*. **0 | 00000001 | 0…0**
$1{,}17 \cdot 10^{-38}$ (одинарная точность), $2.23 \cdot 10^{-308}$ (двойная точность), $3.37 \cdot 10^{-4932}$ (расширенная точность).

***Наибольшее отрицательное число*** аналогично, но бит знака – единица. **1 | 00000001 | 0…0**

***Наибольшее положительное число*** –нулевой знаковый бит, в поле порядка все биты *кроме* самого *младшего*, равны единице, единицы во всех разрядах мантиссы. **0 | 11111110 | 1…1**
$3.37 \cdot 10^{38}$ (одинарная точность), $1.67 \cdot 1^{308}$ (двойная точность), $1.2 \cdot 10^{4932}$ (расширенная точность).

***Наименьшее отрицательное число*** аналогично, но бит знака – единица. **1 | 11111110 | 1…1**

***Положительная и отрицательная бесконечность*** – все единицы в поле порядка, все нули в поле мантиссы. **0/1 | 11111111 | 0…0**
Бесконечность может получиться, например, как результат ***деления конечного числа на нуль.***

***Нечисло*** единицы в поле порядка, любое значение в поле мантиссы. Нечисло может возникнуть в результате выполнения ***неправильной операции при замаскированных особых случаях***.
**1 | 11111111 | x…x**

***Неопределенность*** в поле порядка единицы, в поле мантиссы – число 100...0 (для одинарной и двойной точности) или 1100...0 (для расширенной точности, так как в этом формате хранится старший бит мантиссы). **1 | 11111111 | 10…0**

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

## 8. ОКРУГЛЕНИЕ ЧИСЕЛ

В общем виде число с плавающей запятой, размещенное в разрядной сетке, имеет вид $A_r = +\text{-}m_a\, r^{+\text{-}k}$. Если для записи мантиссы используются только n разрядов, то число может быть представлено в виде двух частей: $A_r = [m_a]r^n + [A_0]r^{k-n}$, где $[A_0]r^{k-n} = A_0$ – часть числа, не вошедшая в разрядную сетку размерностью k.
***округлению подвергаются только числа с плавающей запятой***
1. **Отбрасывание $A_0$**. При этом возникает относительная погрешность
погрешность = $[A_0]\, r^{k-n} / [m_a]\, r^n$
2. **Симметричное округление**. При этом производится анализ величины $A_0$:

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

При условии $|A_0| \geq r^{-1}$ единица добавляется к младшему разряду мантиссы. Данный способ округления наиболее часто используется на практике.
По сути с 0,1 сравниваем часть после запятой

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 просто отбрасываются.

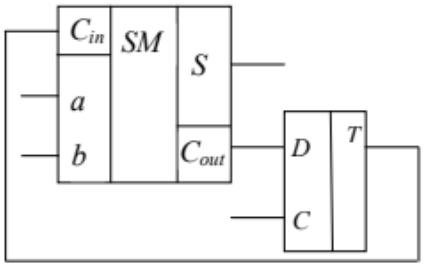
4. **Случайное округление**. Генератор случайных чисел формирует нулевое или единичное значение, посылаемое в младший разряд мантиссы.

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

## 9. ПОСЛЕДОВАТЕЛЬНОЕ СЛОЖЕНИЕ ЧИСЕЛ. ПОСЛЕДОВАТЕЛЬНЫЙ СУММАТОР

***Двоичный сумматор – устройство, предназначенное для выполнения арифметического сложения чисел в двоичном коде***
Для последовательного выполнения операции сложения (разряд за разрядом) используется ***один полный одноразрядный сумматор***.
Он имеет два входа (a и b) на каждый из которых последовательно, начиная с младшего, подаются разряды слагаемых (по одному за такт).

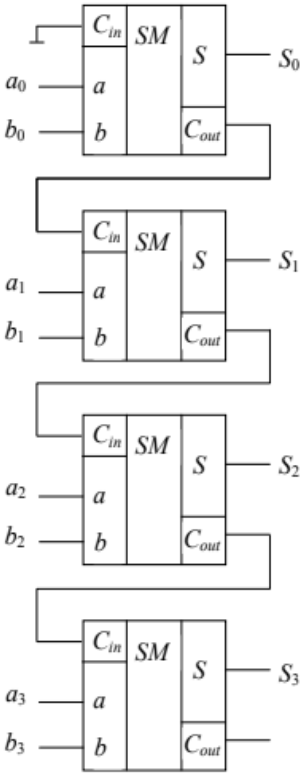
С выхода снимается значение соответствующего разряда суммы. Последнее определяется не только текущими значениями разрядов слагаемых, но и переносом из предыдущего разряда $C_{in}$.




Триггер в схеме для хранения значения переноса до следующего такта

## 10. ПАРАЛЛЕЛЬНОЕ СЛОЖЕНИЕ ЧИСЕЛ. ПАРАЛЛЕЛЬНЫЙ СУММАТОР.

***Параллельное быстрее, потому что в параллельном сумматоре нам не нужны сдвиги, мы захватываем сразу всё число. А в последовательном сумматоре мы ждём чтобы в регистрах первого и второго слагаемого сдвиг произошел, сложение и сдвиги в разных тактах***




***Четырёхразрядный параллельный сумматор с последовательным переносом***.

Для каждого разряда в этой схеме используется отдельный полный одноразрядный сумматор. В младший разряд (a0, b0) переноса нет, поэтому Cin = 0.
На каждый последующий разряд подается перенос из предыдущего. Хоть сумматор и называется параллельным, на самом деле все разряды обрабатываются не одновременно, а только после формирования переноса для данного разряда.
Отсюда следует, что ***быстродействие устройства*** определяется ***суммой задержек передачи сигнала переноса*** с младшего разряда на вход сумматора старшего разряда

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

## 11. НОРМАЛИЗАЦИЯ ЧИСЕЛ

***только для чисел с плавающей запятой***
1) денормализация влево + \ *
2) денормализация вправо - \ /

число нормализовано, если 1,0 или 0,1 т.е. мантисса между $r^{-1} \leq |M_A| < 1$
2 вида сдвигов:
– простой
– модифицированный

Простой сдвиг – сдвиг, выполняемый по правилу:

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

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

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

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

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

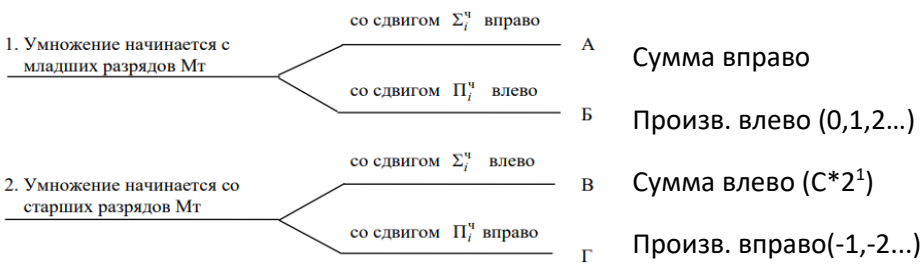
## 12. **СЛОЖЕНИЕ ЧИСЕЛ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ**

При сложении (вычитании) чисел складываемые цифры (разряды) ***должны иметь одинаковый вес***. Это требование выполняется, если складываемые (вычитаемые) числа имеют одинаковые порядки.

**Алгоритм сложения** (**вычитания**) чисел с произвольными знаками состоит в следующем.
**1**. Произвести **сравнение порядков** $p_A$ и $p_B$.
Если $p_A - \boldsymbol{p_B > 0}$, то $p_A > p_B$ и для выравнивания порядков необходимо ***сдвинуть*** вправо мантиссу $\boldsymbol{M_B}$.
Если $p_A - \boldsymbol{p_B < 0}$, то $p_B > p_A$ и для выравнивания порядков необходимо ***сдвинуть*** вправо мантиссу $\boldsymbol{M_A}$
**2**. Выполнить сдвиг соответствующей мантиссы на один разряд, повторяя его до тех пор, пока $p \neq 0$.
**3**. Выполнить **сложение** в модиф.кодах **мантисс** МА и МВ по правилу сложения правильных дробей.
**4**. Если при сложении (вычитании) мантисс произошло **переполнение**, то произвести ***нормализацию*** результата путем ***сдвига мантиссы вместе со знаковым разрядом вправо на один разряд с увеличением порядка на единицу***.
Если же произошла **денормализация**, то выполнить ***сдвиг мантиссы результата на соответствующее количество разрядов влево с соответствующим уменьшением порядка суммы***.

## 13. УМНОЖЕНИЕ ЧИСЕЛ В ПРЯМЫХ КОДАХ

Умножение есть ряд последовательных сложений. Если в разрядите множителя 1, то к сумме прибавляется множимое, 0 – ничего. Потом сдвиг множителя/частичной суммы. Начинать можно как с младших, так и со старших разрядов. Знак определяется сложением по модулю 2 знаковых разрядов.



***Алгоритм А.***

1. Обнуляем содержимое сумматора.
2. Формируем очередное **частичное произведение** $П_i = b_{n-i} * A$
3. Формируем новую **частичную сумму** $\sum_i = \sum_{i-1} + П_i$
4. Выполняем **сдвиг** полученной частичной суммы **вправо** на один разряд $\sum_i * 2^{-1}$. При этом ***содержимое младшего разряда*** сумматора переносится ***в*** освободившийся ***старший разряд регистра множителя***, в котором тоже выполняется сдвиг всех разрядов на один вправо.
5. Увеличиваем счетчик i числа выполненных умножений на единицу. Если I == n, то переходим к действию 6, иначе возвращаемся к действию 2.
6. Получено произведение Мн * Мт. Старшие разряды в сумматоре, младшие − в регистре Мт.

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

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

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

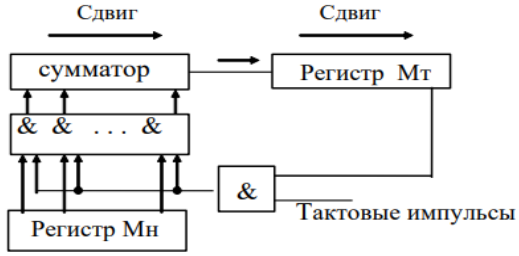
$$= 0 + \left(b_1 \cdot 0, a_1 \ldots a_n\right) \cdot 2^{-1} + \left(b_2 \cdot 0, a_1 \ldots a_n\right) \cdot 2^{-2} + \ldots + \left(b_n \cdot 0, a_1 \ldots a_n\right) \cdot 2^{-n} =$$

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

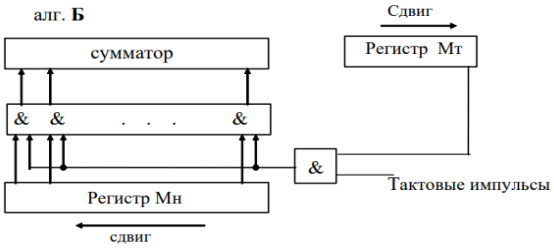
$$= 0 + b_n \cdot A \cdot 2^{-n} + b_{n-1} \cdot A \cdot 2^{-(n-1)} + \ldots + b_2 \cdot A \cdot 2^{-2} + b_1 \cdot A \cdot 2^{-1} =$$

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

алг. **А**




алг. **Б**

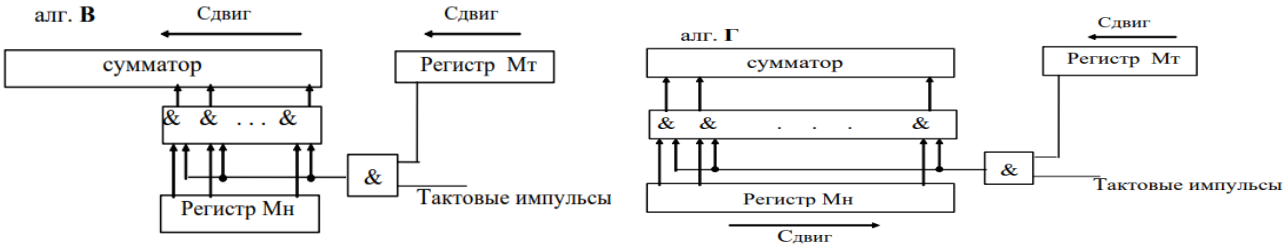



***Алгоритм Б.***

1. Разрядность сумматора равна количеству разрядов множителя + количество разрядов множимого
2. Обнуляем содержимое сумматора результата $\sum_i$ , i = 0.
3. Формируем очередное **частичное произведение** $П_i = b_{n-i} * A$
4. Выполняем **сдвиг полученного частичного произведения вправо на один разряд** $П_i * 2^{i-1}$
5. Формируем новую **частичную сумму** $\sum_i = \sum_{i-1} + П_i * 2^{i-1}$
6. . Увеличиваем счетчик i числа выполненных умножений на единицу. Если I == n, то переходим к действию 6, иначе возвращаемся к действию 3.
7. Получено искомое произведение Мн * Мт.

***Алгоритм В.***

1. Разрядность сумматора m равна количеству разрядов множителя + количество разрядов множимого
2. Обнуляем содержимое сумматора результата $\sum_i$ , i = 0.
3. Формируем очередное **частичное произведение** $П_i = b_i * A$
4. Формируем новую **частичную сумму** $\sum_i = \sum_{i-1} + П_i$
5. Если i < n, то выполняем **сдвиг полученной частичной суммы влево** на один разряд $\sum_i * 2$.
6. . Увеличиваем счетчик i числа выполненных умножений на единицу. Если I == n, то переходим к действию 6, иначе возвращаемся к действию 3.
7. Получено искомое произведение Мн * Мт.




***Алгоритм Г.***

1. Разрядность сумматора равна количеству разрядов множителя m + количество разрядов множимого.
2. Обнуляем содержимое сумматора результата $\sum_i$ , i = 0.
3. Формируем очередное **частичное произведение** $П_i = b_i * A$
4. Выполняем **сдвиг полученного частичного произведения влево на один разряд** $П_i * 2^{m-i}$
5. Формируем новую **частичную сумму** $\sum_i = \sum_{i-1} + П_i * 2^{m-i}$
6. Увеличиваем счетчик i числа выполненных умножений на единицу. Сравниваем его с n. Если они равны (выполнено умножение на все разряды множителя), то переходим к действию 7, иначе возвращаемся к действию 3.
7. Получено искомое произведение Мн * Мт.

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

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

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

## 14. УМНОЖЕНИЕ ЧИСЕЛ С ХРАНЕНИЕМ ПЕРЕНОСОВ

Уменьшает время в n раз.
Умножение основано на том, что переносы между соседними сумматорами не выполняются (т.е. каждый последующий сумматор не ожидает переноса из предыдущего).
Это позволяет сложить n сумматоров за 1 такт, а не за n.
Переносы записываются в дополнительный регистр. Сумма сдвигается, а затем складывается вместе с переносами и очередным частичным произведением.
На последнем такте (при умножении на последний разряд множителя) сложение выполняется с учетом межразрядных переносов. Метод используется только с алгоритмом А

## 15. УМНОЖЕНИЕ НА 2 РАЗРЯДА МНОЖИТЕЛЯ ОДНОВРЕМЕННО В ПРЯМЫХ КОДАХ

***Основное требование***: за 1 такт умножения на 2 разряда может быть сделано не более 1 сложения (т.е. умножение на 1 пару должно выполняться за 1 сложение)

В случае ***пары 00*** необходимо выполнить просто сдвиг частичной суммы на два разряда – $\Sigma^{\text{ч}}_{i-1}\cdot 2^{-2}$

Для ***пары 01*** выполняется добавление множимого в сумматор со сдвигом суммы на два разряда – $(\Sigma^{\text{ч}}_{i-1} + \text{Мн})2^{-2}$.

При наличии ***пары* 10** возможны следующие варианты действий:

~~**а)**~~ в этом случае происходят ~~два сложения~~, что противоречит требованию;
$(\Sigma^{\text{ч}}_{i-1} + \text{Мн} + \text{Мн})\cdot 2^{-2}$

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

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

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

При умножении на ***пару 11*** к частичной сумме необходимо добавить утроенное множимое. Пару 11 можно представить в виде 11 = ($2^2$– 1).
Мн · 11 = Мн · ($2^2$– 1) = Мн · $2^2$– Мн, т. е. в текущем такте к частичной сумме добавляется множимое, взятое со знаком минус.
Добавление Мн · $2^2$ реализуется путем увеличения на единицу следующей старшей пары разрядов Мн.

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

$$T_{умн}^{2\ разр} = (n/2 + 1)[0{,}75 \cdot (t_{сл} + t_{сдв}) + 0{,}25 \cdot t_{сдв}],$$

***В случае переполнения в алгоритме А просто сдвинуть, не отбрасывать 1.***

## 16. УМНОЖЕНИЕ НА 4 РАЗРЯДА МНОЖИТЕЛЯ ОДНОВРЕМЕННО В ПРЯМЫХ КОДАХ

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

*Пример.* Мн = 011
Мт = *C*49
$Мт^{п} = 1\bar{4}5\bar{7}$

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

$[+7Мн]_{доп} = 0.10101$ $[+5Мн]_{доп} = 0.01111$ $[+4Мн]_{доп} = 0.01100$
$[-7Мн]_{доп} = 1.01011$ $[-4Мн]_{доп} = 1.10100$

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

## 17. УМНОЖЕНИЕ ДРОБНЫХ ЧИСЕЛ В ДОПОЛНИТЕЛЬНЫХ КОДАХ

Возможны четыре случая сочетания знаков сомножителей.

***1. Мн > 0,  Мт > 0.***
В этом случае дополнительный код сомножителей совпадает с прямым кодом и умножение выполняется по правилам умножения в прямых кодах, в результате чего получается верный результат.

***2. Мн > 0,  Мт < 0.***
Так как Мн и Мт имеют разные знаки, то результат будет иметь отрицательный знак. Следовательно, ***результат*** должен быть представлен ***в дополнительном коде***
$[Мн \cdot Мт]_{доп} = 2 - Мн \cdot Мт.$
Для формирования произведения выполним умножение Мн на Мт' (дополнение Мт):
$Мн \cdot Мт' = Мн \cdot (1 - Мт) = Мн - Мн \cdot Мт.$
Таким образом, погрешность умножения Δ равна разности
$[Мн \cdot Мт]_{доп}$ и $Мн \cdot Мт'$
$\Delta = 2 - Мн \cdot Мт - Мн + Мн \cdot Мт = 2 - Мн = [-Мн]_{доп}$
и должна быть внесена в полученный результат в качестве **поправки**.

***3. Мн < 0, Мт > 0.***
При этом сочетании знаков возможно умножение **без** ввода **поправки**.

***4. Мн < 0,  Мт < 0.***
При отрицательных сомножителях произведение $Мн \cdot Мт > 0$.

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

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

$[Мн]_{доп} \cdot Мт' = [Мн]_{доп} \cdot (1 - Мт) = [Мн]_{доп} - [Мн]_{доп} \cdot Мт =$

$[Мн]_{доп} - [Мн \cdot Мт]_{доп},$

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

## 18. УМНОЖЕНИЕ ЦЕЛЫХ ЧИСЕЛ В ДОПОЛНИТЕЛЬНЫХ КОДАХ

***1. Мн > 0, Мт > 0.***
Как отмечалось выше, умножение выполняется по правилам умножения чисел в прямых кодах.

***2. Мн > 0, Мт < 0,***

$[Мт]_{доп} = 2^n - Мт.$
Так как сомножители имеют разные знаки, то произведение $Мн \cdot Мт < 0$, следовательно,
$[Мн \cdot Мт]_{доп} = 2^{2n+1??} - Мн \cdot Мт.$
Однако при умножении $Мн \cdot [Мт]_{доп}$ получается
$Мн \cdot (2^n - Мт) = 2^n \cdot Мн - Мн \cdot Мт.$
Следовательно, погрешность в этом случае равна:

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

***3. Мн < 0, Мт > 0.***
Здесь, как и при умножении дробных чисел

***4. Мн < 0, Мт < 0.***

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

$Мн \cdot Мт = 2^{2n+1} - [Мн\ Мт]_{доп}$,

$[Мн]_{доп} = 2^{n+1} - Мн$,

$Мт' = 2^{n} - Мт$. (для целых)

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

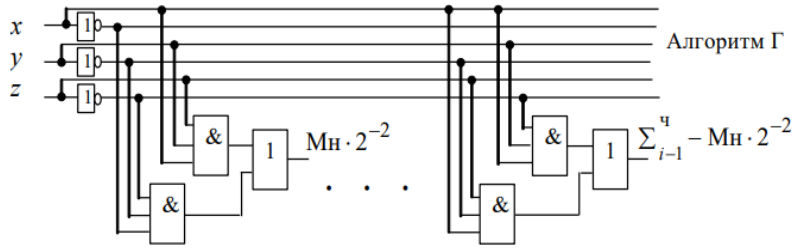
$$[Мн]_{доп} \cdot [Мт]_{доп} = [Мн]_{доп} ( 2^{n} - Мт ) = 2^{n} [Мн]_{доп} - [Мн * Мт]_{доп}$$

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

## 19. УМНОЖЕНИЕ НА 2 РАЗРЯДА МНОЖИТЕЛЯ ОДНОВРЕМЕННО В ДОПОЛНИТЕЛЬНЫХ КОДАХ

| пара | старшая | младшая | старшая | младшая |
|---|---|---|---|---|
| до преобразования | 00 | 11 | 00 | 10 |
| после преобразования | 01 | $0\overline{1}$ | 01 | $\overline{1}0$ |
| действие | +4Мн | −Мн | +4Мн | −2Мн |

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




Как было показано выше, при умножении чисел в дополнительных кодах в общем случае необходимо вводить поправку для получения верного произведения. Однако при умножении на два разряда множителя, этого выполнять не требуется ввиду того, что знак тоже участвует в преобразовании. Если на умножение поступает отрицательный множитель, то при преобразовании его старшей (знаковой) пары (1.1 или 1.0) образуются новые пары (0.1 или 1.0), а разряд, который обычно передается в старшую пару, отбрасывается. За счет этого производится поправка.

***01 допом перед знаком отбрасывается***

***преобразование вместе со знаком, без деления на целую и дробную части***

***При преобразовании Мт в ПРЯМОМ коде пара сначала увеличивается, потом преобразовывается***

***В ДОПОЛН. Коде пара преобразуется и лишь потом увеличивается***

## 20. МАТРИЧНЫЕ МЕТОДЫ УМНОЖЕНИЯ

Пусть имеем сомножители
Мн = $A = a_n \ldots a_2\ a_1$
Мт = $B = b_n \ldots b_2\ b_1$

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

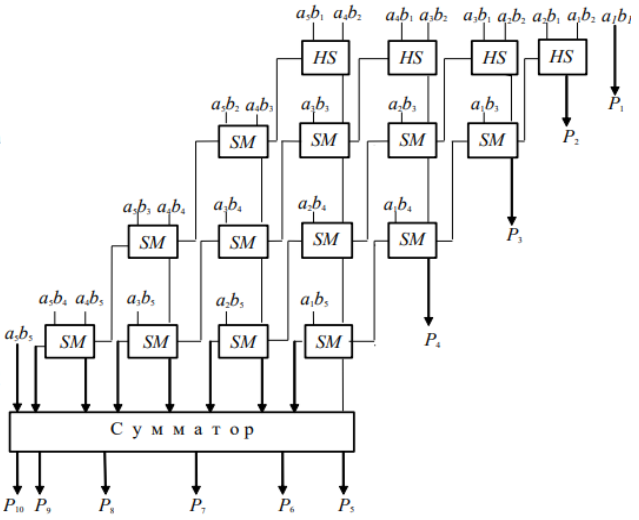
Рассмотрим схему умножения чисел согласно алгоритму Б. Данная схема умножения может быть представлена в виде матрицы (табл.3).

Таблица 3

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

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

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




Полусумматор отличается от сумматора тем, что не имеет входного переноса, т.е. он не учитывает перенос из предыдущего.

## 21. МАШИННЫЕ МЕТОДЫ ДЕЛЕНИЯ. ДЕЛЕНИЕ В <u>ПРЯМЫХ</u> КОДАХ

Деление – простое многократное вычитание делителя вначале из делимого, затем из остатков.

При делении в случае ***переполнения*** следует для чисел с фиксированной запятой процесс остановить, с плавающей запятой продолжить до конца, а потом, после получения последней n-й цифры частного, число сдвинуть вправо на один разряд с добавлением единицы к порядку, равному разности порядков делимого и делителя.

**<u>Алгоритм деления с восстановлением остатка</u> состоит в следующем.**

1. Выполняется ***пробное вычитание*** с формированием первого остатка $\mathbf{A_1 = [Дм]_{доп} + [-Дт]_{доп}}$.

Далее, если $\mathbf{A_1 < 0}$, то в разряд слева от запятой, заносится нуль (**0**),
иначе − единица (**1**) – что является признаком переполнения и деление завершается.

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

3. Формирование ***очередного остатка***. $A_{i+1} = A_i \cdot 2 + [-Дт]_{доп}$ (***сдвиг, -Дт***)
Если $\mathbf{A_{i+1} < 0}$,то в очередной разряд частного справа от запятой записывается ***нуль*** (0), ***иначе*** − ***единица*** (1) (***просто инверсия числа перед запятой***)

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

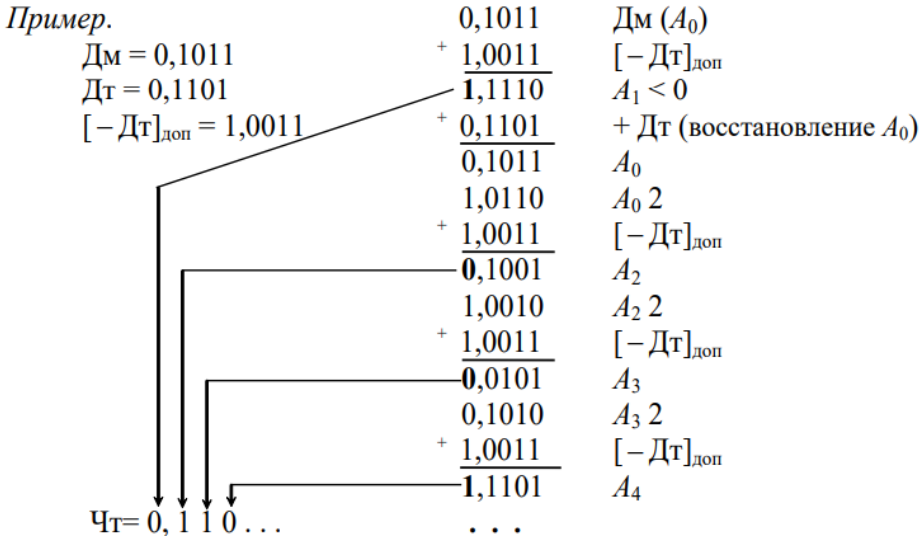
1) необходимо затрачивать ***дополнительно время*** на восстановление остатка;

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

В случае Ai < 0 для получения остатка Ai+1 необходимо выполнить

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

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




**<u>Алгоритм деления без восстановления остатка</u>.**

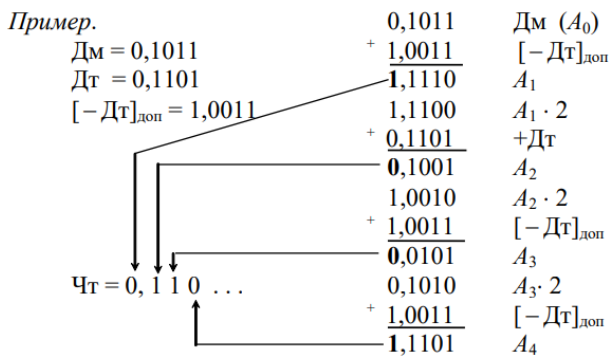
1. Выполняется ***пробное вычитание*** с формированием первого остатка $A_1 = [Дм]_{доп} + [-Дт]_{доп}$.

Далее, если ***$A_1 < 0$***, то в разряд слева от запятой заносится ***нуль*** (0), ***иначе – единица*** (1) – что является признаком переполнения и деление прекращается.

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

3. Если ***$A_{i+1} < 0$***, то в очередной разряд частного справа от запятой записывается ***нуль*** (0), ***иначе*** записывается ***единица*** (1). (***просто инверсия числа перед запятой***)

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

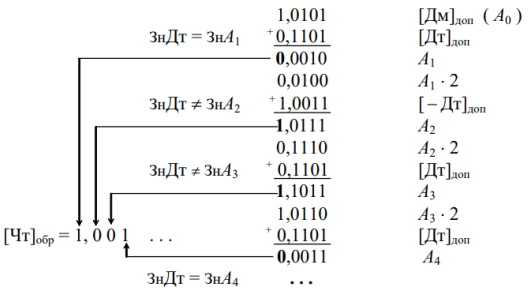



***Различие между алгоритмами:***
С ВОССТ при A < 0 сначала +Дт, а потом сдвиг
БЕЗ ВОССТ сначала сдвиг, а потом только+-Дт

## 22.МАШИННЫЕ МЕТОДЫ ДЕЛЕНИЯ.
## ДЕЛЕНИЕ В ДОПОЛНИТЕЛЬНЫХ КОДАХ

При делении чисел ***знаковая и значащая части*** частного формируются ***раздельно***. Знак частного формируется согласно формуле ***Знак Чт*** = Знак Дм ⊕ Знак Дт.
***Основывается на делении без восстановления остатка***. В отличие от деления в прямых кодах, здесь сравнивается знак ***делимого (остатка)*** со ***знаком делителя***.
Частное формируется в ***обратном коде.***

***Знак*** *А* ***равен*** *Дт, то* ***-Дт, 1***  ***Знак*** *А* ***не равен*** *Дт, то* ***+Дт, 0***




**Алгоритм деления чисел в дополнительных кодах**.

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

2. Формирование очередного остатка. Если Знак Ai ***не равен*** Знаку Дт, то $A_{i+1} = A_i \cdot 2$ **+ [Дт]доп,** ***иначе*** $A_{i+1} = A_i \cdot 2$ + **[−Дт]доп.**

3. Если Знак Ai + 1 ***не равен*** Знаку Дт, то в очередной разряд частного справа от запятой заносится ***нуль*** (0), ***иначе*** − ***единица*** (1).

4. Если достигнута заданная точность частого или получен нулевой остаток Ai + 1, то процесс деления завершается, иначе возвращаемся к п. 2 алгоритма.

## 23. СТРУКТУРНАЯ СХЕМА ОПЕРАЦИОННОГО УСТРОЙСТВА ВЫПОЛНЯЮЩЕГО ОПЕРАЦИЮ ДЕЛЕНИЯ

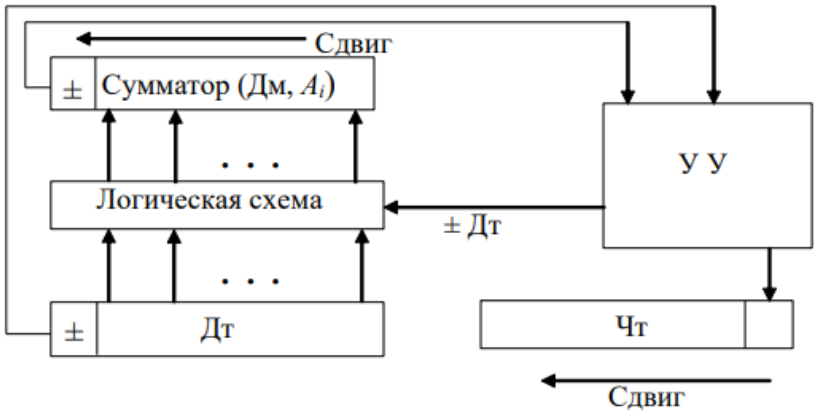



Рис. 12. Структурная схема устройства деления

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

## 24. МЕТОДЫ УСКОРЕНИЯ ДЕЛЕНИЯ

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

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

***Реализация*** этого метода ускорения кроме устройства управления делением ***требует логическую схему, осуществляющую две функции:***
1) ***сдвиг*** модулей *делителя* и *делимого* до тех пор, *пока у модуля делителя после запятой не останется ни одного нуля;*
2) ***выявление остатков*** вида 0,0…01 или 1,1…10.
***Степень сложности*** логической схемы ***определяется количеством разрядов***, ***участвующих в*** косвенном ***сравнении*** модулей делителя и делимого (остатков).

Обычно реализация логических методов ускорения при делении 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… .

## 25. ОДНОРАЗРЯДНЫЙ ДВОИЧНО-ДЕСЯТИЧНЫЙ СУММАТОР

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

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

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

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

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

Если каждый разряд числа имеет вес и при r ≠ 2 k 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 выражения (4), то будем говорить, что число представлено в ***двоично-десятичном коде***. Наибольшее распространение получили двоично-десятичные коды, в которых ***десятичная цифра представляется двоичной тетрадой*** – BCD-коды (***Binary coded decimal***). Существует множество способов кодирования десятичных цифр.

**Набор требований**, позволяющих упростить выполнение арифметических операций и операций перевода чисел:
− ***четность*** − четным десятичным цифрам соответствуют только четные двоичные коды и наоборот. Это обеспечивает эффективность операций округления, умножения и деления чисел в BCD-кодах;
− ***дополняемость*** − сумма двоичного кода и инверсного ему кода любой десятичной цифры должна быть равна 9. Это обеспечивает эффективность операции алгебраического сложения в BCD-кодах;
− ***упорядоченность***, т. е. большей десятичной цифре соответствует большая тетрада и наоборот;
− ***единственность представления десятичной цифры двоичной тетрадой***;
− ***взвешенность***, т. е. каждому разряду двоичного представления десятичной цифры поставлен в соответствие вес. Это обеспечивает эффективность всех арифметических и логических операций в BCD-кодах. Если каждая десятичная цифра кодируется соответствующим двоичным эквивалентом, то такое кодирование называется *кодом прямого замещения*.

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

Основной ***недостаток*** этого кода заключается в том, что инверсия какойлибо цифры оказывается цифрой, дополняющей данную цифру до 15, а не до 9

***Правила потетрадного сложения чисел в BCD-кодах***:

1) Если при потетрадном сложении ***перенос*** в соседнюю старшую тетраду ***не возникает***, то результат суммирования ***не требует коррекции***.
2) ***Коррекция*** результата потетрадного сложения путем добавления поправки **0110** ***требуется*** в случае, если возникает:

**а**) потетрадный перенос в старшую тетраду;
**б**) запрещенная комбинация.

Устройство, которое работает по сформулированным выше правилам, называется ***одноразрядным двоично-десятичным сумматором***

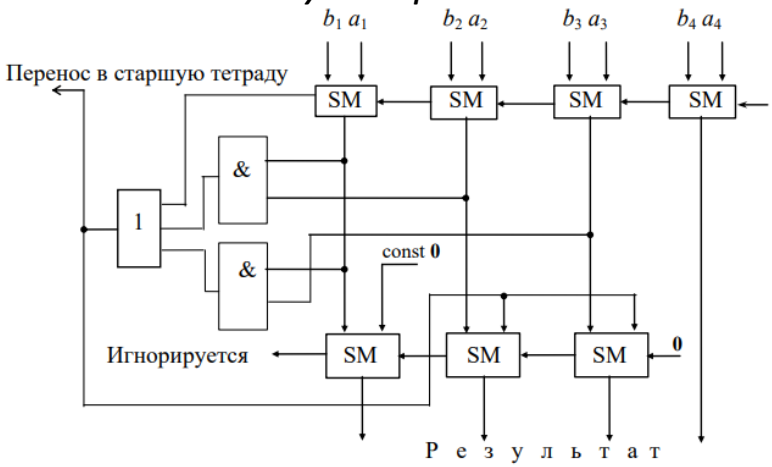


Рис. 14. Структурная схема одноразрядного двоично-десятичного сумматора

## 26. BCD-КОДЫ. СУММИРОВАНИЕ ЧИСЕЛ С ОДИНАКОВЫМИ ЗНАКАМИ

При ***сложении чисел в BCD-коде*** возможны три случая:
**1**. ( a + b ) **≤ 9** . Величина итоговой суммы не превышает девяти и коррекция результата не требуется.
**2. 10 ≤** ( a + b ) **≤** 15. Тетрада результата будет содержать единицы в разрядах с весом 8 и 4 / 8 и 2 – назовем это ***запрещенными комбинациями***.
В тетраде накопилось более девяти единиц и должен быть выполнен ***десятичный перенос***. Перенос единицы в старший разряд выполняется принудительно с помощью логической схемы. Условием для формирования единицы переноса является *возникновение запрещенной комбинации*. Однако тетраду надо освободить от десяти избыточных единиц. Это делается принудительно ***добавлением*** 0110 (***шестерки***), что приводит к ***возникновению шестнадцатеричного переноса,*** который вместе с добавленной шестеркой ***унесет и лишний десяток из тетрады***. Этот перенос игнорируется

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

$$\begin{array}{rr} A = 169 & A = 0.0001\ 0110\ 1001 \\ +\ \underline{B = 378} & +\ \underline{B = 0.0011\ 0111\ 1000} \\ A + B = 547 & A + B = 0.0101\ 1110\ 0001 \\ & \underline{\qquad\qquad 0110\ 0110} \\ & 0.0101\ 0100\ 0111 \end{array}$$

перенос игнорируется

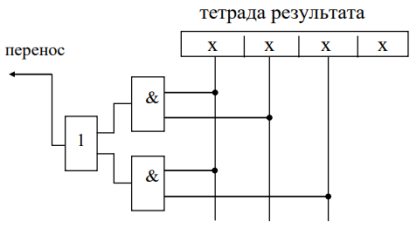


Рис. 13. Схема определения запрещенной комбинации

**3**. ( a + b ) **≥ 16**. В процессе суммирования возникает шестнадцатеричный перенос, в результате которого ***тетраду покидают 16 единиц вместо 10***.
Чтобы восстановить верное значение этой тетрады, необходимо к ней ***добавить*** 0110 (***шесть***).

## 27. BCD-КОДЫ. СЛОЖЕНИЕ ЧИСЕЛ С РАЗНЫМИ ЗНАКАМИ (РЕЗУЛЬТАТ >= 0)

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

| | | |
|---|---|---|
| $a = 7$ | $0.0111$ | $[a]_{обр}$ |
| $b = -3$ | $+1.1100$ | $[b]_{обр}$ |
| $4$ | $10.0011$ | |
| | $\rightarrow 1$ | |
| | $0.0100$ | |

При образовании инверсии отрицательной тетрады в нее добавляются 15 единиц. Эти 15 единиц находятся и в сумме. Но благодаря шестнадцатеричному переносу из тетрады суммы уходит 16 единиц (***1 единица восстанавливается добавлением по цепи циклического переноса***).

## 28. BCD-КОДЫ. СЛОЖЕНИЕ ЧИСЕЛ С РАЗНЫМИ ЗНАКАМИ (РЕЗУЛЬТАТ < 0)

| | | |
|---|---|---|
| $a = 3$ | $0.0011$ | $[a]_{обр}$ |
| $b = -7$ | $+1.1000$ | $[b]_{обр}$ |
| $-4$ | $1.1011$ | |
| | $0.0100$ | |

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

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

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

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

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

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

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

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

Таким образом, в тетраду **производится заем**, если результат:
− ***положительный*** и из тетрады ***нет переноса***; 1010
− ***отрицательный*** и из тетрады ***есть перенос***. 0110
В случае заема в тетраду ***необходима коррекция***

## 29. BCD-КОДЫ С ИЗБЫТКОМ 3

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

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

2) 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 BCD-коде.

***Правило.*** Если из тетрады ***был перенос***, надо ***+3,*** если ***переноса не было*** − ***-3*** или ***+12***, независимо от знака слагаемых и знака суммы.

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

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

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

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

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

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

# ЛОГИЧЕСКИЕ ОСНОВЫ

## 30. ОСНОВНЫЕ ПОНЯТИЯ АЛГЕБРЫ ЛОГИКИ.

***Алгебра логики*** (булева алгебра) – это *математический аппарат*, с помощью которого записывают (кодируют), упрощают, вычисляют и преобразовывают *логические высказывания*.
Алгебра множества всевозможных булевых функций определённых на 0 и 1.
***Логическое высказывание*** – это любое *утверждение*, в отношении которого можно однозначно сказать *истинно* оно или *ложно*.
*Основными понятиями* алгебры логики являются ***двоичные переменные*** и ***логические функции***.

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

***Булева функция f***, зависящая от n переменных $x_1$, $x_2$, … , $x_n$, называется *булевой*, или *переключательной*, если функция *f* и любая из ее переменных $x_i$, (i = 1, ..., n) *принимают значения только из множества {0, 1}.*
Это функция ***аргументы и значение*** которой ***принадлежат множеству {0, 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.***
Булевы функции могут служить аргументами *более сложных* булевых функций.

## 31. СПОСОБЫ ЗАДАНИЯ ФУНКЦИЙ АЛГЕБРЫ ЛОГИКИ.

Существует ***ряд способов задания*** булевых функций:
- ***словесный***
- ***аналитический*** (с помощью описания ф-ций или системы ф-ций)
- ***табличный*** (матричный) – задаётся таблицей истинности. Слева – всевозможные двоичные наборы длины n, а справа – значения функции на этих наборах. Количество наборов - $2^n$.
- ***схемный***

| $x_1x_2x_3$ | $f$ |
|---|---|
| 000 | 0 |
| 001 | 1 |
| 010 | 0 |
| 011 | 0 |
| 100 | 1 |
| 101 | 1 |
| 110 | 0 |
| 111 | 1 |

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

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

***Логический элемент*** в электронных схемах – это устройство, реализующее некоторую операцию алгебры логики. При этом логические сигналы 0 и 1 задаются разными уровнями напряжения. Обычно сигнал логического ***нуля*** задается ***низким*** *уровнем напряжения*, а ***единица*** – ***высоким***.
Функций от одной переменной четыре:
-это константа 0 ($f_0$),
-константа 1 ($f_1$),
-тождественная функция ($f_2$)
-функция отрицания ($f_3$)

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

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

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

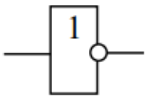
для любых значений $x_1, \ldots, x_{i-1}, x_{i+1}, \cdots, x_n$. Иначе переменная $x_i$ называется *существенной*.

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

Отметим наиболее часто используемые функции из числа приведенных в таблице:

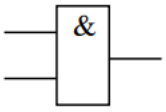
- $f_0(x_1, x_2) = 0$ – тождественный ноль (константа 0);
- $f_1(x_1, x_2) = x_1 \cdot x_2$ – конъюнкция (логическое произведение, И); обозначается знаком &, /\ или · (символы операции конъюнкции в логических выражениях можно опускать, например, $x_1x_2$);
- $f_3(x_1, x_2) = x_1$ – повторение $x_1$;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

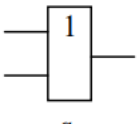
Основными операциями булевой алгебры являются: отрицание, логическое сложение и логическое умножение




***Логическое отрицание*** (функция НЕ). Сложное высказывание f(x), которое истинно, когда x ложно, и наоборот. Функция НЕ записывается следующим образом: $f = \overline{x}$.

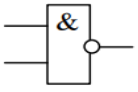
***Логическое умножение*** (конъюнкция). ***Конъюнкция*** (функция ***И***) – это сложное высказывание, которое истинно только тогда, когда **1 И 1**, и ложно для всех остальных наборов переменных. Функция конъюнкции имеет вид f = x1x2. Для обозначения используются также символы & и Λ.

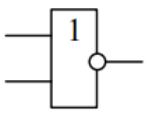



***Логическое сложение*** (дизъюнкция). ***Дизъюнкция*** (функция ***ИЛИ***)– это сложное высказывание, которое истинно тогда, когда истинна хотя бы одна из переменных x1 и x2, и ложно, когда **0 ИЛИ 0.** Функция дизъюнкции имеет вид f = x1 v x2. Для обозначения используется также символ v.

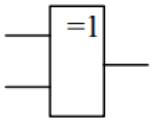
***Отрицание конъюнкции*** (операция ***штрих Шеффера***). ***Отрицание конъюнкции*** (функция ***И-НЕ***) – сложное высказывание, ложное только при 1 и 1. Булева функция И-НЕ имеет вид $f = \overline{x_1 \, x_2}$.




**Отрицание дизъюнкции** (операция ***«стрелка Пирса»,*** функция ***Вебба***). Отрицание дизъюнкции (функция ***ИЛИ-НЕ***)– сложное высказывание, истинное только тогда, когда обе переменные 0 и 0. Булева функция ИЛИ-НЕ имеет вид $f = \overline{x_1 \vee x_2}$.

***Сложение по модулю 2*** – это сложное высказывание, которое ложно тогда, когда истинна только одна из переменных x1 и x2. Функция «сумма по модулю 2» имеет вид ***f* = x1⊕ x2**. Если число переменных n > 2, то функция истинна на тех наборах, в которых число единиц нечетно




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

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

## 32. ФОРМЫ ПРЕДСТАВЛЕНИЯ ФУНКЦИЙ АЛГЕБРЫ ЛОГИКИ

Основными понятиями, лежащими в основе представления булевых функций в различных формах, являются понятия элементарной конъюнкции и элементарной дизъюнкции.
***Элементарной конъюнкцией*** называется *логическое* ***произведение*** любого конечного числа различных между собой булевых *переменных, взятых со знаком инверсии или без него.*
***Элементарной дизъюнкцией*** называется *логическая* ***сумма*** любого конечного числа различных между собой булевых *переменных, взятых со знаком инверсии или без него*

Дизъюнктивная нормальная форма (***ДНФ***) – ***дизъюнкция*** конечного числа элементарных ***конъюнкций***

$f_{\text{ДНФ}} = x_1\bar{x}_2\bar{x}_3 \vee x_1x_4 \vee x_2x_3\bar{x}_4 \vee x_1\bar{x}_2x_3$ ***сумма произведений***

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

Совершенная ДНФ (***СДНФ***) – ДНФ, в которой все *конъюнкции* имеют *ранг n*. СДНФ записывается по таблице истинности согласно правилу: для каждого набора переменных, на котором булева функция принимает ***единичное значение***, записывается конъюнкция ранга n и все эти конъюнкции объединяются знаками дизъюнкции; переменная имеет знак инверсии, если на соответствующем наборе имеет нулевое значение:

$f_{\text{СДНФ}} = x_1x_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1x_2x_3$

Элементарные конъюнкции, образующие **СДНФ**, называют ***конституентами*** (составляющими) ***единиц***, так как они соответствуют наборам, при которых функция принимает значение, равное единице.

Конъюнктивная нормальная форма (***КНФ***) ***конъюнкция*** конечного числа элементарных ***дизъюнкций***:

$f_{\text{КНФ}} = (x_1 \vee \bar{x}_2 \vee \bar{x}_3)(\bar{x}_1 \vee \bar{x}_4)(x_2 \vee \bar{x}_3 \vee x_4)(x_1 \vee x_2 \vee x_3)$. ***произведение сумм***

Совершенная КНФ (СКНФ) – КНФ, в которой все *дизъюнкции* имеют *ранг n*. СКНФ записывается по таблице истинности согласно правилу: для каждого набора переменных, на котором булева функция принимает ***нулевое значение***, записывается дизъюнкция ранга n и все эти дизъюнкции объединяются знаками конъюнкции; переменная имеет знак *инверсии, если* на соответствующем наборе имеет *единичное значение*

$f_{\text{СКНФ}} = (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)(x_1 \vee \bar{x}_2 \vee \bar{x}_3)(\bar{x}_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee x_3)$

Элементарные дизъюнкции, образующие **СКНФ**, называют ***конституентами*** (составляющими) ***нуля***, так как они соответствуют наборам, при которых функция принимает нулевое значение.

## 33. ОСНОВНЫЕ ЗАКОНЫ И ПРАВИЛА АЛГЕБРЫ ЛОГИКИ.

Предназначены для приведения (преобразования) функции к требуемому (удобному) виду.

$$x \vee 1 = 1; \quad x \vee \overline{x} = 1; \quad \overline{0} = 1;$$

$$x \cdot 0 = 0; \quad x \cdot \overline{x} = 0; \quad \overline{1} = 0;$$

$$\overline{\overline{x}} = x;$$

***Закон рефлексивности:***

− для дизъюнкции: x v x = x;

− для конъюнкции: x · x = x.

$$x \vee x \vee \ldots \vee x = x;$$

$$x \cdot x \cdot \ldots \cdot x = x.$$

***Переместительный закон:***

− для дизъюнкции: $x_1$ v $x_2$ = $x_2$ v $x_1$;

− для конъюнкции: $x_1 \cdot x_2 = x_2 \cdot x_1$;

− для сложения по модулю два: $x_1(+)x_2 = x_2(+)x_1$.

***Сочетательный закон:***

− для дизъюнкции: $x_1$ v($x_2$ v $x_3$) = ($x_1$ v $x_2$)v $x_3$;

− для конъюнкции: $x_1 \cdot (x_2 \cdot x_3) = (x_1 \cdot x_2) \cdot x_3$;

− для суммы по модулю два: $x_1(+)(x_2(+)x_3) = (x_1(+)x_2)(+)x_3$,

***Распределительный закон:***

− для дизъюнкции: $x_1$ v $x_2 \cdot x_3 = (x_1 \vee x_2)(x_1 \vee x_3)$,

т. е. ИЛИ переменной и И эквивалентна И из двух ИЛИ этой переменной с сомножителями;

− для конъюнкции: $x_1 \cdot (x_2 \vee x_3) = x_1 \cdot x_2 \vee x_1 \cdot x_3$,

т. е. И переменной и ИЛИ равносильна ИЛИ из двух И этой переменной со слагаемыми.

*Закон инверсии (правило де Моргана):*

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

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

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

Следующие соотношения могут быть легко выведены из рассмотренных выше.

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

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

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

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

## 34. КЛАССЫ ФУНКЦИЙ АЛГЕБРЫ ЛОГИКИ. ФУНКЦИОНАЛЬНО ПОЛНЫЕ НАБОРЫ

***Функционально полный набор*** – это такой набор логических операций (реализующих их логических элементов), позволяющих записать любую ФАЛ, построить любую цифровую схему.

*Класс линейных функций*. Булева функция называется линейной, если ее можно представить полиномом первой степени:

$$f(x_1 x_2 \ldots x_n) = k_0 \oplus k_1 x_1 \oplus k_2 x_2 \oplus \ldots \oplus k_n x_n.$$

*Класс функций, сохраняющих нуль*. К булевым функциям, сохраняющим константу 0, относят такие булевы функции $f(x_1, ..., x_n)$, для которых справедливо соотношение $f(0, ..., 0) = 0$.

*Класс функций, сохраняющих единицу.* К булевым функциям, сохраняющим константу 1, относят такие булевы функции $f(x_1, ..., x_n)$, для которых справедливо соотношение $f(1, ..., 1) = 1$.

*Класс монотонных функций*. Булева функция называется монотонной, если при любом возрастании набора переменных значения этой функции не убывают. Двоичный набор $A = (a_1, a_2, ..., a_n)$ не меньше двоичного набора $B = (b_1, b_2, ..., b_n)$, если для каждой пары $(a_i, b_i)$ $i = 1...n$ справедливо соотношение $a_i \geq b_i$.

*Класс самодвойственных функций*. Булевы функции $f_1(x_1, x_2, ..., x_n)$ и $f_2(x_1, x_2, ..., x_n)$ называются двойственными друг другу, если выполняется соотношение $f_1(x_1, x_2, ..., x_n) = \overline{f_2}(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n)$.

Теорема Поста − Яблонского. Для того чтобы система булевых функций была полной, необходимо и достаточно, чтобы она содержала хотя бы одну функцию:
− не являющуюся линейной;
− не сохраняющую нуль;
− не сохраняющую единицу;
− не являющуюся монотонной;
− не являющуюся самодвойственной.
Иначе говоря, система булевых функций является функционально полной тогда и только тогда, когда она целиком не содержится ни в одном из предполных классов.

| Функция | Вид функции | | | | |
|---|---|---|---|---|---|
| | немонотонные | нелинейные | несамодвойственные | не сохраняющие 0 | не сохраняющие 1 |
| $F_0 \equiv 0$ | – | – | + | – | + |
| $F_1 = x_1 \cdot x_2$ | – | + | + | – | – |
| $F_3 = x_1$ | – | – | – | – | – |
| $F_5 = x_2$ | – | – | – | – | – |
| $F_6 = x_1 \oplus x_2$ | + | – | + | – | + |
| $F_7 = x_1 \vee x_2$ | – | + | + | – | – |
| $F_8 = x_1 \downarrow x_2$ | + | + | + | + | + |
| $F_9 = x_1 \sim x_2$ | + | – | + | + | – |
| $F_{13} = x_1 \rightarrow x_2$ | + | + | + | – | + |
| $F_{14} = x_1 / x_2$ | + | + | + | + | + |
| $F_{15} \equiv 1$ | – | – | + | + | – |

Каждая из F8 и F14 является ФПСБФ. Используя только булеву функцию F14 – «штрих Шеффера», можно записать в виде формулы любую булеву функцию. Признаком функциональной полноты является наличие плюса в каждом столбце таблицы хотя бы для одной из составляющих систему булевых функций.

К таким ФСПБФ, наиболее распространенным в практике построения цифровых автоматов, следует отнести

$$\{\wedge, \vee, \text{не}\},\ \{\wedge, \oplus, \text{не}\},\ \{\wedge, \oplus, 1\},\ \{\wedge, \text{не}\},\ \{\vee, \text{не}\}$$

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

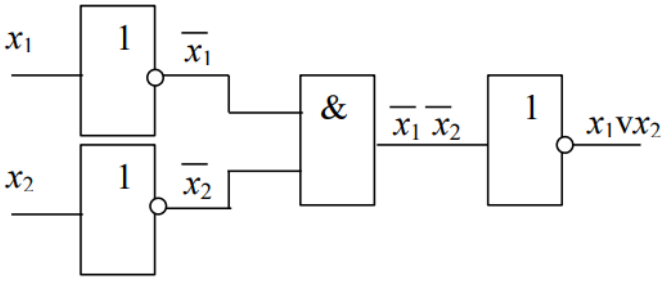



Рис. 21. Реализация операции ИЛИ в базисе И, НЕ

## 35. МИНИМИЗАЦИИ БУЛЕВЫХ ФУНКЦИЙ МЕТОДОМ КВАЙНА

**Теорема Квайна**. Для получения минимальной формы булевой функции необходимо в СДНФ произвести все возможные ***склеивания*** и ***поглощения*** так, чтобы в результате была получена сокращенная ДНФ.

1) *операция неполного склеивания* $Fx \vee F\bar{x} = Fx \vee F\bar{x} \vee F$, где $Fx$ и $F\bar{x}$ – две конъюнкции, а $F$ – конъюнкция, полученная в результате их склеивания (обычного) по $x$;

2) *операция поглощения* $F \vee F\tilde{x} = F$.

Пример минимизации БФ заданной в форме СДНФ.

$$f_{\text{СДНФ}} = \underset{1}{\bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4} \vee \underset{2}{\bar{x}_1x_2\bar{x}_3\bar{x}_4} \vee \underset{3}{x_1\bar{x}_2\bar{x}_3\bar{x}_4} \vee \underset{4}{\bar{x}_1\bar{x}_2\bar{x}_3x_4} \vee \underset{5}{\bar{x}_1x_2\bar{x}_3x_4} \vee \underset{6}{x_1x_2\bar{x}_3x_4} \vee \underset{7}{x_1x_2x_3\bar{x}_4} \vee$$

$$\vee \underset{8}{x_1\bar{x}_2x_3\bar{x}_4} \vee \underset{9}{\bar{x}_1\bar{x}_2x_3x_4} \vee \underset{10}{\bar{x}_1x_2x_3x_4} \vee \underset{11}{x_1\bar{x}_2x_3x_4}$$

Вначале выполняются всевозможные склеивания (и поглощения) для получения сокращенной ДНФ (ДНФ состоящей из простых импликант).

| 1 этап | | | 2 этап | | |
|---|---|---|---|---|---|
| 1 – 2 (по $x_2$) | $\bar{x}_1\bar{x}_3\bar{x}_4$ | 1 | | | |
| 1 – 3 (по $x_1$) | $\bar{x}_2\bar{x}_3\bar{x}_4$ | 2 | 1 – 6 (по $x_4$) | $\bar{x}_1\bar{x}_3$ | 1 |
| 1 – 4 (по $x_4$) | $\bar{x}_1x_2\bar{x}_3$ | 3 | 3 – 4 (по $x_2$) | $\bar{x}_1x_3$ | 2 |
| 2 – 5 (по $x_4$) | $x_1\bar{x}_2\bar{x}_3$ | 4 | 6 – 10 (по $x_3$) | $\bar{x}_1x_4$ | 3 |
| 3 – 8 (по $x_3$) | $x_1x_2\bar{x}_4$ | 5 | 7 – 9 (по $x_2$) | $\bar{x}_1x_4$ | 4 |
| 4 – 5 (по $x_2$) | $\bar{x}_1\bar{x}_3x_4$ | 6 | | | |
| 4 – 9 (по $x_3$) | $x_1\bar{x}_2x_4$ | 7 | | | |
| 5 – 6 (по $x_1$) | $\bar{x}_2x_3x_4$ | 8 | | | |
| 5 – 10 (по $x_3$) | $x_1\bar{x}_2x_4$ | 9 | | | |
| 7 – 8 (по $x_2$) | $x_1\bar{x}_3x_4$ | 10 | | | |
| 8 – 11 (по $x_4$) | $x_1x_2\bar{x}_3$ | 11 | | | |
| 9 – 10 (по $x_2$) | $\bar{x}_1x_3x_4$ | 12 | | | |
| 9 – 11 (по $x_1$) | $\bar{x}_2x_3x_4$ | 13 | | | |

В выделенных красным цветом строках импликанты, полученные на первом этапе, не склеиваются (не порождают новых) следовательно они – простые импликанты.

На втором этапе (шаге) ни одна импликанта не породила новой. Следовательно, обе – простые импликанты.

Из полученных выше простых импликант запишем сокращенную ДНФ:

$$f_{\text{сокрДНФ}} = \bar{x}_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2\bar{x}_4 \vee x_2\bar{x}_3x_4 \vee x_1x_3\bar{x}_4 \vee x_1\bar{x}_2x_3 \vee \bar{x}_2x_3x_4 \vee \bar{x}_1\bar{x}_3 \vee \bar{x}_1x_4$$

Используя импликантную таблицу исключим из нее «лишние» простые импликанты

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\bar{x}_2\bar{x}_3\bar{x}_4$ | v | | v | | | | | | | | | 1 ТФ |
| $x_1\bar{x}_2\bar{x}_4$ | | | v | | | | | v | | | | 2 ТФ |
| $x_2\bar{x}_3x_4$ | | | | | v | v | | | | | | обязат. простая импликанта |
| $x_1x_3\bar{x}_4$ | | | | | | | v | v | | | | обязат. простая импликанта |
| $x_1\bar{x}_2x_3$ | | | | | | | | v | | | v | 1 ТФ |
| $\bar{x}_2x_3x_4$ | | | | | | | | | v | | v | 2 ТФ |
| $\bar{x}_1\bar{x}_3$ | v | v | | v | v | | | | | | | обязат. простая импликанта |
| $\bar{x}_1x_4$ | | | | v | v | | | | v | v | | обязат. простая импликанта |

$$f_{\text{minДНФ}} = x_2\bar{x}_3x_4 \vee x_1x_3\bar{x}_4 \vee \bar{x}_1\bar{x}_3 \vee \bar{x}_1x_4 \vee \bar{x}_2\bar{x}_3\bar{x}_4 \vee x_1\bar{x}_2x_3 \quad \text{- 1 ТФ (первая тупиковая форма)}$$

$$f_{\text{minДНФ}} = x_2\bar{x}_3x_4 \vee x_1x_3\bar{x}_4 \vee \bar{x}_1\bar{x}_3 \vee \bar{x}_1x_4 \vee x_1\bar{x}_2\bar{x}_4 \vee \bar{x}_2x_3x_4 \quad \text{- 2 ТФ (первая тупиковая форма)}$$

На 1 этапе выполняются всевозможные склеивания импликант в исходной СДНФ.

В результате полученные склеенные простые импликанты составляют сокращённую ДНФ, которая может содержать «мнимые» простые импликанты, которые дублируются другими простыми импликантами. Если набор в склеивании не участвовал, то сразу его в сокращённую ДНФ.

На 2 этапе путём удаления «мнимых» простых импликант формируем минимальную ДНФ.

Далее составляется таблица, число строк которой равно числу найденных простых импликант, а число столбцов – числу членов СДНФ данной функции. Если в член функции входит первичная импликанта, то на пересечении их ставится метка (смотреть что с чем склеивали)

Если в столбце таблицы имеется только одна метка, то первичная импликанта, стоящая в соответствующей строке, является *существенной*.

В конце выбор минимального покрытия. Предпочтение отдается варианту покрытия с минимальным числом букв в первичных импликантах, образующих покрытие.

***Простая импликанта*** – такой набор переменных, который не может быть склеен ни с каким другим.

## 36. КАРТЫ ВЕЙЧА. СОСЕДНИЕ НАБОРЫ, ПРОСТАЯ ИМПЛИКАНТА, МИНИМАЛЬНОЕ ПОКРЫТИЕ.

***Карта Вейча (***Минимизирующая карта***)*** представляет собой ***развертку*** n мерного ***куба*** на плоскости.
***Соседние наборы –*** 2 набора, у которых кодовое расстояние ==1
***Кодовое расстояние*** – кол-во разрядов, отличающихся инверсией (x1x2x3 и x1 не x2x3)
Размерность куба определяется кол-вом переменных, от которых зависит функция.

Правила формирования минимальной функции:

1. Карта от n переменных имеет $2^n$ клеток

Минимизация состоит в выделении на карте контуров по следующим правилам:

2. В контур объединяются только количество клеток, равное степени двойки
Если клетка объединена сама с собой, значит набор ни с чем не склеивается

***Соседние наборы*** определяются по инверсиям, т.е. если только одна переменная отличается, то оно

3. Объединяются только клетки с соседними наборами.
Например крайние строки карты (верхние и нижние, левые и правые), а также угловые клетки считаются соседними.

4. В контур объединяются клетки, содержащие только 1 или только 0
на 1 – ДНФ
на 0 – КНФ

5. Все единицы (нули) должны быть «оконтурены», т.е. объединены в контуры

6. Кол-во клеток в контуре должно быть максимальным, обеспечит максимальное склеивание

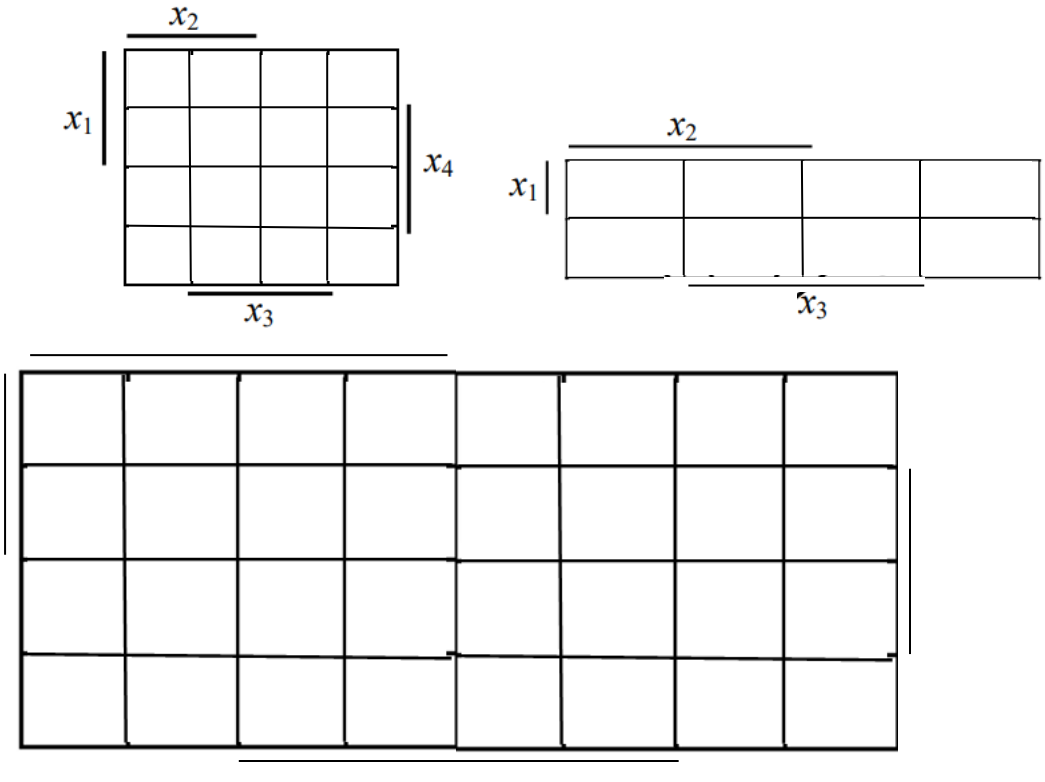
7. Количество контуров должно быть минимальным. Меньше контуров – меньше слагаемых

8. Любая клетка может входить в контуры произвольное количество раз.

9. Контур, содержащий $2^n$ клеток, имеет n осей симметрии (столько переменных убрать можно)

***Целью минимизации*** является нахождение кратчайшей из множества тупиковых форм.

Каждый контур – простая импликанта.
Мин.КНФ записывается на контурах нулевых наборов с инверсией тех переменных, которые соответствуют этим наборам (контуру).

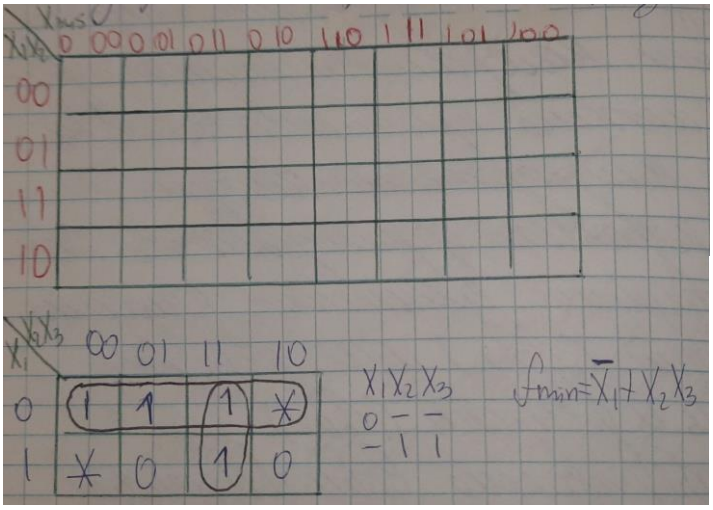



x1 слева, x2 вверху, x4 справа, x3 внизу длинная, x5 отрывистые внизу
Если мы управляем нулями, а хотим единицами, то ДНФ на нулевых наборах можно сделать двойной инверсией КНФ

## 37. КАРТЫ КАРНО. СОСЕДНИЕ НАБОРЫ, ПРОСТАЯ ИМПЛИКАНТА МИНИМАЛЬНОЕ ПОКРЫТИЕ.

Карта Карно является ***координатным способом*** представления булевых функций. Она представляет собой развертку на плоскости n-мерного единичного куба, т.е. клетки карты соответствуют координатам куба.

Переменные функции разбиваются на две группы так, что одна группа определяет координаты столбца карты, а другая − координаты строки. При таком способе построения каждая клетка определяется значениями переменных, соответствующих определенному двоичному набору. Внутри каждой клетки карты Карно ставится значение функции на данном наборе. Переменные в строках и столбцах располагаются так, чтобы соседние клетки карты Карно *различались только в одном разряде переменных*, т. е. были соседними.
***5***: двоичный код, в котором рядом стоящие наборы – соседние
(2 набора, стоящие рядом – соседние.)




| $x_1x_2$ \ $x_3x_4$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0000 | 0001 | 0011 | 0010 |
| 01 | 0100 | 0101 | 0111 | 0110 |
| 11 | 1100 | 1101 | 1111 | 1110 |
| 10 | 1000 | 1001 | 1011 | 1010 |

## 38. МИНИМИЗАЦИЯ НЕ ПОЛНОСТЬЮ ОПРЕДЕЛЕННЫХ ПФ В ДИЗЪЮНКТИВНОЙ И КОНЪЮНКТИВНОЙ ФОРМАХ.

**При работе цифрового устройства могут быть выделены наборы, которые могут поступать на вход ЦУ и формировать на выходе схемы (1 или 0). Но вместе с ними могут существовать наборы, которые не существуют (т.е. не могут прийти на вход ЦУ)**

**Эти наборы, на которых функция не определена, могут быть (не обязательно) использованы для улучшения результата минимизации функции, т.е. для построения лучшей схемы.**
**Такие наборы в таблице истинности и на картах Вейча (Карно) помечаются (X, *, … и т.д.)**
**Этим наборам можно приписывать или нулевые или единичные наборы.**
**Это позволяет формировать наборы объединяя единичные (нулевые) значения и звёздочки ($f_{днф}$ / $f_{кнф}$)**
**Не обязательно всем звёздочкам приписывать значения.**

## 39. КУБИЧЕСКОЕ ЗАДАНИЕ ФАЛ.

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

Для представления булевых функций в кубической форме используется кубический алфавит {0, 1, x}, основанный на двух логических примитивах 0 и 1, символ x в этом алфавите может рассматриваться как унирверсум, т. е. x = 0 v 1.

Этот символ называется ***свободной*** (независимой) ***координатой***. Он может принимать как нулевое, так и единичное значение, остальные компоненты называются связанными. Куб, содержащий свободные координаты, покрывает кубы, на которых он образован. Количество символов x в кубе определяет его размерность (ранг).
Переменные куба принято называть координатами.

Цена схемы, в общем случае, определяется количеством входов элементов, используемых для ее реализации:

$$C = \sum_{i=1}^{k} (n - r_i) + k,$$

где $k$ – количество полученных кубов; $n - r_i$ – количество единичных и нулевых значений.

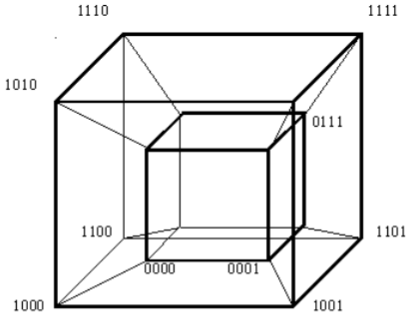
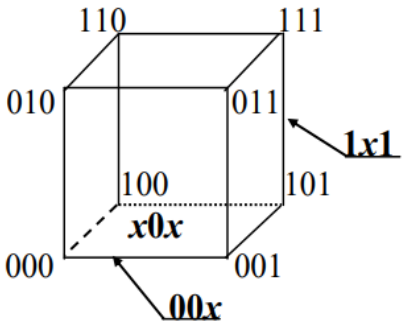



Рис. 24. Геометрическое представление функции четырех переменных

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




***Сложность (цена) по квайну*** определяется суммарным числом входов логических элементов из которых состоит схема. Цена инверсного входа обычно принимается равной двум.
Ранг схемы определяется тем, как связаны элементы, если элемент относится непосредственно к переменным, то это 1 уровень, если логический элемент присоединяется к выходу логического элемента первого уровня, то сам он имеет второй уровень. Максимальный уровень – ***ранг схемы***.

***Вершины – координаты xyz, из рисунка можно понять как задаются. Соседние вершины можно склеивать получая грань и убирая 1 переменную (например останется 00x), грани одной плоскости можно склеить и тем самым убрать 2 переменную (например останется x0x), далее в целом можно и куб склеить, получить xxx :)***

## 40. МИНИМИЗАЦИИ БУЛЕВЫХ ФУНКЦИЙ МЕТОДОМ КВАЙНА-МАК КЛАСКИ.

На первом этапе, рассматриваемого метода, все исходные $n$-кубы разбиваются на непересекающиеся подгруппы по количеству единиц в кубе. Пусть, например, задана функция

$f_{\text{СДНФ}}(x_1x_2x_3x_4) = \text{V}\,(2, 3, 4, 6, 9, 10, 11, 12)$.

Сформируем кубический комплекс $K$, состоящий из кубов нулевой размерности:

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

Выполним разбиение комплекса K на группы, получим

$$K_1^0 = \begin{Bmatrix} 0010 \\ 0100 \end{Bmatrix}, \quad K_2^0 = \begin{Bmatrix} 0011 \\ 1100 \\ 0110 \\ 1001 \\ 1010 \end{Bmatrix}, \quad K_3^0 = \{1011\}.$$

$$K_1^0 \text{ и } K_2^0 \Rightarrow K_1^1 = \begin{Bmatrix} 001x \\ 0x10 \\ x010 \\ x100 \\ 01x0 \end{Bmatrix}, \qquad K_2^0 \text{ и } K_3^0 \Rightarrow K_2^1 = \begin{Bmatrix} x011 \\ 10x1 \\ 101x \end{Bmatrix}.$$

Для рассматриваемого примера сравнение в группах

$K^1_1$ …$K^1_4$ привело к образованию двух новых кубов x01x и x01x и кубов, не образовавших новых {x100, 0x10, 10x1, 01x0}:

$$K_1^2 = \{x01x\}, \qquad K_2^2 = \{x01x\}.$$

Таким образом, получено множество простых импликант $f_{\text{сокр.ДНФ}}$ = {x100, 0x10, 10x1, 01x0, x01x}.

| Простые импликанты | | Минтермы | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 0010 | 0100 | 0011 | 1100 | 0110 | 1001 | 1010 | 1011 |
| *A* | *x*100 | | * | | * | | | | |
| *B* | 0*x*10 | * | | | | * | | | |
| *C* | 10*x*1 | | | | | | * | | * |
| *D* | 01*x*0 | | * | | | * | | | |
| *E* | *x*01*x* | * | | * | | | | * | * |

Следовательно образуются следующие две тупиковые формы:

$f_{\text{МДНФ}}$ = {*x*100, 10*x*1, *x*01*x*, 0*x*10} – 1-я тупиковая форма (*ACEB*);

$f_{\text{МДНФ}}$ = {*x*100, 10*x*1, *x*01*x*, 01*x*0} – 2-я тупиковая форма (*ACED*).

*Замечание*. Если функция не полностью определена, наборы, на которых она не определена, могут участвовать в склеивании, но в импликантную таблицу не вносятся

Для получения МКНФ выполняется формирование кубических комплексов для макстермов, формируется таблица импликант, а тупиковые формы получаются в виде конъюнкции всех существенных импликант и оставшихся простых импликант из набора.

## 41. СИНТЕЗ ОДНОРАЗРЯДНОГО ПОЛНОГО КОМБИНАЦИОННОГО СУММАТОРА.

В зависимости от значений переменных $a_i$, $b_i$, $z_i$ (перенос в $i$-й разряд) формируется значение булевых функций $\text{C}_i$ и $\Pi_i$. Введем следующие обозначения:

$a_i \Rightarrow x$ $\qquad$ $\text{C}_i \Rightarrow$ C , где $\text{C}_i$ – значение суммы в разряде $i$;

$b_i \Rightarrow y$ $\qquad$ $\Pi_i \Rightarrow \Pi$, где $\Pi_i$ – значение переноса из разряда $i$.

$z_i \Rightarrow z$

Таблица 25

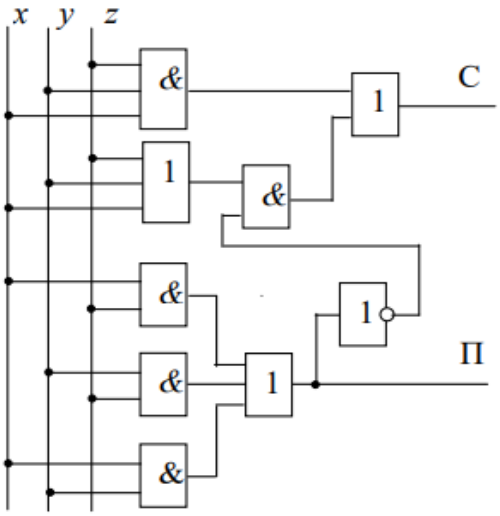
| $x$ | $y$ | $z$ | С | П |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Выполним преобразование функций П и С:

$$П = \bar{x}yz \vee x\bar{y}z \vee xy\bar{z} \vee xyz =$$
$$= \bar{x}yz \vee x\bar{y}z \vee xy\bar{z} \vee xyz \vee xyz \vee xyz =$$
$$= yz \cdot (\bar{x} \vee x) \vee xz(\bar{y} \vee y) \vee xy(\bar{z} \vee z).$$
$$П = yz \vee xz \vee xy.$$
$$С = \bar{x}\bar{y}z \vee \bar{x}y\bar{z} \vee x\bar{y}\bar{z} \vee xyz \vee$$
$$\vee \bar{x}\bar{y}x \vee \bar{x}x\bar{z} \vee y\bar{y}\bar{z} \Leftarrow \text{Логические нули}$$
$$\vee \bar{x}\bar{y}y \vee \bar{x}z\bar{z} \vee z\bar{y}\bar{z} =$$
$$= (x \vee y \vee z)(\bar{x}\bar{y} \vee \bar{x}\bar{z} \vee \bar{y}\bar{z}) \vee xyz,$$

но $\bar{П} = \overline{yz \vee xz \vee xy} = (\bar{x} \vee \bar{y})(\bar{x} \vee \bar{z})(\bar{y} \vee \bar{z}) =$

$$= \bar{x}\bar{y} \vee \bar{x}\bar{z} \vee \bar{y}\bar{z}.$$

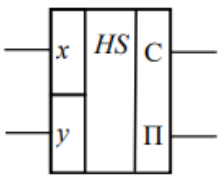



**Правило повтора применяем для П дописывая +xyz+xyz Вынесение за скобки общих множителей, в результате уходят x и не x, y и не y и т.п. Получили П.**

**Т.к. x+1=1, x+0+0=x, то для каждого из трёх слагаемых дописать по 2 с той же инверсией, но меняя переменную без инверсии. Тем самым дописали 6 логических нулей. Получили некое С, сравниваем с П и видим схожесть скобки с точностью до инверсии.**
**Запишем не П. По де моргану раскрываем инверсию, потом ещё 3 инверсии, перемножаем скобки. Получаем например неХнеУ+неУ, неУ глобальнее, она поглощает ту и остаётся, таким образом придём к выражению как в скобке С. Подставляем не П в С.**

## 42. СИНТЕЗ ОДНОРАЗРЯДНОГО КОМБИНАЦИОННОГО ПОЛУСУММАТОРА.

Полусумматор (рис. 44) имеет два входа $x$ и $y$ для разрядов двух слагаемых и два выхода: С – сумма, П – перенос. Таким образом, это устройство предназначено для сложения разрядов двух чисел без учета переноса из предыдущего разряда. Обозначением полусумматора служат буквы *HS* (*half sum* — полусумма).



| $x$ | $y$ | С | П |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

Из таблицы следует что функции П и С имеют вид:

$$П = x\,y,$$

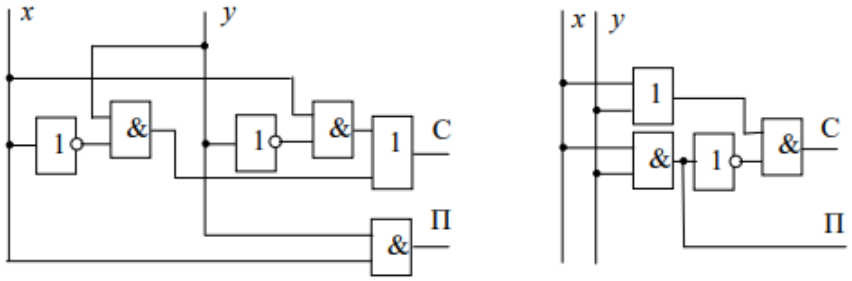
$$С = \bar{x}\,y \vee x\,\bar{y}.$$

Логическая схема, соответствующая записанной системе булевых функций представлена на рис.45.

Данная схема может быть упрощена, если функция С будет записана на нулевых наборах и в ее записи использована функция П. Запись одной функции с участием другой назовем *совместной минимизацей*.

$$\bar{С} = \bar{x}\,\bar{y} \vee x\,y$$

$$С = \overline{\bar{x}\,\bar{y} \vee x\,y} = (x \vee y)\overline{x\,y} = (x \vee y)\cdot П$$




*а* *б*

***2 переменные, базовая сумма, П и С, потом не С, не не С, получаем (x+y)*не П***

## 43. СИНТЕЗ ОДНОРАЗРЯДНОГО ПОЛНОГО КОМБИНАЦИОННОГО СУММАТОРА НА 2 ПОЛУСУММАТОРАХ.

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

$$С = \bar{x}\,\bar{y}z \vee \bar{x}\,y\,\bar{z} \vee x\,\bar{y}\bar{z} \vee x\,y\,z = (\overline{x\,y \vee x\,\bar{y}})\bar{z} \vee (\overline{\bar{x}\,y \vee x\,y})z\,.$$

Введем обозначение $M = x \oplus y$, тогда

$$С = M\bar{z} + \bar{M}\,z = M \oplus z = x \oplus y \oplus z\,.$$

Аналогично можно выполнить преобразование функции переноса П:

$$П = \bar{x}\,yz \vee x\,\bar{y}\,z \vee x\,y\bar{z} \vee x\,y\,z = (\bar{x}\,y \vee x\,\bar{y})z \vee x\,y(\bar{z} \vee z) = (x \oplus y)z \vee xy\,.$$

Схемная реализация полного сумматора на двух полусумматорах и элементе логического ИЛИ будет иметь следующий вид (рис. 48).

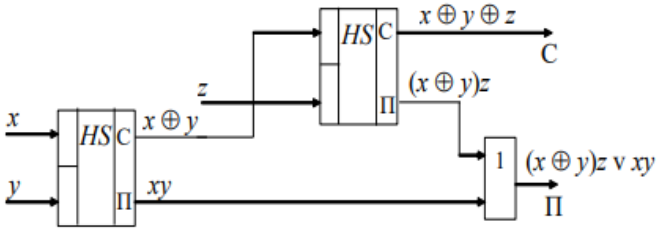



Рис. 48. Схема сумматора на двух полусумматорах и ИЛИ

***С из сумматора, сворачиваем в скобки вынося перенос z и не z. Заменяем на исключающее или и не исключающее или. Преобразовываем и П. С=X xor Y xor Z. В П (X xor Y)*Z + XY.***
***Смотрим на формулы полусумматора, где П для x и y это xy, а С это X xor У***
***На первый полусумматор подаём X и У, на второй С и Z и получаем из них С конечное, П второго полусумматора и первого полусумматора подаём на ИЛИ и получаем итоговый Перенос.***

## 44. СИНТЕЗ ОДНОРАЗРЯДНОГО КОМБИНАЦИОННОГО ВЫЧИТАТЕЛЯ.

В зависимости от значений переменных $a_i$, $b_i$ и $z_i$ ($z_i$ – заем из $i$-го разряда в $i$ + 1-й разряд) формируется значение булевых функций $P_i$ и $З_i$. Введем следующие обозначения:

$a_i \Rightarrow x$ $\quad$ $P_i \Rightarrow$ P, где $P_i$ – значение разности в разряде $i$,

$b_i \Rightarrow y$ $\quad$ $З_i \Rightarrow$ З, где $З_i$ – значение заема в $i$-й разряд из $i$ – 1-го.

$z_i \Rightarrow z$

Таблица истинности, соответствующая устройству, выполняющему операцию вычитания, имеет следующий вид (табл. 26).

Таблица 26

| $x$ | $y$ | $z$ | P | З |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

$З = \bar{x}\,\bar{y}z \vee \bar{x}\,y\,\bar{z} \vee \bar{x}\,y\,z \vee x\,y\,z.$

Выполним склеивания 1 и 3, 2 и 3, 3 и 4 наборов:

$З = \bar{x}\,z \vee \bar{x}\,y \vee y\,z = \bar{x}(z \vee y) \vee y\,z.$

$П = yz \vee x\,z \vee x\,y = x\,(y \vee z) \vee yz.$

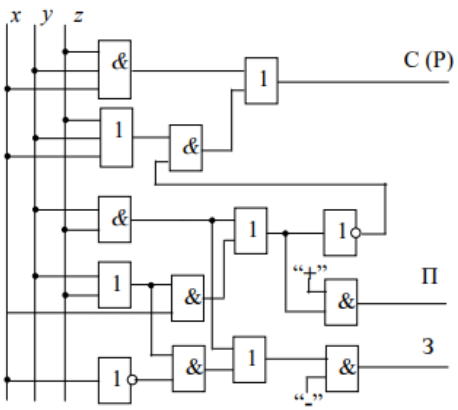
$P = \bar{x}\,\bar{y}z \vee \bar{x}\,y\,\bar{z} \vee x\,\bar{y}\bar{z} \vee x\,y\,z = C.$

Как видно, функции разности P и суммы C совпадают (функция C была получена ранее).

Схемная реализация вычитателя может быть выполна самостоятельно.

**От x отнимаем y и z. При необходимости делаем заём З, получаем ещё и P. З формируем на единицах, потом склеиваем получая 3 слагаемых, можно заметить, что она похожа на П, отличие в инверсии X. P==C.**

## 45. ОБЪЕДИНЕННАЯ СХЕМА ОДНОРАЗРЯДНОГО КОМБИНАЦИОННОГО СУММАТОРА-ВЫЧИТАТЕЛЯ.




**+ и – это сигналы. Если у нас, например, сложение, то посылается единичка на плюсик и формируется сигнал переноса, а заём не формируется, т.к. туда через инвертор приходит нолик.**

## 46. ТРИГГЕР СО СЧЕТНЫМ ВХОДОМ КАК ПОЛНЫЙ ОДНОРАЗРЯДНЫЙ СУММАТОР.

**Первый такт**

1. $\tau(t-1) = 0$ – триггер находится в исходном состоянии.
2. $T = x$ первое слагаемое подается на вход триггера

$\tau(t) = \bar{T}\,\tau(t-1) \vee T\,\bar{\tau}(t-1) = \bar{x} \cdot 0 \vee x \cdot 1 = x.$

После первого такта содержимое триггера соответствует входному сигналу.

**Второй такт**. Во втором такте на вход триггера подается второе слагаемое $y$:

$\tau(t+1) = \bar{T}\,\tau(t) \vee T\,\bar{\tau}(t) = \bar{y} \cdot x \vee y \cdot \bar{x} = x \oplus y,$

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

**Третий такт**. В третьем такте на вход триггера поступает значение, соответствующее третьему слагаемому $z$.

$\tau(t+2) = \bar{T}\,\tau(t+1) \vee T\,\bar{\tau}(t+1) = \bar{y} \cdot x \vee y \cdot \bar{x} = \bar{z}(x \oplus y) \vee z(\overline{x \oplus y}) =$

$= \bar{x}\,\bar{y}z \vee x\,\bar{y}\bar{z} \vee xyz \vee \bar{x}\,y\bar{z}.$

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

Триггер со счетным входом (Т-триггер) может быть рассмотрен как **полный сумматор, работающий в три такта. В основе работы этого устройства лежит операция «сумма по модулю 2».**

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

## 47. СТАНДАРТНЫЕ ФУНКЦИОНАЛЬНЫЕ УЗЛЫ ЦИФРОВОЙ ТЕХНИКИ (МУЛЬТИПЛЕКСОРЫ).

**Мультиплексор –** это устройство, имеющее
- n адресных (управляющих) входов,
- $2^n$ информационных входов.
- 1 выход.

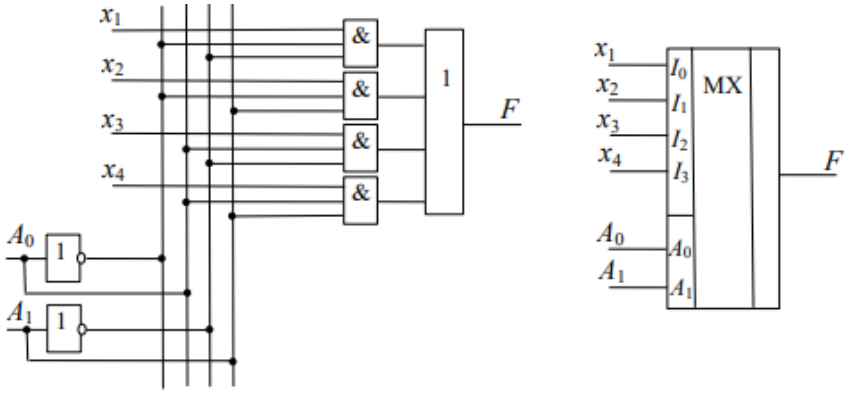
**Принцип работы:** на адресные входы подаётся двоичный *эквивалент* десятичного номера *информационного входа*, значение с которого должно быть сформировано (пропущено, передано) на его выход.

Выбор информационного входа осуществляется подачей соответствующей комбинации (номера этого входа) на управляющие входы мультиплексора

которого должен быть пропущен на выход мультиплексора. Таблица истинности, описывающая работу мультиплексора «1 из 4», имеющего четыре информационных ($x_1x_2x_3x_4$) и два управляющих входа ($A_0A_1$), представлена в табл. 29:

Таблица 29

| $A_0A_1$ | $x_1x_2x_3x_4$ | $F$ |
|---|---|---|
| 0 0 | 0 x x x | 0 |
| 0 0 | 1 x x x | 1 |
| 0 1 | x 0 x x | 0 |
| 0 1 | x 1 x x | 1 |
| 1 0 | x x 0 x | 0 |
| 1 0 | x x 1 x | 1 |
| 1 1 | x x x 0 | 0 |
| 1 1 | x x x 1 | 1 |



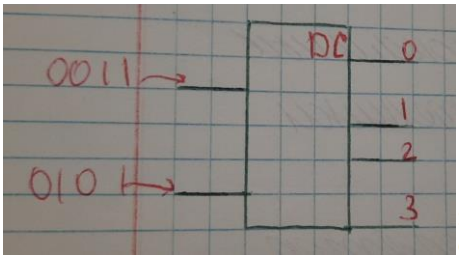
## 48. СТАНДАРТНЫЕ ФУНКЦИОНАЛЬНЫЕ УЗЛЫ ЦИФРОВОЙ ТЕХНИКИ (ДЕШИФРАТОРЫ).

**Дешифратор –** это устройство, имеющее n входов и $2^n$ выходов.
**Принцип работы:** На входы подаётся двоичный эквивалент десятичного номера выхода, на котором должна быть сформирована логическая единица (управляющий сигнал)

Дешифратор преобразует поступающий на ее входы двоичный позиционный код в активный сигнал только на одном из выходов.

При подаче на вход устройства двоичного кода на выходе дешифратора появится сигнал на том выходе, номер которого соответствует десятичному эквиваленту двоичного кода. Отсюда следует, что в любой момент времени выходной сигнал будет иметь место только на одном выходе дешифратора. В зависимости от типа дешифратора, этот сигнал может иметь как уровень логической единицы (при этом на всех остальных выходах уровень логического нуля), так и уровень логического нуля (при этом на всех остальных выходах уровень логической единицы).

Рассмотрим пример синтеза дешифратора (полного), в котором количество разрядов двоичного числа составляет 3, количество выходов – 8.




| $x_1x_2x_3$ | $y_1y_2y_3y_4y_5y_6y_7y_8$ |
|---|---|
| 0 0 0 | 1 0 0 0 0 0 0 0 |
| 0 0 1 | 0 1 0 0 0 0 0 0 |
| 0 1 0 | 0 0 1 0 0 0 0 0 |
| 0 1 1 | 0 0 0 1 0 0 0 0 |
| 1 0 0 | 0 0 0 0 1 0 0 0 |
| 1 0 1 | 0 0 0 0 0 1 0 0 |
| 1 1 0 | 0 0 0 0 0 0 1 0 |
| 1 1 1 | 0 0 0 0 0 0 0 1 |

Из таблицы следует, что для реализации полного дешифратора (рис. 51) на $m$ входов (переменных) потребуются $n = 2^m$ элементов конъюнкции (количество входов каждого элемента равно $m$).